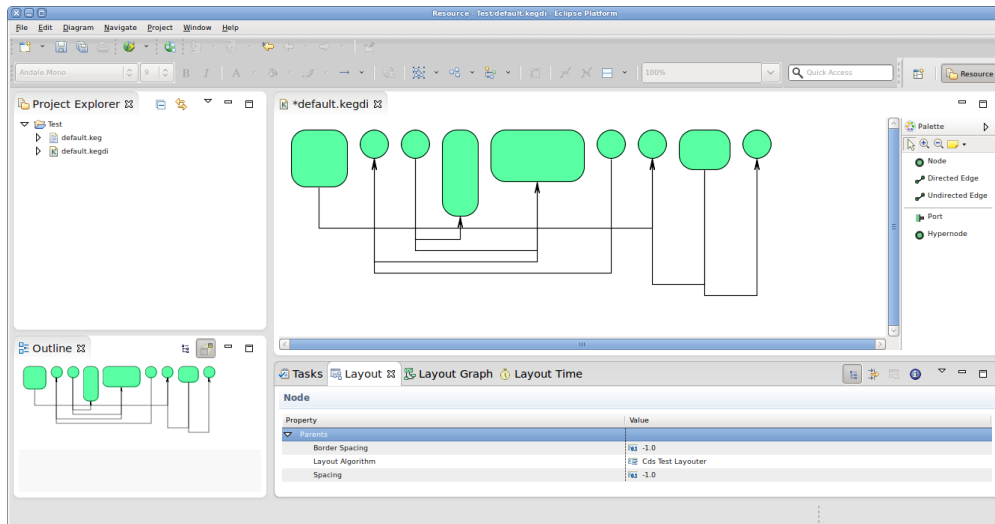


KIML

Welcome to the second tutorial! We will work our way through installing a proper Eclipse setup and developing a first very basic layout algorithm. The layout algorithm will integrate with KIML (KIELER Infrastructure for Meta-Layout), our very own framework that connects graphical editors with layout algorithms. Once you're finished, you should be able to create new Eclipse plug-ins and know how to write layout algorithms for KIML. And you should have a running Eclipse-based application that should look something like this:



- Preliminaries
 - [Required Software](#)
 - [General Remarks](#)
 - [Finding Documentation](#)
 - [Preparing the Repository](#)
 - [Finding the KIML Sources](#)
- Developing Your First Layout Algorithm
 - [Setting Up Your Workspace](#)
 - [Adding a New Plug-in](#)
 - [Writing the Layout Algorithm](#)
 - [Before You Run Away...](#)

Preliminaries

There's a few things to do before we dive into the tutorial itself. For example, to do Eclipse programming, you will have to get your hands on an Eclipse installation first. Read through the following sections to get ready for the tutorial tasks.

Required Software

For this tutorial, we need you to have Eclipse and Git installed:

1. Install Eclipse. For what we do, we recommend installing the Eclipse Modeling Tools, with a few extras. Our [Wiki page on getting Eclipse](#) has the details: simply follow the instructions for downloading and installing Eclipse and you should be set.
2. You should already have obtained a working Git installation for the first tutorial.

General Remarks

Over the course of this tutorial, you will be writing a bit of code. Here's a few rules we ask you to follow:

- All the Java code you write as part of tutorials should be in packages with the prefix `de.cau.cs.rtp.rak.login.tutorialN`, where `login` is your login name as used for your email address at the institute. This rule will apply to all tutorials – once we start with the actual practical projects, we will choose more meaningful package name.
- All Java classes, fields, and methods should be thoroughly documented with the standard [Javadoc](#) comment format. Javadoc comments are well supported by Eclipse through code completion, syntax highlighting, and further features that help you. The code inside your methods should also be well commented. Try to think about what kinds of information would help someone unfamiliar with your code understand it.
- During this tutorial, we will be using Git mostly from the command line instead of using Eclipse's built-in Git support. This is because we've found Eclipse's Git support to be too unstable and buggy for us to trust it completely.

