# XML jGraph (Version: 1.2) - Table of Content

# Preface

XML jGraph is a web application graph shown in a browser page. Works like an HTML page content rendered by the browser.

A graph application for displaying XML metadata based graph for web applications. You no longer have to embed static images in your web page, instead generate XML file and display this graphic view for better user interaction, user input and integration of user events with back end.

# How to install and view graph from browser page

Deploying and running the sample XML metadata files to show some graph is very simple. There is a graph viewer comes in this web archive. All you need is to have a web deployment container.

**Dependencies**
If you already do not have a web container for deploying a WAR application you may download one. The simplest one to download is a Tomcat web container which is J2EE enabled and very easy to deploy and test these samples.

**Steps: in installing and running**

**1. Download the XML jGraph web application archive**

Download the WAR (web application) archive and save it as (or rename it) "xmljgraph_browser.war" file. This is a self sufficient web application archive and you need no other software.

**2. Install the WEB (WAR) application**
Copy the WAR file or open it as instructed next. When deployed the context path of this application will be the name of WAR file; e.g. "xmljgraph_browser".

**Tomcat**:
Copy it to the installation/webapps/ directory.

**JBoss**:
Copy to installation/server/default/deploy directory.

**BEA WebLogic** or **IBM WebSphere**:
Logon to admin console and choose to deploy an existing WAR file which will let you select this WAR file located in any folder.

[NOTE: These are some of the well known J2EE enabled web servers and not the only web servers to deploy. This application may contain other scripting language pages such as ASP, PHP etc. and those may be run for non-J2EE enabled web servers. Following are examples of deploying in some of the J2EE enabled web servers.]

**3. TEST: if your installation is OK.**
To test if your installation is ok try typing the sample metadata file as arguments in the URL location filed on your browser and pressing ENTER key (NOTE: replace PROTOCOL, HOST, and PORT). Examples next to them might also help you.

**Viewer page**
PROTOCOL://HOST:PORT/xmljgraph_browser/**xml_jgraph_viewer.jsp**?
DOCUMENT=xml_meta_file

[
NOTE: If your server not JSP enabled may try ASP or PHP pages

PROTOCOL://HOST:PORT/xmljgraph_browser/xml_jgraph_viewer.asp?
DOCUMENT=xml_meta_file

Or PHP

PROTOCOL://HOST:PORT/xmljgraph_browser/xml_jgraph_viewer.php?
DOCUMENT=xml_meta_file
]

**Examples**
http://www.activetree.com/xmljgraph_browser/xml_jgraph_viewer.jsp?DOCUMENT=http://www.activetree.com/xmljgraph_browser/demo/data/topology1.xml

Or (ASP)

http://www.activetree.com/xmljgraph_browser/xml_jgraph_viewer.asp?DOCUMENT=http://www.activetree.com/xmljgraph_browser/demo/data/topology1.xml

Or (PHP)

http://www.activetree.com/xmljgraph_browser/xml_jgraph_viewer.php?DOCUMENT=http://www.activetree.com/xmljgraph_browser/demo/data/topology1.xml

**3. Try DEMO Samples**
Run demo samples and view the tutorial from your installation. This web application has few demo samples and the link to each of them is in the start page. Sample scripts (e.g. JSP files) are part of the WAR installation in the "demo" directory. Go to your installation and find the "demo" directory under which you will find JSP sample programs.

Demo start page (NOTE: replace PROTOCOL, HOST and PORT)

PROTOCOL://HOST:PORT/xmljgraph_browser/

**4. Tutorial**
Start page also has a link to the tutorial to learn more about the application.

**Arguments for the graph viewer page**

Typically these are the arguments used for viewing a metadata graph. The required argument is the “DOCUMENT” parameter and rest are optional and depending on whether or not the document requires an authentication for download.

Example of a complete viewer URL:
http://www.activetree.com/xmljgraph_browser/xml_jgraph_viewer.jsp?DOCUMENT=http://www.activetree.com/xmljgraph_browser/demo/data/event1.xml&URL_AUTH_ID=&URL_AUTH_PASSWORD=

Notice here is that the “URL_AUTH_ID” and “URL_AUTH_PASSWORD” parameters are left empty because this XML file is not a password protected.

| Parameter | Details |
|---|---|
| DOCUMENT | The URL of the metadata file; DOCUMENT=xml_file<br><br>Example:<br>http://www.activetree.com/xmljgraph_browser/demo/data/topology1.xml |
| URL_AUTH_ID | URL authentication user ID if applicable.<br>Example:<br>URL_AUTH_ID=Id |
| URL_AUTH_PASSWORD | Password for such URL_AUTH_ID if any<br>Example:<br>URL_AUTH_PASSWORD=Pswd |

# How to run metadata samples in standalone mode

Download the standalone ZIP archive and unzip in a local folder. Go to the "bin" directory and run the script files from a command window. There are some BAT files (for Window users) and SH files (for UNIX/LINUX users).

You may need the following arguments to pass to the script files:

Argument-0: XML metadata file [from which the view is constructed]
Argument-1: license key [if applicable]

# Features and Frameworks

This chapter is a brief overview of the graph design framework and features.

## *Graphical Features*

- Hierarchical – viewing of objects that are hierarchical (and non-hierarchical)
- Text – supports styled text
- Images and Icons – content includes images and icons downloaded from the URL location or jar files
- Charts – add charts as the object content (which are also XML content)
- Expand/Collapsible - objects can expand/collapse if it has children objects under it

- Efficient - contents are loaded only when user expands them (or programmatically expanded)
- Faster – Thin client and faster like native applications

## *Architectural Features*

- **XML metadata based**
  Developers can write simple XML content and let browser display the content as graph items. User can interact on your graphic items.

- **Faster development time**
  Simply type XML metadata and display the view in a browser page. Viewer for the graph is free and automatically downloaded for displaying the metadata content.

- **Platform independent**
  Client plug-in is a Java program and therefore you are free from platform dependency and always run from any platform consistently.

- **Download time**
  XML content when delivered to the client can be compressed to make it compact for reduced download time.

