Gephi Toolkit Tutorial

The Gephi Toolkit project package essential modules (Graph, Layout, Filters, IO...) in a standard Java library, which any Java project can use for getting things done.

The toolkit is just a single JAR that anyone could reuse in a Java program. This tutorial aims to introduce the project, show possibilities and start write some code.

Get Gephi Toolkit

The content of this tutorial, and more, can be found in the toolkit-demos archive.

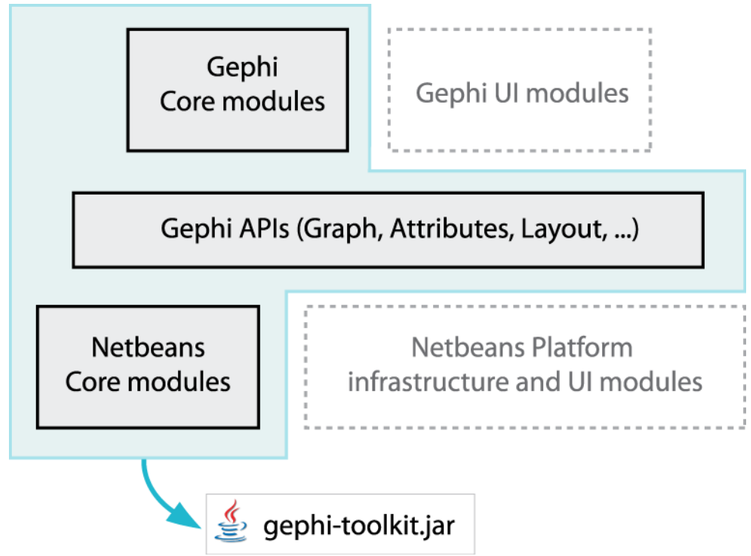
Last updated July 13th, 2010

Introduction

Gephi toolkit is a way to reuse Gephi features with scripts. It has the richness of Gephi features and the flexibility of a library.

Gephi is designed in a modular way:

- Splitted into different modules
- For instance, a module for the graph structure, a module for layout, a module for filter, ...
- Business modules are separated from UI modules
- Core modules can be extracted, no extra dependency



The toolkit wraps core modules, without user interaces in a single JAR library. There is no additionnal code in Gephi to allow this. The desktop application and the tookit relies exactly on the same modules.

Use-cases

Toolkit possible use-case:

* **Create a layout program**	Receive a GEXF file, layout it, and return the resulting GEXF file.
* **Complete chain**	From an imput graph file (DOT, GML, GEXF, ...) to the resulting PDF.
* **Java applet**	Use Gephi modules to store, filter,layout and build a new visualization applet on top of it.
* **Servlet**	Create graph snippets images (PNG) automatically in a servlet.

Find Toolkit Documentation

- Toolkit Portal on Gephi Wiki
- Toolkit Javadoc
- References
- Toolkit-demos, download sources

Create Project

A project needs to be created for using the toolkit features. Creating a new project also creates a workspace. Workspaces are the container for all data.

A project can have several workspaces, independent from each other. The workspace is often asked by modules, so you need to create one at startup.

```
//Init a project - and therefore a workspace
ProjectController pc = Lookup.getDefault().lookup(ProjectController.class);
pc.newProject();
Workspace workspace = pc.getCurrentWorkspace();
```

What is Lookup ?

Lookup is a very useful way to retrieve services and find instances in a modular application like Gephi. Controllers are singletons and can be found by looking into the general Lookup: Lookup.getDefault(). So Lookup is just a container, where modules put objects in it. Other modules get these objects from their class name.

Import Graph File

Importing data from files supported by Gephi.The import is done in two steps. First, all data are put in a Container, which is separated from the main data structure. The second steps is “processing”, when data in the container are appended to the graph structure and attributes.

The API also allows to import data from Reader and InputStream.

```
//Import file
ImportController importController = Lookup.getDefault().lookup(ImportController.class);
Container container;
try {
   File file = new File(getClass().getResource(“/org/gephi/toolkit/demos/resources/test.gexf”).toURI());
   container = importController.importFile(file);
   container.getLoader().setEdgeDefault(EdgeDefault.DIRECTED);   //Force DIRECTED
   container.setAllowAutoNode(false);  //Don’t create missing nodes
} catch (Exception ex) {
   ex.printStackTrace();
   return;
}

//Append imported data to GraphAPI
importController.process(container, new DefaultProcessor(), workspace);
```

Manipulate Graph API (1/2)

