Diagrammer v1 Developer Guide

⚠ This documentation is only for Version 1 of Diagrammer. For version 2 (Kalileo), follow this link

Outline

A step by step dive into the Kap Lab Diagrammer component

Introduction

This documentation present how to integrate diagramming functionality in your AS3 application:

Adding a SVG library in a flex application
Adding layout to a diagram
Binding data to a diagram
Sprites composition in a diagram
We use Flex for this tutorial, but because the Diagrammer is a pure AS3 component, it can also be integrated into Flash AS3 applications.

Keep in mind that the samples used in this documentation are neither meant to be useful nor in phase with software development quality standards. For the sake of simplicity and to keep focus on the original goal which is showing how to integrate the Diagrammer, we tried to reduce the amount of code artifact. So we put all the code we can into a single mxml file, and created as little class as possible, using generic data container such as ArrayCollections. In real world software, you should use a MVC framework, create specific class to hold your data, and so on.

You can download a complete solution with all the code from this documentation if you don't want to type.

Adding a SVG library in a flex application

In this chapter we will show how to write a very simple application and with a few lines of code add a diagramming functionality.

First we need to create the sprite library that will be used in the application.

Creating a SVG shape library

Overview

Diagrammer uses SVG to define sprites. Using SVG helps easily integrate complex shapes into Diagrammer. It also gives the opportunity to export a diagram in SVG for printing.

Diagrammer supports a subset of SVG 1.1. Most of the features needed for sprite definitions are supported :

Document structure (<svg>, <defs>, <g>).
Basic shapes (<rect>, <circle>, <ellipse>, <line>, <polyline>, <polygon>).
Full support of drawing paths with: line, elliptic arcs, cubic Bezier curves, quadratic Bezier curves commands.
SVG text features are partially implemented and match the text rendering capabilities of the flash player (<text>, <tspan>)
Full support of transformations: translate, scale, rotate, skewX, skewY, matrix.
Styling using simple and complex CSS selectors is supported.
Gradients and gradients linking definitions are supported.

Sprite definition

To define sprites, Diagrammer uses extensions to standard SVG. They are defined in the namespace "http://schemas.kapit.fr/svg/2007/". You must add to the <svg> tag in your library the following attribute : xmlns:k="http://schemas.kapit.fr/svg/2007/" .
Each sprite must be defined as a group (<g> tag), with the followings attributes :

k:spriteid defines the identifier of the sprite. It is used in Diagrammer as a unique identifier for a specific shape.
k:groupid defines the group of the sprite. Diagrammer provides a method to retrieve all sprite of a given group. This functionality can use by components (drop down menu, panels) that enable user to select any sprite from a given

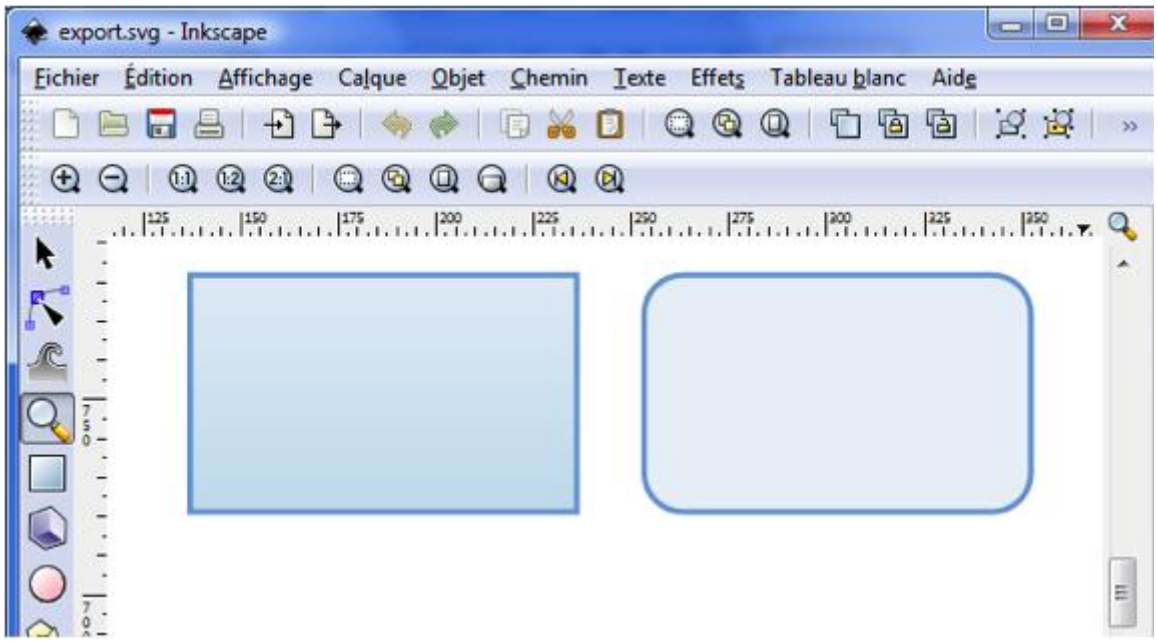
group.

Actions and anchor point definition

Using css styling, we will define how can the user interact with each sprite.
To enable input links to a part of a sprite, we must add the style action-accept:link to this part (it can be a group or any basic drawing instruction).
To enable output links for a part of a sprite, we must add the style action-click:link to this part.
To enable annotation for a part of a sprite, we must add the action-accept:annotation to this part. We usually put this in the group that defines the sprite.
As a basic rule, it is better to add a dedicated transparent path to define the link anchors, but depending on the sprite we can use part of sprite.

Putting it together

In this sample we will use a two sprite library. We define two basic shapes, a rectangle and a rounded rectangle.
Using a vector based image editor that support SVG export (such as Inkscape), we can define more complex sprite. Be sure to remove all non standard tags from the SVG library before using it in Diagrammer.
We will use a SVG file generated by Inkspace to create a simple rectangle with a linear gradient and a rounded rectangle :



The generated files look like this :

Two Sprites Library Creation sample

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:cc="http://creativecommons.org/ns#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/2000/svg"
```