- **Security**
  Download content can be password protected, and authenticated using standard technology features.

- **Client plug-in**
  You write no client to view your graphs. Web browser will automatically download the client plug-in and your XML metafile content will be displayed in the browser window. You never have to worry about installing a client program for viewing your graphs.

- **Thin client plug-in**
  Browser when detects the appropriate tag it will download a client program which is (i) very thin and will be downloaded in seconds, (ii) downloaded only once for the first time such graph plug-in is detected. Any subsequent detection of such plug-in will not cause download of the client program again. Only metadata content such as metafile itself, icons, images, and other resources specified in the file will be downloaded for rendering the content.

- **Open standard metafile format**

XML is an industry standard open data file format and therefore you can always port the metafile content suitable for any other application.

## *Viewer framework*

Everything in the view is considered as an object of some kind. Typically, an XML graph object is a node object or an edge object. Shape of the node object may be of any kind represented by symbols, icons and styled text put together as one unit. Edge too may be of different shape, orientation and styles. Behavior of an object is described using the attributes and attribute value(s) in the XML metadata file.

Rendering information for each object in the metadata can be made part of the object itself or as global properties application for all of objects. View has its own set of default rendering properties. View when render an object a particular graphical property is searched in the following order: (i) in the object itself, (ii) in the global section, and (iii) default properties of the view. It uses the very first one it finds. Therefore, all common properties may be included in the metadata file global section and customization of a property may be overridden in the object definition.

## *Metadata framework*

### Layout

Layout of the view is defined as grids with rows and columns. Each cell location therefore is identified by a (row, column) dimension called grid index. Node objects may be placed in such grid index locations. All cell locations may not be filled up with an object. Moreover, where in the grid location an object be places will entirely depend on how the graph should look visually.

Example: If there are 100 objects in a single hierarchy a grid size such as (rows=10, cols=10), (rows=100, cols=1), (rows=1, cols=100), (rows=2, cols=50), (rows=20, cols=5) may be declared. Then objects may be put in the meaningful grid index (row, column) location.

### Layout objects

A node object can contain child objects and can be expanded or collapsed (programmatically and by user). Layout provides cell locations where graph node objects can be added. Remember that edges are not added as cell objects they are added outside of a node. Once a layout or certain rows and columns are declared one or more cells may be filled leaving rest cells empty. It is not a requirement to fill at lease one cell and all cells may be left empty.

### Nodes

Cell locations may contain node objects. A node object can take many kinds of shape using styled text, icons with its own internal presentation style.

### Edges

An edge object can have some variations by way of specifying how the edge be drawn.

- **Auto edge**
  Auto edges are those automatically gets created and connects between the source and destination node touching the node boundaries. Edges are defined outside of a node definition identified by a unique ID for self, the ID of the parent node, and IDs of the source and destination object.

- **Poly edge**
  These edges are those that pass through center points of bunch of cell locations specified as path items in the metadata. This kind of edge therefore does not need a source object as well as a destination node. It passes through the cell locations specified by path items. A path item is one of the line segment identified by the source and destination cell location.

## *Writing XML Metadata*

Most developers are familiar with writing some XML files. Even if it sounds not so familiar, following the simple to write XML metadata examples next, it certainly will sound easy.

### Metadata Elements

Metadata file consists of the following:

- Graph view attributes identified by “xmljgraph” tag
- Global section defines values of attributes that will be common to all objects in the graph identified by “global” tag.
- Nodes section defines all node objects used in cell locations identified by “nodes” tag
- Edges section defined all edges (in any hierarchy) using “edges” tag
- There can be only one “nodes” and “edges” tag meaning only one graph in a single view with nodes defined under “nodes” and edges defined under “edges” section.

## Defining the graph

All graph starts with "<xmjgraph>" tag and ends with "</xmljgraph>". Graph typically may have a title shown just above the graph and a name typically may be used for searching and other purpose. These attributes are optional.

Example:

```
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">
…
</xmljgraph>
```

In this example graph metadata graph name is "graph1" and titled "XML jGraph (Topology Graph Sample)".

## Global section

This section comes as the very first section under the "xmljgraph" declaration. Global section starts with "global" tag where all common and overridden attribute values such as color and font for labels, icon download URL base location, and any other attribute applicable to "nodes" and "edges". This helps in avoiding redundancy in declaration the same attribute and value in each object.

Example:

```
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">
 <global
  viewInsets="40, 10, 40, 40"
  baseUrl="http://www.activetree.com/images/xmljgraph/"
  background="50,200,50,255"
  foreground="0, 0, 0, 255"
 />
 …
</xmljgraph>
```

## About "baseUrl" and "archive" tags

One or combination of both can be used in the global section to indicate the location of the icons or resources used in the metadata XML file. For example, the icons used may be available in the classpath of an application, archive tag of an applet or they can be located in a remote web site or a local file identified by these two tags.

Here are some examples of usage:

```
Examples of loading icons and resources:
//from jars in classpath
baseUrl="/"
baseUrl="/com/mycompay/icons/"

//from remote website
baseUrl=http://www.activetree.com/images/xmljgraph/

//Using archive that can be across multiple jars
archive=http://www.activetree.com/xmljgraph/demo/icons.jar,
http://www.activetree.com/xmljgraph/demo/icons_others.jar
```

## *Overriding a global attribute*

If an attribute should be different for a particular object it can be added in the object metadata definition and that will override the same attribute in global section and also in view default.

Example:

```
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">
 <global
  ….
  background="50,200,50,255"
  foreground="0, 0, 0, 255"
  ….
 />

<!-- begin nodes -->
….
      <expNode>
        <id>1</id>
        <gridIndex>0,0</gridIndex>
        <headerLabel>J2EE.Component.EJB.Entity</headerLabel>
        <footerLabel>AddService</footerLabel>
        <background>255, 200, 0, 255</background>
        <tooltip>Add Service Entity Bean</tooltip>
        <icons>statef_ejb.gif</icons>
      </expNode>
```

```
….
<!-- end nodes -->
```

# Create a simple graph

Starting to write samples is perhaps the best way to produce some graphs quickly and build better understanding.