The Graph API's role is to store and serve the graph structure. Let's see how to create a graph programatically.

```
//Get a graph model - it exists because we have a workspace
GraphModel graphModel = Lookup.getDefault().lookup(GraphController.class).getModel();

//Create two nodes
Node n0 = graphModel.factory().newNode("n0");
n0.getNodeData().setLabel("Node 0");
Node n1 = graphModel.factory().newNode("n1");
n1.getNodeData().setLabel("Node 1");

//Create an edge - directed and weight 1
Edge e1 = graphModel.factory().newEdge(n1, n0, 1f, true);

//Append as a Directed Graph
DirectedGraph directedGraph = graphModel.getDirectedGraph();
directedGraph.addNode(n0);
directedGraph.addNode(n1);
directedGraph.addEdge(e1);
```

Manipulate Graph API (2/2)

Quick overview of Graph API's possibilities.

```
//Count nodes and edges
System.out.println("Nodes: "+directedGraph.getNodeCount()+" Edges: "+directedGraph.getEdgeCount());

//Iterate over nodes
for(Node n : directedGraph.getNodes()) {
   Node[] neighbors = directedGraph.getNeighbors(n).toArray();
   System.out.println(n.getNodeData().getLabel()+" has "+neighbors.length+" neighbors");
}

//Find node by id
Node node2 = directedGraph.getNode("n2");

//Modify the graph while reading
//Due to locking, you need to use toArray() on Iterable to be able to modify the graph in a read loop
for(Node n : directedGraph.getNodes().toArray()) {
   directedGraph.removeNode(n);
}
```

Import from RDBMS

The database format must be “Edge List”, basically a table for nodes and a table for edges. To be found by the importer, you need to have following columns: ID and LABEL for nodes and SOURCE, TARGET and WEIGHT for edges. Any other column will be imported as attributes.

```
//Import database
ImportController importController = Lookup.getDefault().lookup(ImportController.class);
EdgeListDatabaseImpl db = new EdgeListDatabaseImpl();
db.setDBName(“test”);
db.setHost(“localhost”);
db.setUsername(“root”);
db.setPasswd(“”);
db.setSQLDriver(new MySQLDriver());
//db.setSQLDriver(new PostgreSQLDriver());
//db.setSQLDriver(new SQLServerDriver());
db.setPort(3306);
db.setNodeQuery(“SELECT nodes.id AS id, nodes.label AS label, nodes.url FROM nodes”);
db.setEdgeQuery(“SELECT edges.source AS source, edges.target AS target, edges.name AS label, edges.
weight AS weight FROM edges”);
Container container = importController.importDatabase(db, new ImporterEdgeList());

//Append imported data to GraphAPI
importController.process(container, new DefaultProcessor(), workspace);
```

Layout - Manual

Layouts are iterative algorithms that change nodes' position at each pass. Each layout algorithm needs to get a graph model to be able to query the network. Parameters can be set directly to the layout.

Use ForceAtlas, FruchtermanReingold or LabelAdjust algorithm in a similar way.

```
//Get graph model of current workspace
GraphModel graphModel = Lookup.getDefault().lookup(GraphController.class).getModel();

//Run YifanHuLayout for 100 passes - The layout always takes the current visible view
YifanHuLayout layout = new YifanHuLayout(null, new StepDisplacement(1f));
layout.setGraphModel(graphModel);
layout.resetPropertiesValues();
layout.setOptimalDistance(200f);

layout.initAlgo();
for (int i = 0; i < 100 && layout.canAlgo(); i++) {
   layout.goAlgo();
}
```

Layout - Auto

With AutoLayout class, you can set a layout duration, and an execution ratio for several layouts. For instance you set 0.8 for a Yifan Hu algorithm and 0.2 for Label Adjust. If execution time is 100 seconds, the first algorithm runs for 80 seconds and the second for 20 seconds. It also allows to change property values dynamically, at a certain ratio or interpolated if values are numerical.

```
//Layout for 1 minute
AutoLayout autoLayout = new AutoLayout(1, TimeUnit.MINUTES);
autoLayout.setGraphModel(graphModel);
YifanHuLayout firstLayout = new YifanHuLayout(null, new StepDisplacement(1f));
ForceAtlasLayout secondLayout = new ForceAtlasLayout(null);

AutoLayout.DynamicProperty adjustBySizeProperty = AutoLayout.createDynamicProperty("Adjust by Sizes", Boolean.TRUE, 0.1f);//True after 10% of layout time
AutoLayout.DynamicProperty repulsionProperty = AutoLayout.createDynamicProperty("Repulsion strength", new Double(500.), 0f);//500 for the complete period

autoLayout.addLayout(firstLayout, 0.5f);
autoLayout.addLayout(secondLayout, 0.5f, new AutoLayout.DynamicProperty[]{adjustBySizeProperty, repulsionProperty});

autoLayout.execute();
```