```
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
 xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
 version="1.0"
 width="1024"
 height="800"
 id="sgvlib"
 sodipodi:version="0.32"
 inkscape:version="0.46"
 sodipodi:docname="export.svg"
 inkscape:output_extension="org.inkscape.output.svg.inkscape">
<metadata
   id="metadata2457">
  <rdf:RDF>
    <cc:Work
       rdf:about="">
      <dc:format>image/svg+xml</dc:format>
      <dc:type
         rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
    </cc:Work>
  </rdf:RDF>
</metadata>
<sodipodi:namedview
   inkscape:window-height="669"
   inkscape:window-width="640"
   inkscape:pageshadow="2"
   inkscape:pageopacity="0.0"
   guidetolerance="10.0"
   gridtolerance="10.0"
   objecttolerance="10.0"
   borderopacity="1.0"
   bordercolor="#666666"
   pagecolor="#ffffff"
   id="base"
   showgrid="false"
   inkscape:zoom="0.53125"
   inkscape:cx="512"
   inkscape:cy="400"
   inkscape:window-x="520"
   inkscape:window-y="1"
   inkscape:current-layer="sgvlib" />
<defs
   id="defs1">
  <inkscape:perspective
     sodipodi:type="inkscape:persp3d"
     inkscape:vp_x="0 : 400 : 1"
     inkscape:vp_y="0 : 1000 : 0"
     inkscape:vp_z="1024 : 400 : 1"
     inkscape:persp3d-origin="512 : 266.66667 : 1"
     id="perspective2459" />
  <linearGradient
     id="linearGradient3157">
    <stop
       id="stop3159"
       style="stop-color:#e1eaf5;stop-opacity:1"
       offset="0" />
    <stop
       id="stop3161"
       style="stop-color:#c3daea;stop-opacity:1"
```

```
         offset="1" />
    </linearGradient>
    <linearGradient
       xlink:href="#linearGradient3157"
       id="linearGradient6627"
       gradientUnits="userSpaceOnUse"
       gradientTransform="translate(98.840748,-67.175132)"
       x1="94.95433"
       y1="85.235107"
       x2="94.95433"
       y2="146.25534" />
  </defs>
  <g
     id="grectangle">
    <rect
       id="rect3155"
       style="fill:url(#linearGradient6627);fill-opacity:1;stroke:#6996cf;stroke-width:1.5;stroke-
linecap:round;stroke-linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       x="137.22656"
       y="18.47105"
       rx="0"
       ry="0"
       width="98.994949"
       height="60.609154" />
    <path
       class="bg"
       id="rect31551"
       style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
       d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
  <g
     transform="translate(116.20044,0)">
    <rect
       style="fill:#e1eaf5;fill-opacity:0.8;stroke:#6996cf;stroke-width:1.5;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       id="rect3157"
       y="18.47105"
       x="137.22656"
       rx="10"
       ry="10"
       height="60.609154"
       width="98.994949" />
    <path
       class="bg"
       id="rect31552"
       style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
       d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
</svg>
```

We first clean the files, remove non standard tag and namespaces (sodipodi, inscape, purl, cc, rdf).

Non Standard tags and namespaces Remove Sample

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
```

```
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   version="1.0"
   width="1024"
   height="800"
   id="sgvlib">
  <defs
     id="defs1">
    <linearGradient
       id="linearGradient3157">
      <stop
         id="stop3159"
         style="stop-color:#e1eaf5;stop-opacity:1"
         offset="0" />
      <stop
         id="stop3161"
         style="stop-color:#c3daea;stop-opacity:1"
         offset="1" />
    </linearGradient>
    <linearGradient
       xlink:href="#linearGradient3157"
       id="linearGradient6627"
       gradientUnits="userSpaceOnUse"
       gradientTransform="translate(98.840748,-67.175132)"
       x1="94.95433"
       y1="85.235107"
       x2="94.95433"
       y2="146.25534" />
  </defs>
  <g
     id="grectangle">
    <rect
       id="rect3155"
       style="fill:url(#linearGradient6627);fill-opacity:1;stroke:#6996cf;stroke-width:1.5;stroke-
linecap:round;stroke-linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       x="137.22656"
       y="18.47105"
       rx="0"
       ry="0"
       width="98.994949"
       height="60.609154" />
    <path
       class="bg"
       id="rect31551"
       style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
       d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
  <g
     transform="translate(116.20044,0)">
    <rect
       style="fill:#e1eaf5;fill-opacity:0.8;stroke:#6996cf;stroke-width:1.5;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       id="rect3157"
       y="18.47105"
       x="137.22656"
       rx="10"
       ry="10"
```

```
        height="60.609154"
        width="98.994949" />
    <path
        class="bg"
        id="rect31552"
        style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
        d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
</svg>
```

We then put the css part into the definition part :

CSS Integration Sample

```
  ...<defs id="defs1">
    <style type="text/css">
      <![CDATA[
.basic {
action-accept:annotation;
}
.bg {
action-click:link;
action-accept:link;
}
.annotation {
font-style:italic;
}
.link-annotation
{
        font-size:12px;
        fill:#666;
}
]]>
    </style>
    <linearGradient id="linearGradient3157">...
```

We defined basic and bg style. Basic style will be applied to a sprite to enable annotations. Bg style will be applied to anchor part.
The system style annotation and link-annotation are overloaded to fit our taste.
Then for each sprite we modify the group to set spriteid and groupid.

Basic Style Integration sample

```
  ...<g
    k:spriteid="rectangle"
    k:groupid="Basic"
    class="basic"
    id="grectangle">...
  ...<g
    k:spriteid="rounded-rectangle"
    k:groupid="Basic"
    id="grrectangle"
    class="basic" transform="translate(116.20044,0)">...
```

Now the library is ready. We can open it with any SVG editor to check that it still renders as intended (but we shouldn't export it again to avoid adding unwanted tags).

Here is the SVG file (without the <?xml header) :

SVG Output Sample

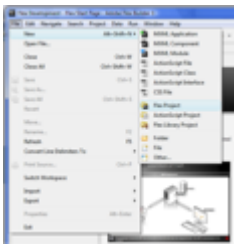
```
<svg xmlns:svg="http://www.w3.org/2000/svg" xmlns:k="http://schemas.kapit.fr/svg/2007/"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg" version="1.0" width="1024"
height="800" id="sgvlib">
  <defs id="defs1">
    <style type="text/css">
      <![CDATA[
.basic {
action-accept:annotation;
}
.bg {
action-click:link;
action-accept:link;
}
.annotation {
font-style:italic;
}
.link-annotation
{
        font-size:12px;
        fill:#666;
}
]]>
    </style>
    <linearGradient id="linearGradient3157">
      <stop id="stop3159" style="stop-color:#e1eaf5;stop-opacity:1" offset="0" />
      <stop id="stop3161" style="stop-color:#c3daea;stop-opacity:1" offset="1" />
    </linearGradient>
    <linearGradient xlink:href="#linearGradient3157" id="linearGradient6627"
gradientUnits="userSpaceOnUse" gradientTransform="translate(98.840748,-67.175132)"
x1="94.95433" y1="85.235107" x2="94.95433" y2="146.25534" />
  </defs>
  <g
    k:spriteid="rectangle"
    k:groupid="Basic"
    class="basic"
    id="grectangle">
    <rect
       id="rect3155"
       style="fill:url(#linearGradient6627);fill-opacity:1;stroke:#6996cf;stroke-width:1.5;stroke-
linecap:round;stroke-linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       x="137.22656"
       y="18.47105"
       rx="0"
       ry="0"
       width="98.994949"
       height="60.609154" />
    <path
       class="bg"
       id="rect31551"
       style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
       d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
  <g
    k:spriteid="rounded-rectangle"
```

```
    k:groupid="Basic"
    id="grrectangle"
    class="basic">
    <rect
      style="fill:#e1eaf5;fill-opacity:0.8;stroke:#6996cf;stroke-width:1.5;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       id="rect3157"
       y="18.47105"  x="137.22656"
       rx="10"  ry="10"
       height="60.609154"  width="98.994949"  />
    <path
        class="bg"
        id="rect31552"
        style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
        d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z"  />
  </g>
</svg>
```