## *Write a simple metadata file*

Here are step by step guide in writing a simple metadata file “topology.xml”.

Following are some of the commonly used attributes applicable for view definition:

. xmljgraph – must start with this tag
. name – a name (optional)
. title – title of the view (optional)

Apply them in creating a the XML file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">
</xmljgraph>
```

## Run your metadata sample

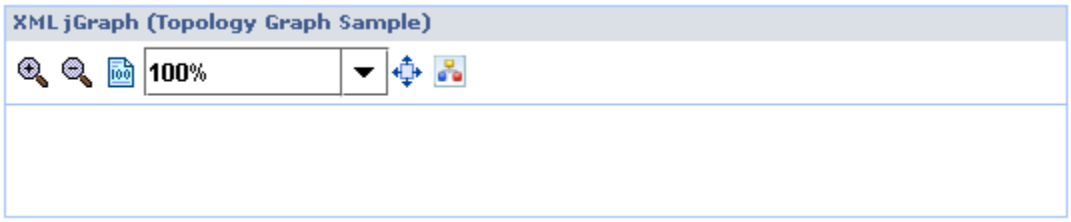
If the XML jGraph archive is not downloaded by you already, download it now. Unzip in a local folder. If the archive is unzipped under “C:/” in windows machine, it created a folder something like “XmlJGraph_x.y.z”. Go to the “bin” directory and the current directory looks something like “c:/XmlJGraph_x.y.z/bin”. There are some script files to run XML files for graph viewing, printing and converting the graph to images. There is this script file “XmlGraphViewerDemo.bat” for windows and “XmlGraphViewerDemo.sh” for Unix/Linux for displaying the view from the XML meta file.

From your current bin directory type the following command with your metadata XML file absolute path as argument to the script file.

```
C:/XmlJGraph_x.y.z/bin/XmlJGraphViewerDemo.bat  <xmlFile>
```

If you run this “topology.xml” metadata file it will produce a graph with empty content.

*XML jGraph with empty content*




## Add content into your graph

Objects such as “nodes” and “edges” can be added to the graph. Edges are typically the lines connecting a source object and destination object. For declaring an edge requires the ID of the source and the destination object. It is therefore essential to add the “id” attribute for each object you declare in the graph.

When adding node objects it might contain icons and images downloadable from a web site. It is worth adding a the “baseUrl” entry in the “global” section of the graph as the base URL and the icons if specified in any object will be relative to this base URL path.

Also, added another global entry “viewInsets” in order to provide spaces around the objects added to the view.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">

 <global
  viewInsets="10, 10, 10, 10"
  baseUrl="http://www.activetree.com/images/xmljgraph/"
 />

</xmljgraph>
```

## Add nodes

All nodes are declared using “nodes” tag. Node objects declared under “nodes” as “expNode”. Simple meaning of “expNode” is that a node can have children in it; i.e. node is expandable and therefore can be hierarchical.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

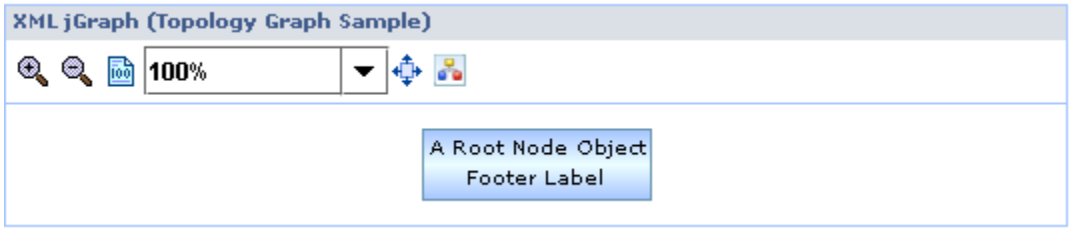
```
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">

 <global
  viewInsets="10, 10, 10, 10"
  baseUrl="http://www.activetree.com/images/xmljgraph/"
 />

 <nodes>
  ....
 </nodes>

</xmljgraph>
```

## Add a children node in your graph

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">

 <global
  viewInsets="10, 10, 10, 10"
  baseUrl="http://www.activetree.com/images/xmljgraph/"
 />

 <nodes>
  <expNode>
   <id>0</id>
   <gridIndex>0,0</gridIndex>
   <headerLabel>J2EE.Component.EJB.Entity</headerLabel>
   <footerLabel>AddService</footerLabel>
   <tooltip>Add Service Entity Bean</tooltip>
  </expNode>
 </nodes>

</xmljgraph>
```

Run it again and it will display a simple graph with one simple looking node object with two labels.

*Xml jGraph with a one node object*



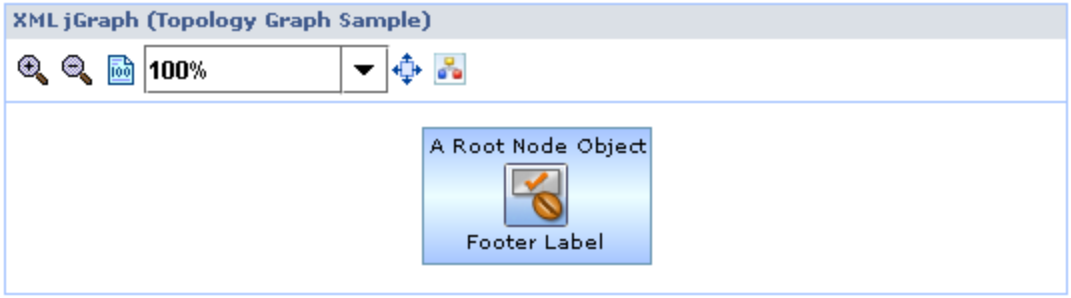

## Adding icons to a node

There are lots of things possible to do for a node object and one of them is to show some icons at the center of the node. There can be more than one icons and they will be added as comma separated list; e.g. "`icon1.gif, icon2.gif, …, iconN.gif`" in the "icons" tag.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
….
   <expNode>
     ….
     <icons>statef_ejb.gif</icons>
   </expNode>
….
</xmljgraph>
```

Run it will show a graph with icons and labels both in the object.