Statistics and metrics

How to execute a metric algorithm and get the result for each node. The algorithm write values in a column it creates. It is also possible to get the statistics report, as in Gephi.

```
//Get graph model and attribute model of current workspace
GraphModel graphModel = Lookup.getDefault().lookup(GraphController.class).getModel();
AttributeModel attributeModel = Lookup.getDefault().lookup(AttributeController.class).getModel();

//Get Centrality
GraphDistance distance = new GraphDistance();
distance.setDirected(true);
distance.execute(graphModel, attributeModel);

//Get Centrality column created
AttributeColumn col = attributeModel.getNodeTable().getColumn(GraphDistance.BETWEENNESS);

//Iterate over values
for (Node n : graph.getNodes()) {
   Double centrality = (Double)n.getNodeData().getAttributes().getValue(col);
}
```

Ranking - Color

Ranking can be used with any numerical column, including the degree. A transformer is applied to change the color, the size, the label color or the label size of the node or the edge. An interpolator can be set to the transformer to modify the interpolation. The ranking scales the data between 0 and 1.

```
//Rank color by Degree
RankingController rankingController = Lookup.getDefault().lookup(RankingController.class);
NodeRanking degreeRanking = rankingController.getRankingModel().getDegreeRanking();
ColorTransformer colorTransformer = rankingController.getObjectColorTransformer(degreeRanking);
colorTransformer.setColors(new Color[]{new Color(0xFEF0D9), new Color(0xB30000)});
rankingController.transform(colorTransformer);
```

Ranking - Size

The result of the centrality algorithm is used here for the ranking. The metric is written in an AttributeColumn, which is used to create the NodeRanking object.

The ranking is always applied in the current workspace.

```
//Get Centrality
GraphDistance distance = new GraphDistance();
distance.setDirected(true);
distance.execute(graphModel, attributeModel);

//Rank size by centrality
RankingController rc = Lookup.getDefault().lookup(RankingController.class);
AttributeColumn col = attributeModel.getNodeTable().getColumn(GraphDistance.BETWEENNESS);

NodeRanking centralityRanking = rc.getRankingModel().getNodeAttributeRanking(centralityColumn);
SizeTransformer sizeTransformer = rankingController.getObjectSizeTransformer(col);
sizeTransformer.setMinSize(3);
sizeTransformer.setMaxSize(20);
rankingController.transform(sizeTransformer);
```

Partition

Partition works with attribute column that have a set of different values. It colors nodes or edges with the same value. The Modularity algorithm in Gephi is a community-detection algorithm which writes for each node to which community it belongs. The partition object allows to retrieve all values and the array of nodes for each of it.

```
//Run modularity algorithm - community detection
Modularity modularity = new Modularity();
modularity.execute(graphModel, attributeModel);

//Partition with ‘modularity_class’, just created by Modularity algorithm
AttributeColumn modColumn = attributeModel.getNodeTable().getColumn(Modularity.MODULARITY_CLASS);
Partition p = partitionController.buildPartition(modColumn, graph);

System.out.println(p.getPartsCount() + “ partitions found”);

NodeColorTransformer nodeColorTransformer = new NodeColorTransformer();
nodeColorTransformer.randomizeColors(p);
partitionController.transform(p, nodeColorTransformer);
```

Filter (1/4)

Shows how to create an execute a filter query. When a filter query is executed, it creates a new graph view, which is a copy of the graph structure that went through the filter pipeline. Several filters can be chained by setting sub-queries. A query is a tree where the root is the last executed filter.

```
//Filter, remove degree < 10
FilterController filterController = Lookup.getDefault().lookup(FilterController.class);
DegreeRangeFilter degreeFilter = new DegreeRangeFilter();
degreeFilter.setRange(new Range(10, Integer.MAX_VALUE));     //Remove nodes with degree < 10
Query query = filterController.createQuery(degreeFilter);
GraphView view = filterController.filter(query);
graphModel.setVisibleView(view);    //Set the filter result as the visible view
```

The graph from the graph view can be obtained from the GraphModel by calling:

```
Graph graph = graphModel.getGraph(view);
```

or, if the view has been set as the visible view like above:

```
Graph graph = graphModel.getGraphVisible(view);
```

Filter (2/4)