Finding Documentation

During the tutorial, we will cover each topic only briefly, so it is always a good idea to find more information online. Here's some more resources that will prove helpful:

- [Java Platform, Standard Edition 6 API Specification](#)
As Java programmers, you will already know this one, but it's so important and helpful that it's worth repeating. The API documentation contains just about everything you need to know about the API provided by Java6.
- [Eclipse Help System](#)
Eclipse comes with its own help system that contains a wealth of information. You will be spending most of your time in the *Platform Plug-in Developer Guide*, which contains the following three important sections:
 - **Programmer's Guide**
When you encounter a new topic, such as SWT or JFace, the Programmer's Guide often contains helpful articles to give you a first overview. Recommended reading.
 - **References -> API Reference**
One of the two most important parts of the Eclipse Help System, the API Reference contains the Javadoc documentation of all Eclipse framework classes. Extremely helpful.
 - **References -> Extension Points Reference**
The other of the two most important parts of the Eclipse Help System, the Extension Point Reference lists all extension points of the Eclipse framework along with information about what they are and how to use them. Also extremely helpful.
- [Eclipsepedia](#)
The official Eclipse Wiki. Contains a wealth of information on Eclipse programming.
- [Eclipse Resources](#)
Provides forums, tutorials, articles, presentations, etc. on Eclipse and Eclipse-related topics.

You will find that despite of all of these resources Eclipse is still not as well commented and documented as we'd like it to be. Finding out how stuff works in the world of Eclipse can thus sometimes be a challenge. However, this does not only apply to you, but also to many people who are conveniently connected by something called *The Internet*. It should go without saying that if all else fails, [Google](#) often turns up great tutorials or solutions to problems you may run into. And if it doesn't, Miro and I will be happy to help you as well.

As far as KIML and layout algorithms are concerned, you can always refer to our Wiki which has a section about [KIML and the KIELER layout projects](#). The documentation is not complete, however, so feel free to ask Miro or Christoph Daniel for help if you have questions that the documentation fails to answer.

Preparing the Repository

We have created a Git repository for everyone to do his tutorials in. You can access the repository online through our Stash tool [over here](#). Clone that repository:

1. Open a console window and navigate to an empty directory that the repository should be placed in.
2. Enter the command `git clone ssh://git@git.rtsys.informatik.uni-kiel.de:7999/PRAK/13ss-layout-tutorials.git .` (including the final dot, which tells git to clone the repository into the current directory instead of a subdirectory)
3. You should now have a clone of the repository in the current directory.

You will use this repository for all your tutorial work, along with everyone else. To make sure that you don't interfere with each other, everyone will work on a different branch. This is not exactly how people usually use Git, but goes to demonstrate Git's flexibility... Add a branch for you to work in:

1. Enter `git checkout -b login_name`

You have just added and checked out a new branch. Everything you commit will go to this branch. To push your local commits to the server (which you will need to do so we can access your results), do the following:

1. Enter `git push origin login_name`

You would usually have to enter `git pull` first, but since nobody will mess with your branch anyway this won't be necessary. By the way, you only need to mention `origin login_name` with the first `git push`, since Git doesn't know where to push the branch yet. After the first time, Git remembers the information and it will be enough to just enter `git push`.

Finding the KIML Sources

If you want to develop a layout algorithm using KIML, you will have to get your hands at the KIML source code first. Of course, the code is available through a Git repository.

1. Open a console window and navigate to an empty directory that the repository should be placed in.
2. Enter the command `git clone ssh://git@git.rtsys.informatik.uni-kiel.de:7999/KIELER/pragmatics.git .`
3. You should now have a clone of the repository in the current directory.

KIML is implemented as an Eclipse plug-in that you will have to import into your Eclipse workspace. We won't do this now; it will be one of the first steps in the development of your layout algorithm.