*Node object with icons*




# Creating Single Hierarchical Graph

This section shows how to add multiple nodes inside a parent node to make a two dimensional simple graph view.

It is mentioned before that the XML jGraph is hierarchical and a node is defined with "expNode" which is an expandable node object. Adding some children nodes in a parent node and making the parent node invisible will make a single hierarchical graph view.

## *Set rows and columns*

The root node object should have "rows" and "columns" attributes for defining two dimensional grid cells like a table with rows and columns. For simplicity this example is using 2 rows and 2 columns. A maximum of 4 graph node objects can be added in these cell. Remember edges are not cell object but objects added to the nodes if the nodes in the layout may be connected to show some relationship between them. Edges are optional and declared separately under "edges" tag.

```
….
 <nodes>
  <expNode rows="2" columns="2">
    ….
  </expNode>
 </nodes>
….
```

## *Set root attributes*

By default a node is made visible and rendered based on the attributes. You have option to let the root be visible or not visible. Making the root itself not displayed will make more like a one level graph than showing root with expand and collapse icons. In this demo root node is simple not displayed by adding "display" attribute to "false". Since the nodes in the root will be displayed when the graph is displayed for the first time an "expanded" attribute is added with a value to "true".

All node objects must have an "id", a "gridIndex" including the root object.

```
….
 <nodes>
  <expNode rows="4" columns="2">
    <id>0</id>
    <gridIndex>0,0</gridIndex>
    <expanded>true</expanded>
    <display>false</display>
    <childs>
        ….
    </childs>
  </expNode>
 </nodes>
```

```
....
```

## *Add childrens to the root*

Add the children nodes in the cell locations. They can be added randomly to any cell location. However it all depends on application and the relative relationship between the siblings. In this demo there are 3 childrens added. Two childrens added to the first column and the 3th child is added at cell (0, 1) location; i.e. row=0 and column=1. Child nodes is added to the root with "childs" attribute.

Here is the complete source metadata after adding three node objects to the graph.

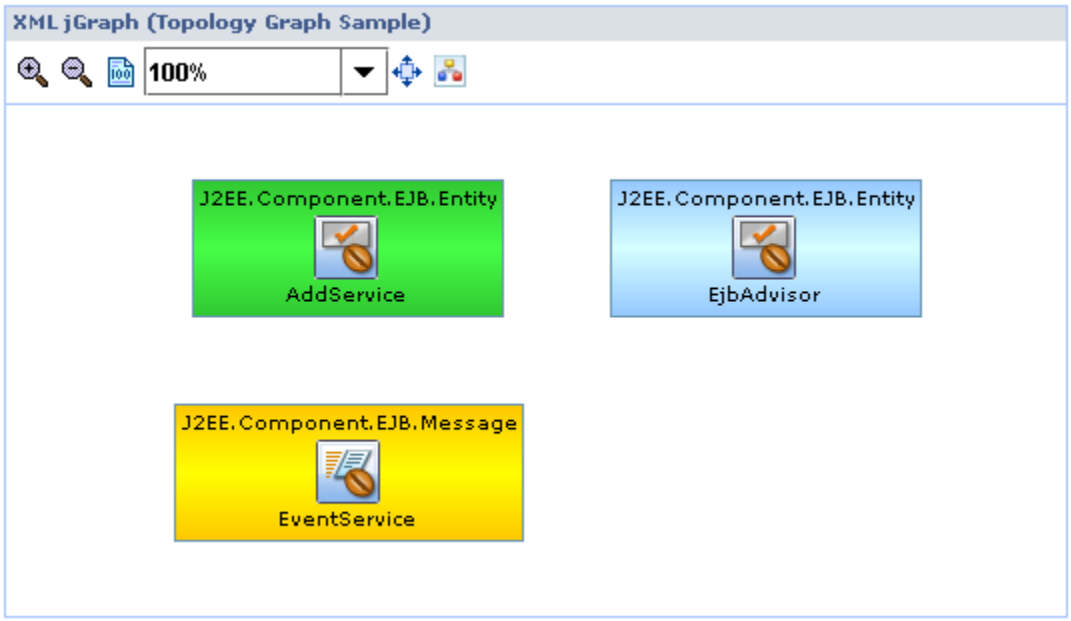
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">

 <global
  viewInsets="10, 10, 10, 10"
  baseUrl="http://www.activetree.com/images/xmljgraph/"
 />

 <nodes>
  <expNode rows="2" columns="2">
   <id>0</id>
   <gridIndex>0,0</gridIndex>
   <expanded>true</expanded>
   <display>false</display>
   <childs>
    <expNode>
     <id>1</id>
     <gridIndex>0,0</gridIndex>
     <headerLabel>J2EE.Component.EJB.Entity</headerLabel>
     <footerLabel>AddService</footerLabel>
     <tooltip>Add Service Entity Bean</tooltip>
     <icons>statef_ejb.gif</icons>
     <background>50,200,50,255</background>
    </expNode>
    <expNode>
     <id>2</id>
     <gridIndex>0,1</gridIndex>
     <headerLabel>J2EE.Component.EJB.Entity</headerLabel>
     <footerLabel>EjbAdvisor</footerLabel>
     <tooltip>Advisor Entity Bean</tooltip>
     <icons>statef_ejb.gif</icons>
     <background>50,200,50,255</background>
```