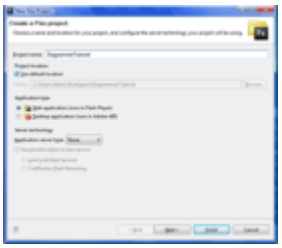
Now let's write some code.

Creating the Flex project

Launch Flex Builder and choose New / Flex Project



Enter a name for your Project ("DiagrammerTutorial" for instance). Select Web Application and none for application server type. Press Finish to accept all default options



From the folder where you downloaded it, drag the Diagrammer.swc file into the libs folder in the flex navigator.

Type the following code in the DiagrammerTutorial.mxml file :

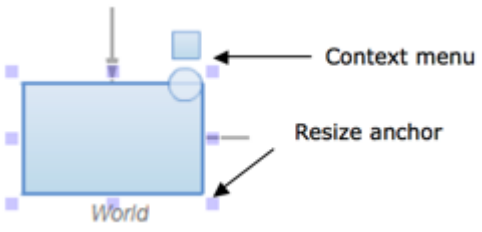
Basic Integration Tutorial

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
```

```
        xmlns:diagctl="com.kapit.diagram.controls.*"
        xmlns:diagview="com.kapit.diagram.view.*"
        layout="absolute" preinitialize="init();">

<mx:XML xmlns="" id="svglib">
Paste your svg library in this area...
</mx:XML>
<mx:Script>
        <![CDATA[
                import com.kapit.diagram.library.SVGAssetLibrary;


                public function init():void
                {
                        var lib:SVGAssetLibrary=new
SVGAssetLibrary(svglib);
                }
                public function initDiagram():void
                {
                                diagram.multipanel=false;
                                diagram.selectionenabled=true;
                                diagram.keyboardenabled=true;
                                diagram.dragenabled=true;
                                diagram.dropenabled=true;
                }
        ]]>
</mx:Script>
<mx:ApplicationControlBar width="100%" height="50" horizontalAlign="right">
<diagctl:SVGAssetLibraryGroupButton width="150" groupid="Basic"
cornerRadius="10" paddingLeft="8" paddingBottom="0" paddingTop="0"
useHandCursor="true" toolTip="Drag &amp;amp; drop" labelPlacement="right"
textAlign="left"/>
</mx:ApplicationControlBar >
<mx:HBox left="10" top="60" bottom="10" right="10" horizontalGap="10">
<diagview:DiagramView width="100%" height="100%" id="diagram"
creationComplete="initDiagram();">
</diagview:DiagramView>
</mx:HBox>
</mx:Application>
```

We removed the SVG part to show that you don't have to write lots of code. The svg library can be found on the previous section of this documentation.
Save the file, compile the project and run the application :

Drag shapes from the upper right menu to the main view. Clicking on an shape shows a upper right graphical menu that enables you to create new objects linked to the selected one.



Now let's explain the code in detail.

Detailed explanation

On the Application tag we add reference to the view and controls of the diagrammer. We also register a preinitialization function.

On the Application tag we add reference to the view and controls of the diagrammer. We also register a preinitialization function.

Preinitialization Tutorial

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
        xmlns:diagctl="com.kapit.diagram.controls.*"
        xmlns:diagview="com.kapit.diagram.view.*"
        layout="absolute" preinitialize="init();">
```

We then include the svg library in an xml tag.

Basic Integration sample

```
<mx:XML xmlns="" id="svglib">
<svg xmlns:svg="http://www.w3.org/2000/svg" ...
...
...
</svg>
</mx:XML>
```

Including the svg library in the source is not mandatory. For example you can use an URLLoader to load the library from a svg file somewhere on the network.
In the init fonction we simply load the library from the xml. This must be done before the DiagramView object is created.

SVG Including in XML Tag sample

```
        import com.kapit.diagram.library.SVGAssetLibrary;


        public function init():void
        {
                var lib:SVGAssetLibrary=new SVGAssetLibrary(svglib);
        }
```

The application layout is simple. We use an ApplicationControlBar to hold a SVGAssetLibraryGroupButton, and an HBox to hold a DiagramView.

SVGAssetLibraryGroupButton Integration Tutorial

```
<mx:ApplicationControlBar  width="100%" height="50" horizontalAlign="right">
<diagctl:SVGAssetLibraryGroupButton width="150" groupid="Basic"
cornerRadius="10" paddingLeft="8" paddingBottom="0" paddingTop="0"
useHandCursor="true" toolTip="Drag &amp;amp; drop" labelPlacement="right"
textAlign="left"/>
</mx:ApplicationControlBar>
<mx:HBox left="10" top="60" bottom="10" right="10" horizontalGap="10">
<diagview:DiagramView width="100%" height="100%" id="diagram"
creationComplete="initDiagram();">
</diagview:DiagramView>
```

```
</mx:HBox>
</mx:Application>
```

The SVGAssetLibraryGroupButton groupid property is set to "Basic", wich is the value used in our k:groupid attributes for our sprites in the library.

Basic SVGAssetLibraryGroupButton Setting Tutorial

```
<diagctl:SVGAssetLibraryGroupButton width="150" groupid="Basic"
```

The DiagramView object registers an initDiagram function

InitDiagram Function Registration Tutorial

```
<diagview:DiagramView width="100%" height="100%" id="diagram"
creationComplete="initDiagram();">
```

The initDiagram function sets some diagram's options.

initDiagram Function Options Setting Tutorial

```
        public function initDiagram():void
        {
                        diagram.multipanel=false;
                        diagram.selectionenabled=true;
                        diagram.keyboardenabled=true;
                        diagram.dragenabled=true;
                        diagram.dropenabled=true;
        }
```

Multipanel is set to false so that the diagram doesn't support multipanel view. This option allows the diagram to be divided in verticals panels and lanes.
SelectionEnabled is set to true, so that objects can be selected on the diagram.
KeyboardEnabled is set to true, so that you can scroll with keyboard, enter text, etc.
DragEnabled enables dragging of object in the diagram.
DropEnabled enables dropping of objects in the diagram.