The filter here is a PartitionFilter, which is built from a Partition created from a column named ‘source’.

```
//Filter, keep partition ‘Blogarama’. Build partition with ‘source’ column in the data
PartitionController pc = Lookup.getDefault().lookup(PartitionController.class);
Partition p = pc.buildPartition(attributeModel.getNodeTable().getColumn(“source”), graph);

NodePartitionFilter partitionFilter = new NodePartitionFilter(p);
partitionFilter.unselectAll();
partitionFilter.addPart(p.getPartFromValue(“Blogarama”));

Query query = filterController.createQuery(partitionFilter);
GraphView view = filterController.filter(query);
graphModel.setVisibleView(view);    //Set the filter result as the visible view
```

Filter (3/4)

How to use the Ego filter. The depth can a number or Integer.MAX_VALUE for the connected component.

```
//Ego filter
EgoFilter egoFilter = new EgoFilter();
egoFilter.setPattern("obamablog.com"); //Regex accepted
egoFilter.setDepth(1);
Query queryEgo = filterController.createQuery(egoFilter);
GraphView viewEgo = filterController.filter(queryEgo);
graphModel.setVisibleView(viewEgo);    //Set the filter result as the visible view
```

Filter (4/4)

It is possible to use Intersection and Union operators to create queries. Simply add other queries as sub-queries of the operator.

```
//Combine two filters with AND - Set query1 and query2 as sub-query of AND
IntersectionOperator intersectionOperator = new IntersectionOperator();
Query andQuery = filterController.createQuery(intersectionOperator);
filterController.setSubQuery(andQuery, query1);
filterController.setSubQuery(andQuery, query2);
GraphView view = filterController.filter(andQuery);
graphModel.setVisibleView(view);    //Set the filter result as the visible view
```

Preview

Preview is the last step before export and allows display customization and aesthetics refinements. It works with the current workspace. The example below shows how to change edge coloring mode, edge thickness and label font size.

```
//Preview
PreviewModel model = Lookup.getDefault().lookup(PreviewController.class).getModel();
model.getNodeSupervisor().setShowNodeLabels(Boolean.TRUE);
ColorizerFactory colorizerFactory = Lookup.getDefault().lookup(ColorizerFactory.class);
model.getUniEdgeSupervisor().setColorizer((EdgeColorizer) colorizerFactory.createCustomColorMode(Color.
LIGHT_GRAY));    //Set edges gray
model.getBiEdgeSupervisor().setColorizer((EdgeColorizer) colorizerFactory.createCustomColorMode(Color.
RED));    //Set mutual edges red
model.getUniEdgeSupervisor().setEdgeScale(0.1f);
model.getBiEdgeSupervisor().setEdgeScale(0.1f);
model.getNodeSupervisor().setBaseNodeLabelFont(model.getNodeSupervisor().getBaseNodeLabelFont().
deriveFont(8));
```

Export Graph File

Exports the current workspace to a file. For a simple export just call the exportFile() method with the right file extension. For customizing the exporter, query it from the export controller and change settings.

```
//Export full graph
ExportController ec = Lookup.getDefault().lookup(ExportController.class);
try {
    ec.exportFile(new File("test-full.gexf"));
} catch (IOException ex) {
    ex.printStackTrace();
    return;
}

//Export only visible graph
GraphExporter exporter = (GraphExporter) ec.getExporter("gexf");     //Get GEXF exporter
exporter.setExportVisible(true);  //Only exports the visible (filtered) graph
try {
    ec.exportFile(new File("test-visible.gexf"), exporter);
} catch (IOException ex) {
    ex.printStackTrace();
    return;
}
```

Export PDF/SVG

The PDF and SVG exporter works with settings made in Preview. As with all exporters, it is possible to write the result in a Writer or OutputStream. In the exemple below the PDFExporter is retrieved and configured.

```
//Simple PDF export
ExportController ec = Lookup.getDefault().lookup(ExportController.class);
try {
   ec.exportFile(new File("simple.pdf"));
} catch (IOException ex) {
   ex.printStackTrace();
   return;
}

//PDF Exporter config and export to Byte array
PDFExporter pdfExporter = (PDFExporter) ec.getExporter("pdf");
pdfExporter.setPageSize(PageSize.A0);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ec.exportStream(baos, pdfExporter);
byte[] pdf = baos.toByteArray();
```

Conclusion

Share your ideas and find more help on the Gephi forum, the community is eager to help you get things done!

Please also consider the gephi-toolkit is a young project and remains imperfect. We hope this project helps all developers to use network visualization and analytics in their project and build great systems.

We are looking for your help, join the network and become a contributor!