```
      </expNode>
      <expNode>
        <id>3</id>
        <gridIndex>1,0</gridIndex>
        <headerLabel>J2EE.Component.EJB.Message</headerLabel>
        <footerLabel>EventService</footerLabel>
        <tooltip>EventService Message EJB</tooltip>
        <icons>message_ejb.gif</icons>
        <background>255, 200, 0, 255</background>
      </expNode>
    </childs>
  </expNode>
 </nodes>
</xmljgraph>
```

### *Run it*

Running this XML metadata sample displayed the following graph. No edges are added yet and no relationship among the siblings is shown.

*Xml jGraph with multiple nodes in a single hierarchy*




## Adding Edges

In lot of applications edges are used for showing some kind of relationship between nodes. Pretend this sample developed is a software application and some edges can show

the way the application modules related to each other; i.e. relationship between the siblings.

Another example is perhaps a flow diagram typically used for decision making where edges are used for making connection between nodes as indicators of flow of control from one location to another.

## *Edges tags*

Just like the "nodes" tag to start adding nodes "edges" tag is used for adding edges to a node object. In this sample, for example, cell(0,0) wants to make a connection to the cell(1,0) and cell(0,1) to cell(1,0). Therefore, it needs to add two edge objects for the root node declared under a separate section; e.g. "<edges> <edge>…</edge></edges>.

An edge can of different types; e.g. "edge", "polyEdge" etc. In this demo it is using "edge" type that automatically connects two nodes.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
….

<nodes>
  <!-- nodes are added here -->
</nodes>

<edges>
   <edge>
      <!-- edge attributes goes here -->
   </edge>
</edges>
</xmljgraph>
```

## *Edge tag*

An automatic type of edge is added using the "<edge>" and "</edge> tags shown next.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
….
<nodes>
  <!-- nodes are added here -->
</nodes>

<edges>
```

```
   <edge>
     <!-- details of an edge is here -->
  </edge>
</edges>
</xmljgraph>
```

## *Add edges*

This sample added couple of edges mentioned before under the “edges” tag after the “nodes” declaration section. Even though order does not matter, since edge exists only when node for which edges are declared exists. Therefore, good practice to add “nodes” first and then decide if there are “edges” to add for any of the nodes.

Here is the complete XML metadata file after adding the edges into it. Look at the later section describes more about the “edge” attributes. Typically an “edge” requires a “id” for self identification, parent ID “parentId” for graph to figure where this edge belong, a “fromNodeId” as id of the source child for the “parentId”, a “toNodeId” that is the destination child in the parent. Edge tag optionally can have a label and a tool tip as well. Edge can also have an arrow at the end with arrow control attributes mentioned with details in the “Edge attributes” section later in this document.

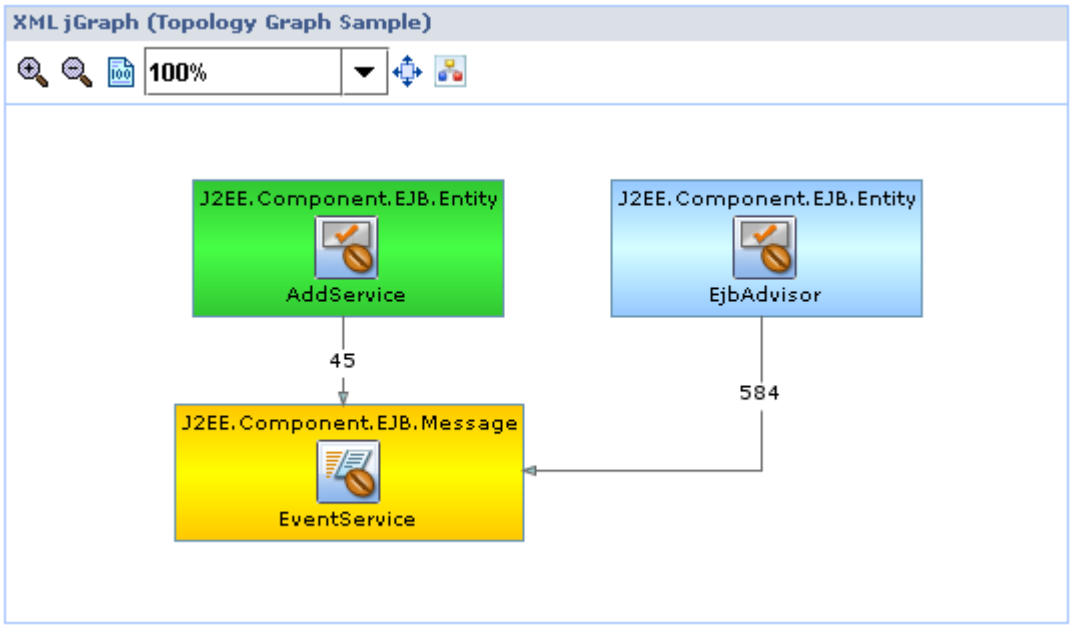
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmljgraph name="graph1" title="XML jGraph (Topology Graph Sample)">

 <global
  viewInsets="10, 10, 10, 10"
  baseUrl="http://www.activetree.com/images/xmljgraph/"
 />

 <nodes>
  <expNode rows="2" columns="2">
   <id>0</id>
   <gridIndex>0,0</gridIndex>
   <expanded>true</expanded>
   <display>false</display>
   <childs>
    <expNode>
     <id>1</id>
     <gridIndex>0,0</gridIndex>
     <headerLabel>J2EE.Component.EJB.Entity</headerLabel>
     <footerLabel>AddService</footerLabel>
     <tooltip>Add Service Entity Bean</tooltip>
     <icons>statef_ejb.gif</icons>
```

```
        <background>50,200,50,255</background>
      </expNode>
      <expNode>
        <id>2</id>
        <gridIndex>0,1</gridIndex>
        <headerLabel>J2EE.Component.EJB.Entity</headerLabel>
        <footerLabel>EjbAdvisor</footerLabel>
        <tooltip>Advisor Entity Bean</tooltip>
        <icons>statef_ejb.gif</icons>
        <background>50,200,50,255</background>
      </expNode>
      <expNode>
        <id>3</id>
        <gridIndex>1,0</gridIndex>
        <headerLabel>J2EE.Component.EJB.Message</headerLabel>
        <footerLabel>EventService</footerLabel>
        <tooltip>EventService Message EJB</tooltip>
        <icons>message_ejb.gif</icons>
        <background>255, 200, 0, 255</background>
      </expNode>
    </childs>
  </expNode>
</nodes>

<edges>
  <edge>
    <parentId>0</parentId>
    <id>100</id>
    <fromNodeId>1</fromNodeId>
    <toNodeId>3</toNodeId>
    <tooltip>Called 45 times</tooltip>
    <edgeLabel>45</edgeLabel>
    <drawEdgeLabel>true</drawEdgeLabel>
  </edge>
  <edge>
    <parentId>0</parentId>
    <id>101</id>
    <fromNodeId>2</fromNodeId>
    <toNodeId>3</toNodeId>
    <tooltip>Called 584 times</tooltip>
    <edgeLabel>584</edgeLabel>
    <drawEdgeLabel>true</drawEdgeLabel>
  </edge>
</edges>
</xmljgraph>
```

### *Run it*

Running this further edited XML metadata displayed the following graph with both nodes and edges.

