JGraph Updating Guide

Table of Contents

Introduction

This guide should help you to update existing code to the lastest JGraph versions. We try and keep the impact of new versions as minimal as possible, some bug fixes and extensions still require API changes. We will use deprecation where possible and explain the motivation of all changes and how to migrate your code in this guide.

Feel free to contact support@jgraph.com if you feel a change is missing or to request additional information for a change. (Please provide your order number when contacting support.)

JGraph 5.5.1

New Hooks in AbstractCellView

In order to avoid creation of unused attribute maps in the cell views, and to avoid or change the retrieval and cloning of the cell's attributes in the CellView.refresh, the AbstractCellView provides two hooks: createAttributeMap is called at construction time to create the allAttributes and attributes (same instance until the first refresh call) and getCellAttributes is called from refresh to retrieve and clone the cell attributes from the model.

These hooks may be used to reduce the memory footprint for large graphs. Please have a look at the FastGraph example.

Deprecated GraphCell.changeAttributes

The default values for points and the label position in edges is not required to be ensured in the DefaultEdge (ie on cell-level). Rather, the EdgeView's update method should check whether these values exist, and create them on the fly if they are missing. (In analogy to the bounds-check in VertexView.) Thanks to this, it is no longer required to change the attributes of a cell *via* the cell instance using changeAttributes, the attributes can be changed directly using the following code (where change is a Map and cell is an Object):

```
AttributeMap undo = graph.getModel().getAttributes(cell).applyMap(change);
```

Other Changes

- Various performance and memory improvements in edges and edge rendering and -hit detection
- Stores default values in allAttributes in VertexView.update
- Adds new examples to the commercial distribution
- Removes unnessesary calls to AttributeMap.createPoint

JGraph 5.5

GraphModel.valueForCellChanged

Sometimes it is required that a value for a cell be changed without knowing the actual cell- or graph model class, and without addint the change to the command history. Therefore, a new method has been added to the GraphModel interface which allows a client to change the value (aka. user object) of a cell without making any typecasts or adding the change to the command history.

Without this method a client that didn't want to affect the command history while changing the value of a cell was forced to either cast to the custom graph model or to the custom cell type in order to change the value.

The advantage of this change is that it is now possible to change the value of a cell (eg. prior to insertion) without knowing the custom model or custom types it contains. This is useful for functionalities that create new cells (such as group actions, mouse tools and import routines), because these are typically unaware of the concrete graph model or cell types, or there may be more than one model or cell type throughout the lifecycle of these objects.

In such a setup, the cell passed to the functionality is refered to as a "prototype" and is prepared (cloned and updated) for use in a new model with the following lines of code:

```
GraphModel model = graph.getModel();
Object newCell = DefaultGraphModel.cloneCell(model, prototype);
model.valueForCellChanged(cell, "Hello, world!");
```

Note that in this example, the prototype is an Object and no typecast is required to change the cell value. The value does not need to be a String, it can be any object.

How to migrate?

Developers of custom models and custom graph cells do only need to change the modifier of the existing valueForCellChanged method. This method has been there before, but was not part of the graph model interface. Instead it was used internally by the handleAttributes method to encapsulate the actual changing of the value, so that it could be overridden by subclassers to handle custom user objects. This is still the same, but with the new interface the method can be called by external parties as well, which means the visiblity must be changed from protected to public.

EdgeView.getEdgeRenderer

This method does a cast to EdgeRenderer on the value returned by getRenderer, and may therefore cause exceptions for subclassers that do not inherit from EdgeRenderer to implement their edge rendering. We have fixed this so that the invocations of this method are limited to the EdgeView class, so that a subclasser can safely assume that no other invocations to this method will be made. It is therefore required to override all methods which rely on this method within the EdgeView class, namely:

- getShape
- getLabelBounds
- getExtraLabelBounds
- intersects
- getBounds

The reason for closely coupling the EdgeView and EdgeRenderer is due to caching and separation of concerns between the EdgeRenderer and the EdgeView. For example: The EdgeRenderer is in charge of finding the actual bounds of the edge, whereas the EdgeView is in charge of caching these bounds until they need to be updated.

When implementing a custom edge renderer that does not inherit from EdgeRenderer then you need to make sure that these collaborations are clearly defined and then override the above methods with your custom code.

Note that there was one additional call to this method from within the BasicGraphUI (which was used to position the in-place editor). This call was fixed to use getRenderer and perform a typecast, using the top-left corner of the edge's bounding box as a default. You may want to override that getEditorLocation method to change this for a custom edge renderer.

Labels for Self-References (aka Loops)

The labels for self-references with no additional control points have been made moveable. The underlying functionality interprets the label position as an absolute vector in pixel coordinates in this special case, because one cannot use the normalized vector between to the end points of such loops (as they are at the same location and thus the resulting vector is 0).