Adding layout to a diagram

In this part we will add layout functionality to our diagramming application. Diagrammer layouts are powerful algorithms that automatically place sprites and links of a diagram

Adding layout library

From the folder where you downloaded it, drag DiagramProxies.swc to the libs folder in the Flex Navigator.

Adding code

In the script part of DiagrammerTutorial.mxml, add a reference to the proxy.

Proxy Reference Integration Tutorial

```
        import com.kapit.diagram.library.SVGAssetLibrary;

        import com.kapit.diagram.proxies.KDLProxy;
        import com.kapit.diagram.layouts.utils.Constants;
        private var proxy:KDLProxy;

        public function init():void
        {
```

In the initDiagram function, add the initialization of the proxy. Proxy initialization must be made after diagram creation.

Proxy Initialization Tutorial

```
                diagram.dropenabled=true;

                proxy = new KDLProxy(diagram);
                proxy.importGraph();
        }
```

In the ApplicationControlBar, add a button that calls the doRadialLayout function on click.

Basic Integration sample

```
<mx:ApplicationControlBar  width="100%" height="50" horizontalAlign="right">
<mx:Button id="radialLayout" label="Radial Layout" click="doRadialLayout();"/>
<diagctl:SVGAssetLibraryGroupButton width="150" groupid="Basic"
cornerRadius="10" paddingLeft="8" paddingBottom="0" paddingTop="0"
useHandCursor="true" toolTip="Drag &amp;amp; drop" labelPlacement="right"
textAlign="left"/>
```

Add the doRadialLayout function at the end of the script part

RadialLayout Integration Code

```
        }
        public function doRadialLayout():void
        {
                proxy.radialLayout.nodesSpacing = 10;
                proxy.exportGraph(Constants.RADIAL_LAYOUT);
        }
        ]]>
</mx:Script>
```

Running the test

Build the application and launch it.

Create linked sprites.

Click on the Radial Layout button and let the magic happens...

More layouts

Now your application supports the radial layout. The Diagrammer has many more built-in layouts such as hierarchical, radial, balloon, organic layouts.
Let's add the animated organic layout to your application.
Add a new button to the Application control bar that calls the doAnimatedLayout function when clicked.

Animated Layout Call Code

```
<mx:ApplicationControlBar  width="100%" height="50" horizontalAlign="right">
<mx:Button id="animatedLayout" label="Animated Layout" click="doAnimatedLayout();"/>
<mx:Button id="radialLayout" label="Radial Layout" click="doRadialLayout();"/>
```

Add the doAnimatedLayout function at the end of the script part

Animated Layout Integration Code

```
        public function doAnimatedLayout():void
        {

        proxy.exportGraph(Constants.ANIMATED_FORCEDIRECTED_LAYOUT);
        }
        ]]>
```

Compile and launch the application. Create linked objects, click Animated Layout button, and drag one object. You will see all the other objects following your object in a nice animation.

Binding data to a diagram

In this part we will add data binding between the diagram view and application data. This enables developers to propose alternative views of sprites and links of the diagram. Using flex bindings any change in property of sprites and links in the diagram are automatically reported in alternative views.

Data binding from diagram objects to application data uses proxies. To enable binding, a developer must create sprite proxy classes implementing ISpriteProxy interface and link proxy classes implementing ILinkProxy interface. Then he must create a configuration file that links spriteid, links and proxies. Then he must create a DiagramModel with the configuration file and define it as the model of the DiagramView.

When this plumbery is done, everytime a sprite or a link is created, deleted or changed, the corresponding method of the corresponding proxy is called. It is the responsibility of the developer to implements application data manipulation so that application data reflects the states of the diagram.

In this tutorial we will use only one proxy for sprites and one for links. For real world application, you can have specific proxy for some sprites, or have several sprites sharing the same proxy.

Writing data object

From the Flex Navigator src folder, select New / ActionScript Class. Name it MyObject.

Add the following code:

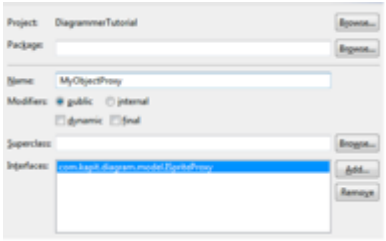
Data Object Definition

```
package
{
	public class MyObject
	{
		[Bindable]
		public var type:String;

		[Bindable]
		public var spriteid:String;

		[Bindable]
		public var did:String;

		[Bindable]
		public var name:String;

		[Bindable]
		public var uid:String;

		public function MyObject()
		{
		}

	}
```

We define all attributes as bindable to display them in a DataGrid with no code.

Writing objects proxy

From the Flex Navigator src folder, select New / ActionScript Class. Name it MyObjectProxy, and add ISpriteProxy as an interface. This way all methods will be automaticaly created by Flex builder for you.



Add the following code:

Object Proxy Integration Tutorial

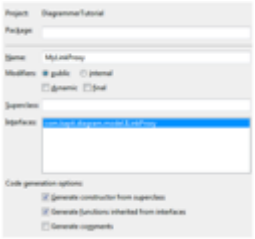
```
package
{
	import com.kapit.diagram.IDiagramElement;
	import com.kapit.diagram.layers.DiagramLane;
	import com.kapit.diagram.model.ISpriteProxy;
	import com.kapit.diagram.view.DiagramSprite;
```

```
        import com.kapit.diagram.view.DiagramView;

        import mx.collections.ArrayCollection;
        import mx.utils.UIDUtil;

        public class MyObjectProxy implements ISpriteProxy
        {
                public static var _objects:ArrayCollection = null;

                protected var _view:DiagramView;

                public function MyObjectProxy(view:DiagramView)
                {
                        _view = view;
                }
                protected function getElementIndex(el:IDiagramElement):int
                {
                        if (el.dataobjectid &&  objects)
                        {
                                for(var i:int=0; i < _objects.length; i++)
                                {
                                        var obj:Object =
_objects.getItemAt(i);
                                        if (obj.uid == el.dataobjectid)
                                        {
                                                return i;
                                        }

                                }
                        }
                        return -1;
                }
                public function createDataObject(el:IDiagramElement):String
                {
                        var type:String = el.getTagName();
                        var spriteid:String = DiagramSprite(el).spriteid;
                        var name:String = el.did;
                        var uid:String = UIDUtil.createUID();
                        var obj:MyObject = new MyObject();
                        obj.type = type;
                        obj.spriteid = spriteid;
                        obj.did = el.did;
                        obj.uid = uid;
                        obj.name = "";
                        if (_objects)
                                _objects.addItem(obj);
                        return uid;
                }

                public function removeDataObject(el:IDiagramElement):void
                {
                        var index:int = getElementIndex(el);
                        if (index != -1)
                                _objects.removeItemAt(index);
                }

                public function allowLinkAction(el:IDiagramElement):Boolean
                {
                          return true;
                }
```

```
                public function propertyModified(el:IDiagramElement,
propname:String, propvalue:Object, shapeid:String):void
                {
                        var index:int = getElementIndex(el);
                        var obj:MyObject = null;
                        if (index != -1)
                        {
                                obj = _objects.getItemAt(index) as
MyObject;
                        }

                        if ("text" == propname)
                        {
                                if (obj)
                                {
                                        obj.name=String(propvalue);
                                }
                        }
                }

                public function preAcceptLinkSource(spriteid:String,
sourcespriteid:String, el:IDiagramElement):Boolean
                {
                        return true;
                }

                public function preAcceptLinkTarget(spriteid:String,
targetspriteid:String, el:IDiagramElement):Boolean
                {
                        return true;
                }

                public function dataObjectPropertyModified(uid:String,
propname:String, propvalue:Object):void
                {
                }

                public function acceptLinkTarget(el:IDiagramElement,
target:IDiagramElement):Boolean
                {
                        return true;
                }

                public function dataObjectRemoved(uid:String):void
                {
                }

                public function dataObjectLoaded(el:IDiagramElement):void
                {
                }


                public function acceptLinkSource(el:IDiagramElement,
source:IDiagramElement):Boolean
                {
                        return true;
                }

                public function laneChanged(el:IDiagramElement,
lane:DiagramLane):void
                {
```

```
                }

                public function
acceptPropertyModification(el:IDiagramElement, propname:String,
propvalue:Object, shapeid:String):Boolean
                {
                        return true;
                }

                public function
acceptRemoveObject(el:IDiagramElement):Boolean
                {
                        return true;
                }
        }
}
```

From the Flex Navigator src folder, select New / ActionScript Class. Name it MyLinkProxy, and add ILinkProxy as an interface. This way all methods will be automatically created by Flex builder for you.



Add the following code:

Link Proxy Integration Tutorial

```
package
{
        import com.kapit.diagram.IDiagramElement;
        import com.kapit.diagram.model.ILinkProxy;
        import com.kapit.diagram.view.DiagramLink;
        import com.kapit.diagram.view.DiagramView;

        import mx.collections.ArrayCollection;
        import mx.utils.UIDUtil;

        public class MyLinkProxy implements ILinkProxy
        {
                public static var _links:ArrayCollection = null;

                protected var _view:DiagramView;

                public function MyLinkProxy(view:DiagramView)
                {
                        _view = view;
                }

                public function createDataObject(el:IDiagramElement):String
                {
                        var type:String = el.getTagName();
                        var sourceid:String =
DiagramLink(el).sourceobject.dataobjectid;
                        var targetid:String =
DiagramLink(el).targetobject.dataobjectid;
                        var uid:String = UIDUtil.createUID();
```

```
                        var obj:Object = new Object();
                        obj.start = sourceid;
                        obj.end = targetid;
                        obj.uid = uid;
                        if (_links)
                                _links.addItem(obj);
                        return uid;
                }

                public function scopeChanged(link:DiagramLink,
oldscope:String):void
                {
                }

                public function removeDataObject(el:IDiagramElement):void
                {
                        if (el.dataobjectid && _links)
                        {
                                for(var i:int=0; i < _links.length; i++)
                                        if (_links.getItemAt(i).uid ==
el.dataobjectid)
                                        {
                                                _links.removeItemAt(i);
                                                break;
                                        }
                        }
                }

                public function propertyModified(el:IDiagramElement,
propname:String, propvalue:Object, shapeid:String):void
                {
                }

                public function dataObjectPropertyModified(uid:String,
propname:String, propvalue:Object):void
                {
                }

                public function dataObjectRemoved(uid:String):void
                {
                }

                public function dataObjectLoaded(el:IDiagramElement):void
                {
                }

                public function
acceptPropertyModification(el:IDiagramElement, propname:String,
propvalue:Object, shapeid:String):Boolean
                {
                        return true;
                }

                public function
acceptRemoveObject(el:IDiagramElement):Boolean
                {
                        return true;
                }

        }
```

```
}
```

Adding mappings

In the DiagrammerTutorial.mxml file, add a xml document holdings mappings.

Mapping Adding Code

```
</mx:XML>
<mx:XML xmlns="" id="diagramMappings">
        <mappings>
                <sprite id="rectangle" width="40" height="40"
proxyclass="MyObjectProxy" />
                <sprite id="rounded-rectangle" width="40" height="40"
proxyclass="MyObjectProxy" />
                <lane proxyclass="" />
                <panel proxyclass="" />
                <lanelink proxyclass="MyLinkProxy" />
                <panellink proxyclass="MyLinkProxy" />
                <viewlink proxyclass="MyLinkProxy" />
        </mappings>
</mx:XML>
<mx:Script>
```

Add imports and application data definition. In this tutorial we will use ArrayCollection as an example, as we don't want to multiply class definitions.

Imports and Application Definition Code

```
        private var proxy:KDLProxy;
        import com.kapit.diagram.model.DiagramModel;
        import MyObjectProxy;
        import MyLinkProxy;
        import mx.collections.ArrayCollection;

        [Bindable]
        public var myObjects:ArrayCollection = new ArrayCollection();

        [Bindable]
        public var myLinks:ArrayCollection = new ArrayCollection();

        public function init():void
```

In the initDiagram function, set the proxy ArrayCollection and set the diagram model.

Setting ArrayCollection Proxy and Diagram Model Sample

```
diagram.dropenabled=true;

MyObjectProxy._objects = myObjects;
MyLinkProxy._links = myLinks;
var model:DiagramModel=new DiagramModel(diagramMappings);
diagram.model=model;

proxy = new KDLProxy(diagram);
```

Data bindings are now activated. You can test it by debugging your application and setting breakpoint in MyObjectProxy and MyLinkProxy methods. You will see that the proxy object is created when the first diagram object is created, and that createDataObject function is called each time an object is created.