*Xml jGraph with objects and edges*




## Node attributes

Following are some of the basic essential node level attributes

### *Commonly used attributes*

Rows and columns are most commonly used attributes for creating a hierarchical graph with expandable childrens.

### Rows and Columns

A node object may have “rows” and “columns” attributes when declared in the metadata. This is essential for creating a hierarchical view of objects. A node may have its own layout with number of rows and columns but may not be filled with any objects at all. However, it is suggested not to add a layout to a node if there are no children nodes going to be added to it.

| expNode attributes | Value type | Details |
|---|---|---|
| rows | Integer | Indicates number of rows for placing the children |

| | | |
|---|---|---|
| | | objects. |
| columns | Integer | Number of column |
| | | Example:<br><expNode rows=”2” columns=”2”><br>…<br></expNode><br><br>This declaration of a node can have maximum of 4 children objects at its 4 cell locations. You may not actually add any children and in that case this parent node will be treated as a child node and not expandable. |

### Header label

A node object can have a header label displayed at the top using the <headerLabel> tag. Header label is optional just like any other elements in a node are optional.

### Footer label

An optional footer label can be displayed using the <footerLabel> tag.

### Fonts for labels

Both header and footer labels can optionally have fonts in order to change the default font used. <headerLabelFont> and <footerLabelFont> are the font control tags for header and footer of a node respectively.

Example (header label font):
<headerLabel>Header label</header>
<headerLabelFont>Times New Roman, Bold, 12</headerLabelFont>

Example (footer label and font):
<footerLabel>Footer label</footerLabel>
<footerLabelFont>Helvetica, BoldItalic, 10</footerLabelFont>

## *Other Attributes*

| Attribute tag | Optional | Details |
|---|---|---|
| id | No | The unique ID of an object.<br><br><id>0</id> |
| gridIndex | No | The row, column location of an object.<br><br><gridIndex>0, 1</gridIndex> |
| ipadx | Yes | Internal X padding of an object with respect to its boundary.<br><br><ipadx>2</ipadx> |
| ipady | Yes | Internal padding Y padding of an object with respect to the object boundary.<br><br><ipady>2</ipady> |
| insets | Yes | Insets applied to the contents of an object all around it.<br><br><insets>5, 5, 5, 5</insets> |
| imageableInsets | Yes | Imageable insets is the space around the children objects when placed inside of a current object.<br><br><imageableInsets>10, 10, 10, 10</imageableInsets> |
| headerLabel | Yes | The upper label of an object.<br><br><headerLabel>Header label</headerLabel> |
| headerLabelFont | Yes | Font definition of the header label.<br><br><headerLabelFont>Courier, Bold, 12</headerLabelFont> |
| headerLabelColor | Yes | Foreground color for the leader label.<br><br><headerLabelColor>0, 0, 0, 255</headerLabelColor> |
| headerBackground | Yes | Header label background may be filled with a different background color than the object background color.<br><br><headerBackground>150, 0, 0, |

| | | |
|---|---|---|
| | | 255</headerBackground> |
| fillHeaderBackground | Yes | Whether or not to fill the header label background.<br><br><fillHeaderBackground>true</fillHeaderBackground> |
| | | |
| foregound | Yes | Object foreground applicable for all labels drawn inside an object. This foreground will be overridden if the individual label is specified another color.<br><br><foreground>0, 0, 0, 255</foreground> |
| background | Yes | Background color of an object for filling the entire object background.<br><br><background>255, 255, 255, 255</background> |
| footerLabel | Yes | Footer label displayed at the lower side of an object.<br><br><footerLabel>Footer label</footerLabel> |
| footerLabelFont | Yes | Font for the footer label.<br><br><footerLabelFont>Helvetica, Plain, 12</footerLabelFont> |
| footerLabelColor | Yes | Foreground color for the footer label.<br><br><footerLabelColor>0, 0, 0, 255</footerLabelColor> |
| footerBackground | Yes | Footer bay be painted with a different color than object background.<br><br><footerBackground>255, 255, 100, 255</footerBackground> |
| fillFooterBackground | Yes | “true” to fill the footer background, “false” to leave it with the object background.<br><br><fillFooterBackground>true</fillFooterBackground> |
| | | |
| focusColor | Yes | Color to draw the focus when an object is focused and the focus for the object is enabled. |

| | | |
|---|---|---|
| | | <focusColor>120, 240, 200, 255</focusColor> |
| selectionColor | Yes | Color to draw the selection when an object is selected and if the object is selectable.<br><br><selectionColor>100, 200, 150, 255</selectionColor> |
| highlightColor | Yes | Color to use if an object to be highlighted.<br><br><highlightColor>200, 150, 240, 255</highlightColor> |
| borderColor | Yes | Color used for drawing the border of an object. |
| expansionBorderColor | Yes | Object with childrens in it when expanded another border may be drawn around it may have a different color. |
| focusBorderThinkness | Yes | Object when focused a border may be drawn and how think is that border is specified using this tag. |
| expBorderThinkness | Yes | Thickness of the border shown when an object is expanded and drawBorderOnExpansion is enabled.<br><br><expBorderThickness>2</expBorderThickness> |
| icons | Yes | A comma separated list of image URL strings relative to the baseUrl.<br><br><icons>icon1.gif, icon2,gif</icons> |
| tooltip | Yes | Tooltip string for an object displayed when mouse is moved on it.<br><br><tooltip>My tooltip</tooltip> |
| elementsHGap | Yes | Horizontal gap between two items within an object. For example, if there are more than one icons, a gap between two icons may be used from the value of this element.<br><br><elementsHGap>2</elementsHGap> |
| elementsVGap | Yes | Vertical gap between any two elements in an object. For example, the gap between the header and icons or the gap between icons |