The advantage of this change is that it is now possible to move labels on loops that have no control points. On the downside the labels will "jump" (because of the special coordinate system) once the loop is changed to become a link between two different ports.

Other Changes

- Avoids cropping of edge label (and in-place editing) in EdgeRenderer

JGraph 5.4.4

BasicMarqueeHandler

The paint and overlay methods have been changed to include the current graph. Since these methods are always called in the context of a specific event, the graph is part of the caller's state, not the callee. To migrate your code, simply change the subclassers method signature to match that of the parent class, namely by inserting an argument in paint and overlay as in:

```
public void paint(JGraph graph, Graphics g);
public void overlay(JGraph graph, Graphics g, boolean clear);
```

BasicGraphUI.MouseHandler.handleEditTrigger

The handleEditTrigger returns a boolean value to indicate whether the editing has actually started. This is a fix for events that are outside the cell editor's hit region, in which case the cell is not selected when the edit click count is set to 1. This bug still exists in various Swing components. (Thanks to Timothy Wall from the Abbott project for this fix!)

New Helper Methods

- GraphLayoutCache.getEdges: Returns all visible, conntected edges for a cell (with various switches)
- GraphLayoutCache.edit(Map): Shortcut method to avoid passing null parameters
- GraphLayoutCache.editCell(Object, Map): Changes a single cell (no nested map required)
- GraphLayoutCache.getNeighbours: Returns the neighbours of a cell (with various switches)
- GraphConstants.merge(Map, Map): Merges two nested maps

Other Changes

- Changes GraphLayoutCache.setAllAttributeLocal to setAllAttributesLocal
- Replaced certain setViews calls in EdgeRenderer with assignment of view where setView is called directly afterwards for performance reasons
- Adds createGraph method, graph accessors, static inner classes in GraphEd
- Fixes GraphEd.connect to check against acceptsSource and acceptsSource

JGraph 5.4.3

Edge Labels

We use a new positioning for edge labels, and fixed moving of them with the mouse.

- X-coordinate: the percentual position on the length of the edge in direction of the edge
- Y-coordinate: the absolute offset, orthogonally to the edge

Note that this requires to change the position of the default label from (u/2,u/2) to (u/2,0), meaning 50% (in the center of the edge) with 0 px offset. For a label at the end of an edge with some 20 px offset you would use (u, 20), and for a label at the start of the edge at the other side of it you would use (0, -20) - eg. for multiplicities, where u = GraphConstants.PERMILLE.

JGraph 5.4.1

AskLocalAttributes

The isAskLocalAttributes field has been removed from GraphLayoutCache. The switch was used to control if the local attributes should be ignored. This switch was never used since the existence of local attributes normally also implies that they should be used.

AllAttributesLocal

As a "replacement" of the above, a allAttributesLocal switch was added to GraphLayoutCache. The switch controls if all attributes should be considered local. This is allows to control attributes without actually knowing them, and is useful in the context of "view-local" geometries.

Performance Improvements

The performance improvements are technically simple but quite considerable in wrt time consumption (eg. 5 times faster for inserting 10'000 nodes). They consist of removing model.contains calls in GraphLayoutCache and DefaultGraphModel.

Insets

The inset attribute is used in BasicGraphUI.getPreferredSize to add an inset to the default size returned by the respective renderer. The inset attribute now has a non-final default value in GraphConstants.DEFAULTINSET. Note that changing this value affects all graph instances.

Other Changes

- A method to snap a Rectangle2D to the grid (if it is active) has been added to JGraph.
- The GraphLayoutCache constructor adds a hiddenSet parameter.
- The AttributeMap does no longer contain default bounds.
- A HugeGraphTest example has been added.
- New hooks have been added to GraphEd (example): createEdgeAttributes, createGroupCell, createDefaultGraphEdge. In addition, the createDefaultGraphCell adds the port to the cell, not the calling method.

JGraph 5.4

Event Notification in GraphLayoutCache

The GraphLayoutCache does no longer implement the Observable interface, it now has a full-featured event notification mechanism. The listener should implement the GraphLayoutCacheListener interface and add the instance using the addGraphLayoutCacheListener method on the layout cache. The layout cache delivers GraphLayoutCacheEvents from which you can retrieve the new and the previous attributes. (Note: This will be required to optimize repaint upon changes.) The getInserted and getRemoved methods return the cells that have been shown or hidden.

How to migrate?