The code you wrote add an Object in the myOBjects and myLinks ArrayCollections and keep them synchronized with

the diagram sprites and links.

Now we will add some code to use these ArrayCollections.

Displaying application data

We change the application layout by adding two DataGrids. One will display the sprites and the other the links.

DataGrids Integration Code

```
</mx:ApplicationControlBar>
<mx:HBox left="10" top="50" bottom="10" right="10" horizontalGap="10">
<mx:Panel width="50%" height="100%" layout="absolute" title="Diagram"
id="diagramPanel">
<diagview:DiagramView id="diagram" creationComplete="initDiagram();" >
</diagview:DiagramView>
</mx:Panel>
<mx:VBox width="50%" height="100%">
        <mx:Panel id="objectPanel" width="100%" height="50%"  title="Objects">
                <mx:DataGrid id="objectsGrid" dataProvider="{myObjects}"
width="100%" height="100%" >
                        <mx:columns>
                                <mx:DataGridColumn headerText="Type"
dataField="spriteid" />
                                <mx:DataGridColumn headerText="Sprite"
dataField="did" />
                                <mx:DataGridColumn headerText="ID"
dataField="uid" />
                                <mx:DataGridColumn headerText="Name"
dataField="name" />
                        </mx:columns>
                </mx:DataGrid>

        </mx:Panel>
        <mx:Panel id="linksPanel" width="100%" height="50%"  title="Links">
                <mx:DataGrid id="linksGrid" dataProvider="{myLinks}" width="100%"
height="100%" >
                        <mx:columns>
                                <mx:DataGridColumn headerText="Start"
dataField="start"/>
                                <mx:DataGridColumn headerText="End"
dataField="end"/>
                        </mx:columns>
                </mx:DataGrid>
        </mx:Panel>
</mx:VBox>
</mx:HBox>
</mx:Application>
```

Build and run the application. Add sprites and links to the diagram, you will see them appear on the corresponding grid. Add annotation to objects and see them reflected in the name column.

Now we have two DataGrids connected to the Diagram. But when you select a sprite or a link, nothing is selected on

the grids. And when you select a row on a grid nothing get selected on the Diagram. Let's fix this.

Synchronizing selection

First add some imports

Basic Import Code

```
import mx.collections.ArrayCollection;
import com.kapit.diagram.DiagramEvent;
import com.kapit.diagram.view.DiagramObject;
import mx.events.ListEvent;
[Bindable]
public var myObjects:ArrayCollection = new ArrayCollection();
```

Register to the selection changed of the diagram in the initDiagram function

Selection Changed Registration Code

```
diagram.model=model;

        diagram.addEventListener(DiagramEvent.SELECTION_CHANGED,handleDi
agramSelectionChanged);

proxy = new KDLProxy(diagram);
```

Add the function to handle selection on the diagram and select the corresponding row on the correct grid.

Diagram Selection and DataGrid Row Correspondency Code

```
private function handleDiagramSelectionChanged(e:Event):void
{
        var arr:Array=diagram.getSelectedObjects();
        var
uid:String=(arr&&arr.length==1)?DiagramObject(arr[0]).dataobjectid:null;

        if(uid)
        {
                var found:Boolean=false;
                for(var i:int=0;i<myObjects.length;i++)
                        {
                        if (myObjects.getItemAt(i).uid == uid)
                        {
                                objectsGrid.selectedIndex = i;
                                found = true;
                                break;
                        }
                }
                if (!found)
                {
                        for(i=0;i< myLinks.length;i++)
                        {
                                if (myLinks.getItemAt(i).uid == uid)
                                {
                                        linksGrid.selectedIndex = i;
                                        found = true;
                                        break;
                                }
                        }
                }
        }
}
```

Now we have one way selection synchronization : when an object is selected on the diagram the corresponding row get selected on the grid. We need to add the other way.

Register a function on the itemClick event for the two DataGrid :

Item Click Event Integration Code

```
<mx:DataGrid id="objectsGrid" dataProvider="{myObjects}"
itemClick="handleObjectSelected(event);" width="100%" height="100%" >
...
<mx:DataGrid id="linksGrid" dataProvider="{myLinks}"
itemClick="handleLinkSelected(event);" width="100%" height="100%" >
```

Add the function in the script part

Item Click Handler Description Code

```
private function handleObjectSelected(event:ListEvent):void
{
        var uid:String = myObjects.getItemAt(event.rowIndex).uid;
        var dob:DiagramObject =
DiagramObject(diagram.getElementByDataObjectId(uid));
        diagram.deselectAll();
        diagram.selectObject(dob);
}
private function handleLinkSelected(event:ListEvent):void
{
        var uid:String = myLinks.getItemAt(event.rowIndex).uid;
        var dob:DiagramObject =
DiagramObject(diagram.getElementByDataObjectId(uid));
        diagram.deselectAll();
        diagram.selectObject(dob);
}
```

And that's it. Compile and run the application, create sprites and links, click on a diagram object and grid row and see as all is kept in sync.

Now the last improvement we can add is the ability to change graph object property from user input outside of the diagram view.

Graph object modification

First let's make the name field editable.

Enabling Name Field Edition Code

```
                <mx:DataGrid id="objectsGrid" dataProvider="{myObjects}"
itemEditEnd="handleEditEnd(event);" itemClick="handleObjectSelected(event);" width="100%"
height="100%" editable="true" >
                        <mx:columns>
                                <mx:DataGridColumn headerText="Type"
dataField="spriteid" editable="false"/>
                                <mx:DataGridColumn headerText="Sprite"
dataField="did" editable="false"/>
                                <mx:DataGridColumn headerText="ID"
dataField="uid" editable="false"/>
                                <mx:DataGridColumn headerText="Name"
dataField="name" editable="true"/>
                        </mx:columns>
                </mx:DataGrid>
```

Add some imports

Basic Import Code

```
import mx.events.ListEvent;
import mx.controls.TextInput;
import mx.events.DataGridEvent;
[Bindable]
```

Then type the function handleEditEnd

Edit End Handler Integration Code

```
        private function handleEditEnd(event:DataGridEvent):void
        {

            var edit:TextInput =
TextInput(event.currentTarget.itemEditorInstance);
            var name:String = edit.text;

            var uid:String = myObjects.getItemAt(event.rowIndex).uid;

            var dob:DiagramObject =
DiagramObject(diagram.getElementByDataObjectId(uid));
            if (dob.annotation)
                dob.annotation.text = name;
            else
                diagram.createAnnotation(dob, name);
        }
```

In the handleEditEnd function we retrieve the new value and set it as an annotation for the corresponding sprite in the diagrammer. If the sprite doesn't already has an annotation we create it.