| | | |
|---|---|---|
| | | and footer.<br><br><elementsVGap>2</elementsVGap> |
| treeIconToElementGap | Yes | If a node has childrens a plus or a minus sign is shown for navigating to and from children objects. This element is used to specify a gap between the icon and other elements in the object.<br><br><treeIconToElementGap>3</treeIconToElementGap> |
| expansionIconUrl | Yes | An absolute URL for loading and displaying the tree will expand sign replacing the default icon.<br><br><expansionIconUrl>http://www.activetree.com/images/xmljgraph/plus.gif</expansionIconUrl> |
| collapseIconUrl | Yes | An absolute URL for loading and displaying the tree will collapse sign replacing the default icon.<br><br><expansionIconUrl>http://www.activetree.com/images/xmljgraph/minus.gif</expansionIconUrl> |
| drawHeaderFooterOnExpansion | Yes | Whether or not to draw the header and footer labels when this object will be shown expanded.<br><br>< drawHeaderFooterOnExpansion>true</ drawHeaderFooterOnExpansion> |
| expansionStyle | Yes | Values are “card”, and “clear”. Default is “card”.<br><br><expansionStyle>plain</expansionStyle> |
| gradientBackground | Yes | Whether or not to paint the background as gradient paint. Default is true.<br><br>< gradientBackground>false</ gradientBackground> |
| antiAliasShapes | Yes | Whether or not icons and shapes are painted anti-aliased. Default is true.<br><br>< antiAliasShapes>false</ antiAliasShapes> |

| | | |
|---|---|---|
| antiAliasLabels | Yes | Whether or not labels are painted as anti-aliased. Default is false.<br><br><antiAliasLabels>true</ antiAliasLabels> |
| selectionThickness | Yes | Thickness of the selection border.<br><br><selectionThickness>3</selectionThickness> |
| selected | Yes | When or not an object be displayed as selected without the user selection on it. Default is false when an object is displayed.<br><br><selected>true</selected> |
| drawBorderOnExpansion | Yes | "true" to draw a border around the object when expanded, "false" for not to draw ant border.<br><br>< drawBorderOnExpansion>true</ drawBorderOnExpansion> |
| drawExpansionIcon | Yes | Whether or not to draw the "expand/collapse" icons for showing and navigating to and from parent and its children objects. Default is true.<br><br>< drawExpansionIcon>false</ drawExpansionIcon> |
| drawBorder | Yes | An object may be shown with a border around it. Default is true.<br><br></drawBorder>false</drawBorder> |
| expanded | Yes | If an object has children objects in it may be shown as expanded during the initial display of the graph or at any time later programmatically.<br><br><expanded>true</expanded> |
| is3D | Yes | A 3D like effect may be drawn if this flag is set to "true".<br><br><is3D>true</is3D> |
| fillNode | Yes | Whether or not to fill the background of the node.<br><br><fillNode>false</fillNode> |
| focusable | Yes | Whether or not to show a focus on an object |

| | | |
|---|---|---|
| | | when user move mouse in and out of it.<br><br><focusable>false</focusable> |
| selectable | Yes | User when interacts with the displayed object in the graph may select an object is this flag is set to “true”. If set to “false” even user click on it will not show it as selected.<br><br><selectable>false</selectable> |
| visible | Yes | An object and its entire hierarchy of children objects may be made invisible by simply setting this flag to “false”. All object starting from this current object will not be used for rendering.<br><br><visible>false</visible> |
| display | Yes | This is different from <visible” flag and is used for specifying whether or not an object be shown or not shown in the graph. Setting this flag to “false” on an object will not effect its children objects.<br><br><display>false</display> |
| rowAlignment | Yes | Alignment of an object with respect to the other objects in the same row of the parent.<br><br>Values are: “leading”, “left”, “center”, and “right”.<br><br><rowAlignment>left</rowAlignment> |
| columnAlignment | Yes | Alignment of an object with respect to the other objects in the same column of the parent.<br><br>Values are: “top”, “bottom”, and “center”.<br><br>< columnAlignment >left</ columnAlignment > |
| | | |

## Edge attributes

Edge supports following attributes

| Attribute tag | Value type | Meaning |
|---|---|---|
| parentId | Integer | This is ID identifies in which node hierarchy this edge should be displayed.<br><br>Example:<br>< parentId>0</ parentId> |
| fromNodeId | Integer | "parentId" identified the location of this edge and "fromNodeId" identified the source object of the edge.<br><br>Example:<br>< fromNodeId>1</fromNodeId> |
| toNodeId | Integer | "toNodeId" identifies the destination object for this edge.<br><br>Example:<br><toNodeId>3</toNodeId> |
| edgeLabel | String | Edge can show a label using this tag.<br><br>Example:<br><edgeLabel>My label</edgeLabel> |
| drawEdgeLabel | "true"/"false" | |
| edgeLabelFont | String of format: "fontName, fontStyle, fontSize" | Eaxmples:<br>"Courier New, Italic, 12"<br>"Helvetica, BoldItalic, 10"<br>"Time New Roman, Plain, 14" |
| edgeColor | String of format: "red, green, blue, alpha" | A color attribute is specified with the following order of int values: "red, green, blue, alpha".<br><br>Example:<br><edgeColor>150, 200, 255, 255</edgeColor> |
| edgeThickness | Integer | Example (for thickness of two pixels):<br><edgeThickness>2</edgeThickness> |
| drawArrow | "true"/"false" | "true" – for showing the arrow at the destination object<br>"false" – connection without the arrow<br><br>Example:<br><drawArrow>true</drawArrow> |

| | | |
|---|---|---|
| fillArrow | “true”/”false” | |
| arrowFillColor | String of format: “r, g, b, a” | Example: < arrowFillColor>150, 200, 255, 255</ arrowFillColor> |
| selectionMode | String | “multiple” or “single” |
| defaultSelectionEnabled | “true”/”false” | Boolean “true” or “false” ; default “true” |

# User event handling

A displayed graph will let user interact with it. Events generated during such interaction can be received by adding event listeners. So far there is no discussion about how a XML metadata graph can respond to a user and programmatic interaction events.