For code migration all existing observers should now implement the above interface with the respective method and remove the observer's update method:

```
class ACacheListener implements GraphLayoutCacheListener {
   public void graphLayoutCacheChanged(GraphLayoutCacheEvent e) {
      ...
```

AttributeMap Does Not Store User Object

This fundamental change has the following advantages:

- No custom storage map injection at cell creation time, i.e. so no setAttributeMap() call required
- Graphcells must no longer keep the user object and the attribute map in sync
- No double-referencing of the same user object from graphcell and attribute map

On the downside this adds another control attribute, in other words: You can not use attributes with the name value in a storage map.

How to migrate?

If you are using custom user objects then you should have implemented a custom attribute map with at least a valueChanged and clone method to take care of your custom user objects. If the attribute map only contains these two methods it is no longer required. Instead, you should create or use the custom graph model and override the following two methods:

```
public class GPGraphModel extends DefaultGraphModel {
   protected Object cloneUserObject(Object userObject) {
           ...
   }
   protected Object valueForCellChanged(Object cell, Object newValue) {
           ...
   }
}
```

If the attribute map is no longer required then all calls to cell.setAttributes that were used to inject the map may also be removed. (See GPGraphModel in the JGraphpad project for an example.)

New Methods

3 useful static helper methods have been added to the DefaultGraphModel:

- DefaultGraphModel.setSource/Target: Sets the source or target of an edge
- DefaultGraphModel.getUserObject: Gets the user object from a graph cell

In the GraphModelChange the getConnectionSet and getParentMap methods have been added.

Other Changes

3 new attributes are available to control cell behaviour, 1 attribute has been renamed:

- selectable/childrenSelectable: Enable/disable selection of cells and children
- constrained: Forces constrained sizing on a cell
- groupBorder attribute is now called inset

The JGraph.convertValueToString method has been changed to no longer map the cell. Instead, it uses the cell's toString method to determine the value. The method still handles view-local values though. See source code and javadocs for more details on the implementation.

Abbott Tests

Timothy Wall has contributed test code for the Abbot testing framework. Please see the test directory. You must download the latest version of Abbot to compile and run the tests.

JGraph 5.3

Automatic Selection

The graph layout cache offers the setSelectsLocalInsertedCells and setSelectsAllInsertedCells methods for automatic selection. The first method will select all cells which are inserted through the local cache, while the latter will select all inserted cells that are visible. (Therefore it is not possible to select cells in all but the local layout cache.)

Attribute Maps

The AttributeMap had quite some changes over time. For migration it is important to understand that we have split the use of such maps into transport and storage. The transport objects are normal maps, as they do not need to override certain methods. The storage maps are AttributeMaps, which are no longer created with a static or non-static hook. The storage maps are only used when replacing the cell's or cellview's attributes field, otherwise one should use a transport map, such as a Hashtable.

Storage Maps

Here is how to set a storage map for a cell:

```
DefaultGraphCell cell = new DefaultGraphCell();
cell.setAttributes(new MyAttributeMap());
```

Transport Maps

This changes the vertex background color to blue:

```
Hashtable map = new Hashtable();
```

```
GraphConstants.setBackground(map, Color.BLUE);
graph.getGraphLayoutCache().edit(new Object[]{vertex}, map);
```

Note that by using the edit(Object[], Map) method we do no longer require a nested map.

New Methods

Many of the methods found in JGraphUtilities (JGraphAddons) have been moved to the DefaultGraphModel and GraphLayoutCache including:

- GraphLayoutCache.insertEdge: Inserts an edge
- GraphLayoutCache.insertVertex: Inserts a vertex
- GraphLayoutCache.insertGroup: Inserts a group

JGraph 5.2

JGraph.setSelectNewCells

This method was replaced by setSelectClonedCells, which uses the following pattern when cloning cells. The actual functionality of selecting newly inserted cells was removed from the core. Use the following pattern instead:

```
graph.getGraphLayoutCache().insert(cells, ...);
graph.setSelectionCells(cells);
```

DefaultCellViewFactory

JGraph is no longer a cell view factory. Instead, the cell view factory is a standalone object that is referenced from the layout cache. Also, the cell view factory is no longer in charge of updating the auto size, this is done in the basic graph ui. Use the following code to create custom cell views:

```
graph.getGraphLayoutCache().setFactory(new DefaultCellViewFactory() {
   public CellView createView(GraphModel model, Object cell) {
      ...
   }
}
```

Extended Observer Pattern

The layout cache does now provide a getChanged method to indicate the changed cell views to its observers. The changed cell views are collected using addChanged before the call to notifyObservers. You must still call setChanged to notify the observers. The changed cell views are cleared when clearChanged is called.

Standalone GraphLayoutCache and Cell Views

For the cell views, the constructor now only takes the cell. The refresh method is used to refresh on a model-level change, and adds the graph model as a parameter. The layout cache is no longer in charge of auto-sizing and selection of new cells. Constructors have been changed to take a model and a factory. This change allows to use GraphLayoutCache and contained cell views in more than one JGraph instance.