You now have a bi-directionnal data bindings between the diagram and application data.

Sprites composition in a diagram

In this part we will add sprite composition capability to our application.
Sprite composition is a mecanism of visually combining several sprites into a single one, just like in a group. But there are differences between grouping and composition :

one of the sprites (the master or composite) contains all the composition elements. He can be move and resize, and the composition elements are moved and resized accordingly.
Not all sprites can become composite or composition elements. This is defined by the SVG style "action-accept:composition" for a composition element and "action-accept:composite" for a composite (or master element).
A composite has reserved areas (svg elements) where composition elements can be put. There is only one composition element per area. So there is a maximum number of elements in a composition, and the maximum depends on the composite sprite definition.
Composition is triggered when the user drags a sprite (the composition element) into another sprite (the composite).
Composition is considered as a change in data model, so acceptance of composition is enforced through methods of the sprite proxies, and occurrence of composition lead to sprite proxies methods calls.

Creating a SVG shape library

Starting for the result of previous part, we change the SVG library to add composition capabilities.

Modify the XML svglibrary tag content accordingly:

Composition Capabilities Add On The XML SVG Library Tag

```
<svg xmlns:svg="http://www.w3.org/2000/svg" xmlns:k="http://schemas.kapit.fr/svg/2007/"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg" version="1.0" width="1024"
height="800" id="sgvlib">
  <defs id="defs1">
    <style type="text/css">
      <![CDATA[
.basic {
action-accept:annotation;
}
.bg {
action-click:link;
action-accept:link;
}
.composition {
action-accept:composition;
action-accept:annotation;
}
.composite {
action-accept:composite;
}
.annotation {
font-style:italic;
}
.link-annotation
{
        font-size:12px;
        fill:#666;
}
]]>
    </style>
    <linearGradient id="linearGradient3157">
      <stop id="stop3159" style="stop-color:#e1eaf5;stop-opacity:1" offset="0" />
      <stop id="stop3161" style="stop-color:#c3daea;stop-opacity:1" offset="1" />
    </linearGradient>
    <linearGradient xlink:href="#linearGradient3157" id="linearGradient6627"
gradientUnits="userSpaceOnUse" gradientTransform="translate(98.840748,-67.175132)"
x1="94.95433" y1="85.235107" x2="94.95433" y2="146.25534" />
  </defs>
  <g
    k:spriteid="rectangle"
    k:groupid="Basic"
    class="composition"
    id="grectangle">
    <rect
       id="rect3155"
       style="fill:url(#linearGradient6627);fill-opacity:1;stroke:#6996cf;stroke-width:1.5;stroke-
linecap:round;stroke-linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       x="137.22656"
       y="18.47105"
       rx="0"
       ry="0"
       width="98.994949"
```

```
        height="60.609154" />
     <path
        class="bg"
        id="rect31551"
        style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
        d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
  <g
     k:spriteid="rectangle-composed"
     k:groupid="Composed"
     class="composition"
     id="grectangle">
     <rect
        id="rect3155"
        style="fill:url(#linearGradient6627);fill-opacity:1;stroke:#3060DE;stroke-width:5;stroke-
linecap:round;stroke-linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
        x="137.22656"
        y="18.47105"
        rx="0"
        ry="0"
        width="98.994949"
        height="60.609154" />
     <rect
        id="rect3156"
        style="fill:none;stroke:#3060DE;stroke-width:5;stroke-linecap:round;stroke-linejoin:miter;stroke-
miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
        x="147.22656"
        y="28.47105"
        rx="0"
        ry="0"
        width="78.994949"
        height="40.609154" />
     <path
        class="bg"
        id="rect31551"
        style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
        d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
  </g>
  <g
     k:spriteid="rounded-rectangle"
     k:groupid="Basic"
     id="grrectangle"
     class="basic" transform="translate(116.20044,0)">
     <rect
       style="fill:#e1eaf5;fill-opacity:0.8;stroke:#6996cf;stroke-width:1.5;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:1"
       id="rect3157"
       y="18.47105" x="137.22656"
       rx="10" ry="10"
       height="60.609154" width="98.994949" />
     <path
        class="bg"
        id="rect31552"
        style="fill:none;fill-opacity:0;stroke:#6996cf;stroke-width:3;stroke-linecap:round;stroke-
linejoin:miter;stroke-miterlimit:4;stroke-dasharray:none;stroke-opacity:0"
        d="M 137.22656,18.47105 L 236.22151,18.47105 L 236.22151,79.080204 L 137.22656,79.080204 L
137.22656,18.47105 z" />
```

```
    <path
      class="composite"
      style="fill:#6996cf;fill-opacity:0.1;stroke:#FFFFFF;stroke-width:1;stroke-opacity:0.3"
      id="compo1"
           d="M 180,18 L 210,18 L 210,38 L 180,38 L 180,18 z" />
    <path
      class="composite"
      style="fill:#6996cf;fill-opacity:0.1;stroke:#FFFFFF;stroke-width:1;stroke-opacity:0.3"
      id="compo2"
           d="M 180,59 L 210,59 L 210,79 L 180,79 L 180,59 z" />
  </g>
</svg>
```

Changing the mappings

Replace the diagramMappings tag with this:

Diagram Mapping Modification Code

```
<mx:XML xmlns="" id="diagramMappings">
        <mappings>
                <sprite id="rectangle" width="40" height="40"
proxyclass="MyObjectProxy" />
                <sprite id="rectangle-composed" width="40" height="40"
proxyclass="MyObjectProxy" />
                <sprite id="rounded-rectangle"  width="40" height="40"
proxyclass="MyObjectProxy" />
                <sprite id="comment" proxyclass="MyObjectProxy" />
                <lane proxyclass="" />
                <panel proxyclass="" />
                <lanelink proxyclass="MyLinkProxy" />
                <panellink proxyclass="MyLinkProxy" />
                <viewlink proxyclass="MyLinkProxy" />
        </mappings>
</mx:XML>
```

We only changed the sprite proxy definition to match our new library.

Creating objects programmaticaly

We will create a composite at the application start.

First add some imports

Basic Import Code

```
import com.kapit.diagram.view.DiagramSprite
```

Change the initDiagram function:

Composite Creation Code

```
                public function initDiagram():void
                {
...

                                proxy = new KDLProxy(diagram);
                                proxy.importGraph();

                                var  s2:DiagramSprite =
diagram.createSprite("rounded-rectangle", null, true, false);
                                var  s1:DiagramSprite =
diagram.createSprite("rectangle", null, true, false);
```

```
                    diagram.createComposition(s2, s1, false);
                }
```

We first create a rounded-rectangle sprite, which can become a composite, then a rectangle, which can become a composition element.