## *Even handling attributes*

User can interact with the graph elements using the following mouse events. These events may be added to an object in order to perform certain action on such events. These events are predefined events and actions are performed based on the action list provided as arguments to these events.

| Event name | Meaning |
|---|---|
| onMouseDown | Mouse pressed |
| onMouseUp | Mouse released |
| onMouseClick | Mouse clicked |
| onMouseDoubleClick | Mouse double clicked |
| onMouseEnter | Mouse entered |
| onMouseOver | Mouse moved on an object |
| onMouseOut | Mouse exited the object |
| onMouseDrag | Mouse drag |

## *Performing single action*

These examples show single action to perform against a mouse event.

Examples:

### Adding mouse double click handler to a node object

```
<expNode>
  <id>1</id>
  <gridIndex>0,2</gridIndex>
  <footerLabel>Home page</footerLabel>
  <drawBorder>false</drawBorder>
  <icons>page-group-32.gif</icons>
  <onMouseDoubleClick>
    <web>
      <url>http://www.activetree.com</url>
      <target>newwin</target>
    </web>
  </onMouseDoubleClick>
</expNode>
```

## Adding mouse click handler to a node

```
<expNode>
  <id>4</id>
  <gridIndex>2,2</gridIndex>
  <footerLabel>Smart jRTF</footerLabel>
  <icons>page-32.gif</icons>
  <onMouseClick>
    <web>
      <url>http://www.activetree.com/silent_print_rtf_from_browser/</url>
      <target>newwin</target>
    </web>
  </onMouseClick>
</expNode>
```

## Adding mouse click handler to an edge object

```
<polyEdge>
  <parentId>0</parentId>
  <id>1006</id>
  <pathType>cellIndexes</pathType>
  <path>
    <pathItem>2,3</pathItem>
    <pathItem>3,3</pathItem>
    <pathItem>4,3</pathItem>
  </path>
  <onMouseClick>
    <web>
```

```
      <url>http://xmljgraph.activetree.com</url>
      <target>newwin</target>
    </web>
  </onMouseClick>
</polyEdge>
```

## Performing multiple actions

One may perform a series of actions specified in the metadata file. One can perform "click" as well as "double click" actions on an object.

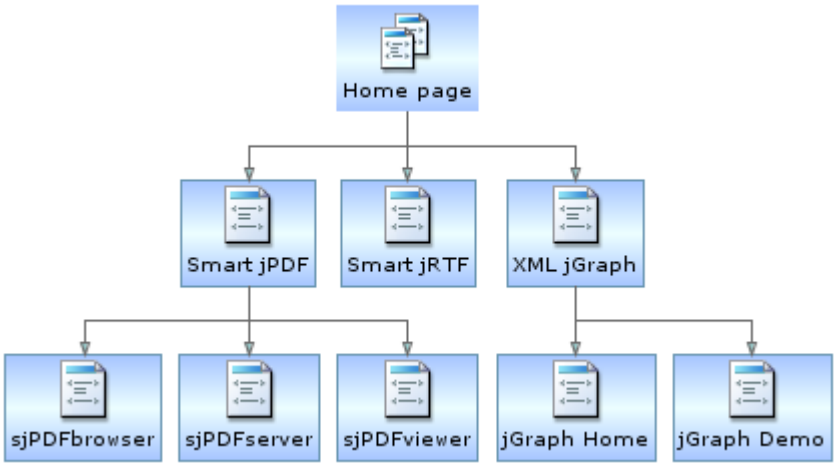
Example:
In this example a mouse click show a page while a mouse double click show a different page.

```
<expNode>
  <id>1</id>
  <gridIndex>0,2</gridIndex>
  <footerLabel>Home page</footerLabel>
  <drawBorder>false</drawBorder>
  <icons>page-group-32.gif</icons>
  <onMouseClick>
    <web>
      <url>http://xmljgraph.activetree.com</url>
      <target>newwin</target>
    </web>
  </onMouseClick>
  <onMouseDoubleClick>
    <web>
      <url>http://www.activetree.com</url>
      <target>newwin</target>
    </web>
  </onMouseDoubleClick>
</expNode>
```

## Sample event handling metadata file

In both the archives there is an event handling sample **`event1.xml`** made available as an example metadata file. In the browser graph viewer demo page there is a link to run and view this sample under "event handling" link.

Graph shown next is from “event1.xml” metadata sample shown by the graph viewer. Event sample “event1.xml” has added “onMouseClick” and “onMouseDoubleClick” on some of the nodes and edges. When running this sample try clicking (onMouseClick event) on edges which will open a new page in a browser window. Double clicking on nodes (onMouseDoubleClick) will also open new page in a browser window depending on which one is double clicked. Multiple events can handlers may be added to each graph object; e.g. “onMouseClick” and “onMouseDoubleClick” for performing two different actions on the same object.




## *Custom event handling*

Apart from predefined actions may be performed against an event one can also perform custom event handling through adding beans. Beans can be a Java object and a method to call, a java script method and more.

Customer handlers are called by the graph and the actual action may be performed by the implementation of the custom handler.

### Adding Java action handlers

Java action handler may be created by implementing `ContextEventListener` interface `processAction` method that takes a `ContextEvent` argument.

In the downloaded archive there is an example Java action handler implementation in `MyGraphEventListener` class.

All of the existing sample XML metafiles are using “contextEventListener” tag to listen to a `ContextEventListener` through the demo implementation in the

`MyGraphEventListener`. One might see a lot of console output messages that are originating from this listener class attached to the view through the XML metadata shown next.

Here is how to add a event handler to your metafile:

```
<global
  ….
  contextEventListener=”demo.activetree.graph.custom.listener.MyGraphEventListener”
  ….
/>
```

This example implementation `MyGraphEventListener` is available in the downloaded archives.