Developing Your First Layout Algorithm

Now that the preliminaries are out of the way, it's time to develop your first layout algorithm! It will, however, be a very simple one. This tutorial focuses on creating Eclipse plug-ins and on learning how to develop with KIML; thinking of and implementing cool layout algorithms is what the rest of the practical will focus on, and that is where the fun will be had!



Remember to replace each occurrence of `login_name` with your own login name (e.g. `mzp`), and each occurrence of `Login_name` with your capitalized login name (e.g. `Mzp`).

Setting Up Your Workspace

You will start by importing the plug-ins necessary to program with KIML.

1. Start Eclipse and create a new workspace.
2. Setup your workspace as explained in [this guide](#).
3. We will now make the two local clones of our Git repositories known to Eclipse. To that end, open the *Git Repository Exploring* perspective through *Window -> Open Perspective -> Other*.
4. Click on *Add an existing local Git repository* and choose the location of the tutorial repository. Note that when you open the repository entry, the branch you previously checked out is marked as the current branch under *Branches -> Local*.
5. Add the KIML repository.
6. We will now import the projects required for KIML development to your workspace. Right-click on the KIML repository and choose *Import Projects*.
7. Choose *Import existing projects*, and select the *plugins* folder from the *Working Directory*. Then click *Next*.
8. Import the following plug-ins. This constitutes a basic configuration for the development of layout algorithms.
 - `de.cau.cs.kieler.core`
 - `de.cau.cs.kieler.core.kgraph`
 - `de.cau.cs.kieler.kiml`
 - `de.cau.cs.kieler.kiml.gmf`
 - `de.cau.cs.kieler.kiml.service`
 - `de.cau.cs.kieler.kiml.ui`
 - `de.cau.cs.kieler.klay.layered`
9. To actually test your layout algorithms, you will need some kind of simple graph editor. The following plug-ins will add our KEG editor to your installation, which is just that.
 - `de.cau.cs.kieler.core.annotations`
 - `de.cau.cs.kieler.core.kgraph.edit`
 - `de.cau.cs.kieler.core.kivi`
 - `de.cau.cs.kieler.core.model`
 - `de.cau.cs.kieler.core.model.gmf`
 - `de.cau.cs.kieler.core.ui`
 - `de.cau.cs.kieler.karma`
 - `de.cau.cs.kieler.keg`
 - `de.cau.cs.kieler.keg.diagram`
 - `de.cau.cs.kieler.keg.diagram.custom`
 - `de.cau.cs.kieler.keg.edit`

Adding a New Plug-in

We need to create a new plug-in to implement the layout algorithm in. Switch back to the Java or Plug-in Development perspective and follow these steps:

1. Click *File > New > Other... > Plug-in Development > Plug-in Project*.
2. Enter `de.cau.cs.rtrprk.login_name.tutorial2` as the project name. Uncheck *Use default location* and use `tutorial_repository_path/de.cau.cs.rtrprk.login_name.tutorial2` as the location. Click *Next*.
3. Set the version to `0.1.0.qualifier`, vendor to *Christian-Albrechts-Universität zu Kiel*, and execution environment to *J2SE-1.5*. (do this for all plug-ins that you create!)
4. Uncheck all checkboxes in the *Options* group and click *Finish*.
5. If Eclipse asks you whether the *Plug-in Development* perspective should be opened, choose either *Yes* or *No*. It doesn't make much of a difference anyway.

You should now commit your new, empty project to the Git repository. We will do that from within Eclipse.

1. Right-click your project in the *Package Explorer* and click *Team > Share Project...*
2. As the repository type, select *Git* and click *Next*.
3. Tell Eclipse what repository to add the project to. The repository you placed the project in is already preselected. Simply click *Finish*.
4. Since Git support is now enabled for the project, right-click it again and click *Team > Commit...*
5. Select all files, enter a (meaningful) commit message, and click *Commit*.