Then we create a composition with the rounded-rectangle as a master, and the rectangle as an element.

At this point if you run your application you will see the composition sprite. You can move it, resize it and you will see how the composition element are moved and resized.

Composition and data object proxies

We will create a composite at the application start.

Adding properties to data object

Edit the MyObject.as file to add properties :

Data Object Properties Add

```
                [Bindable]
                public var uid:String;

                [Bindable]
                public var compositions:Number;

                [Bindable]
                public var master:String;

                public function MyObject()
```

We will count the number of elements in a composition (the compositions properties), and store the master (composite) of a composition element.

Handling composition in object proxy

Modify MyObjectProxy.as file to handle composition messages.

Composition Handler on The Object Proxy Code

```
protected function getElementSpriteId(el:IDiagramElement):String
{
        if (el is DiagramSprite)
        {
                var s:DiagramSprite = el as DiagramSprite;
                return s.spriteid;
        }
        return null;
}
public function createDataObject(el:IDiagramElement):String
{
        var type:String = el.getTagName();
        var spriteid:String = DiagramSprite(el).spriteid;
        var name:String = el.did;
        var uid:String = UIDUtil.createUID();
        _objects.length
        var obj:MyObject = new MyObject();
        obj.type = type;
        obj.spriteid = spriteid;
        obj.did = el.did;
        obj.uid = uid;
```

```
        obj.name = "";
        obj.compositions = 0;
        obj.master = "";
        if (_objects)
                _objects.addItem(obj);
        return uid;
}

public function removeDataObject(el:IDiagramElement):void
{
        var index:int = getElementIndex(el);
        if (index != -1)
                _objects.removeItemAt(index);
}

public function allowLinkAction(el:IDiagramElement):Boolean
{
        return true;
}

public function propertyModified(el:IDiagramElement, propname:String,
propvalue:Object, shapeid:String):void
{
        var index:int = getElementIndex(el);
        var obj:MyObject = null;
        if (index != -1)
        {
                obj = _objects.getItemAt(index) as MyObject;
        }
        var s:DiagramSprite = null;
        if (el is DiagramSprite)
                s = el as DiagramSprite;

        if ("text" == propname)
        {
                if (obj)
                {
                        obj.name=String(propvalue);
                }
        }
        if ("compositionmasterleave" == propname)
        {
                if (s && "rectangle-composed" == shapeid)
                {
                        s.spriteid = "rectangle";
                        s.dragenabled = true;
                        s.selectable = true;
                        if (obj)
                        {
                                obj.spriteid = s.spriteid;
                                obj.master="";
                        }
                }
        }
        if ("compositionmaster" == propname)
        {
                if (s && "rectangle" == shapeid)
                {
                        s.spriteid = "rectangle-composed";
                        s.dragenabled = true;
                        s.selectable = true;
```

```
                        if (obj)
                        {
                                obj.master=DiagramSprite(propvalue).did;
                                obj.spriteid = s.spriteid;
                        }
                }
        }
        if ("compositionelement" == propname)
        {
                if (obj)
                {
                        obj.compositions=obj.compositions + 1;
                }
        }
        if ("compositionelementremove" == propname)
        {
                if (obj)
                {
                        obj.compositions=obj.compositions - 1;
                }
        }

}
```

The property **compositionmaster** is changed when a sprite enter a composition, and **compositionmasterleave** when he leaves a composition.

In our application when a sprite enter a compositon we :

Make it selectable and movable.
Change its sprite.
We he leaves a composition, an element sprite is set back to original sprite.
The property **compositionelement** is changed when a composite receive a new composition element, and**compositionelementremove** when the element leaves (or is deleted).
In our application we increase and decrease the number of element of the composite when an element is added or removed.

Displaying properties in a DataGrid

Change the objects datagrid to display compositions and master properties

Composition & Master Properties Display Integration Code

```
                <mx:DataGrid id="objectsGrid" dataProvider="{myObjects}"
itemEditEnd="handleEditEnd(event);" itemClick="handleObjectSelected(event);" width="100%"
height="100%" editable="true" >
                        <mx:columns>
                                <mx:DataGridColumn headerText="Type"
dataField="spriteid" editable="false"/>
                                <mx:DataGridColumn headerText="Sprite"
dataField="did" editable="false"/>
                                <mx:DataGridColumn headerText="ID"
dataField="uid" editable="false"/>
                                <mx:DataGridColumn headerText="Name"
dataField="name" editable="true"/>
                                <mx:DataGridColumn headerText="Elements"
dataField="compositions" editable="false"/>
                                <mx:DataGridColumn headerText="Master"
dataField="master" editable="false"/>
                        </mx:columns>
```

```
                </mx:DataGrid>
```

Run the application and see as the elements and master fields gets updated when you move elements from composite to another, delete them, add them...

Alter composition acceptance through proxies

Change the MyObjectProxy class to handle accept message:

Message Acceptance Handler Through Proxies Integration Code

```
public function acceptPropertyModification(el:IDiagramElement,propname:String,propvalue:Object,shapeid:String):Boolean
{
        var index:int = getElementIndex(el);
        var obj:MyObject = null;
        if (index != -1)
        {
                obj = _objects.getItemAt(index) as MyObject;
        }

        if ("text" == propname)
        {
                return true;
        }
        if ("compositionmasterleave" == propname)
        {
                return true;
        }
        if ("compositionmaster" == propname)
        {
                return ((el as DiagramSprite).masterobject == null || (el as DiagramSprite).masterobject == propvalue);
        }
        if ("compositionelement" == propname)
        {
                // No more than one element in a composition
                if (obj)
                {
                        return ((obj.compositions<1) || (obj.compositions == 1 && (propvalue as DiagramSprite).masterobject == el));
                }
        }
        if ("compositionelementremove" == propname)
        {
                return true;
        }
        return true;
}

public function acceptRemoveObject(el:IDiagramElement):Boolean
{
        var s:DiagramSprite = el as DiagramSprite;
        if (s && s.masterobject)
                return false;
        return true;
}
```

Returning false in these function forbid the user to do the action he attempted.

In the compositionmaster case, we accept that a sprite enter a composition only if it wasn't in a composition before or with the same master (in the case the element is moved in another area inside the composite).

In the **compositionelement** case, we accept that a sprite enter a composition only if it the composition is empty or with the same master (in the case the element is moved in another area inside the composite). This way we limit the number of element of a composition to 1, although we can have several area in a composite.

In the acceptRemoveObject, we don't accept to remve a sprite inside a composition (with a non null masterobject).

As you can see, with acceptance functions you have great flexibility to enforce your business rules.