Writing the Layout Algorithm

When writing our layout algorithm, we are going to need to be able to access code defined in several other plug-ins. To do that, we need to add dependencies to those plug-ins:

1. In your new plug-in, open the file `META-INF/MANIFEST.MF`. The plug-in manifest editor will open. Open its *Dependencies* tab.
2. Add dependencies to the following plug-ins:
 - `de.cau.cs.kieler.core`
 - `de.cau.cs.kieler.core.kgraph`
 - `de.cau.cs.kieler.kiml`
3. Save the editor.

Layout algorithms interface with KIML by means of a layout provider class that has to be created and registered with KIML.

1. Right-click the source folder of your plug-in and click *New > Class*.
2. Set the package to `de.cau.cs.rtptrak.login_name.tutorial2`, enter `Login_nameLayoutProvider` as the class name, and select `de.cau.cs.kieler.kiml.AbstractLayoutProvider` as the superclass. (This will only be available through the *Browse* dialog if you have saved the plug-in manifest editor; if you haven't, Eclipse won't know about the new dependencies yet.)
3. Select *Generate comments* and click *Finish*.

Implement the layout provider class:

1. Add the following constants:

```
/** default value for spacing between nodes. */
private static final float DEFAULT_SPACING = 15.0f;
```

2. Use the following code as the skeleton of the `doLayout(...)` method:

```
progressMonitor.begin("Login_name layouter", 1);
KShapeLayout parentLayout = parentNode.getData(KShapeLayout.class);

float objectSpacing = parentLayout.getProperty(LayoutOptions.SPACING);
if (objectSpacing < 0) {
    objectSpacing = DEFAULT_SPACING;
}

float borderSpacing = parentLayout.getProperty(LayoutOptions.BORDER_SPACING);
if (borderSpacing < 0) {
    borderSpacing = DEFAULT_SPACING;
}

// TODO: Insert actual layout code.

progressMonitor.done();
```

3. It is now time to write the code that places the nodes. Your code should place them next to each other in a row, as seen in the screenshot at the beginning of the tutorial.

Tips

The following tips might come in handy...

- Read the documentation of the [KGraph](#) and [KLayoutData](#) meta models. The input to the layout algorithm is a `KNode` that has child `KNodes` for every node in the graph. Iterate over these nodes by iterating over the `getChildren()` list of the `parentNode` argument.
- Retrieve the size of a node and set its position later using the following code:

```
KShapeLayout nodeLayout = node.getData(KShapeLayout.class);

// Retrieving the size
float width = nodeLayout.getWidth();
float height = nodeLayout.getHeight();

// Setting the position
nodeLayout.setXpos(x);
nodeLayout.setYpos(y);
```

- `objectSpacing` is the spacing to be left between each pair of nodes.
- `borderSpacing` is the spacing to be left to the borders of the drawing. The top left node's coordinates must therefore be at least `(borderSpacing, borderSpacing)`.
- At the end of the method, set the width and height of `parentLayout` such that it is large enough to hold the whole drawing, including borders.
- A complete layout algorithm will of course also route the edges between the nodes. Ignore that for now – you will do this at a later step.

Before you can test your layout code, you will have to register your new layout provider with KIML.

1. Open the `META-INF/MANIFEST.MF` file again and switch to the *Extensions* tab.
2. Add an extension for `de.cau.cs.kieler.kiml.layout.layoutProviders`.
3. Right-click the extension and click *New > layoutAlgorithm*.
4. Set the name to `Login_name Test Layouter` and the class to your layout provider class name.
5. Right-click the new *layoutAlgorithm* and click *New > knownOption*. Set option to `de.cau.cs.kieler.spacing`.
6. Add another *knownOption* for `de.cau.cs.kieler.borderSpacing`.

7. Save the editor.

We will now have to add a new run configuration that will start an Eclipse instance with your layout code loaded into the application, ready to be used.

1. Click *Run > Debug Configurations...*
2. Right-click *Eclipse Application* and click *New*. Set the configuration's name to *Layout Test*.
3. In the *Arguments* tab, make sure the the program arguments include *-debug* and *-consoleLog*.
4. On the *Plug-ins* tab, set *Launch with* to *plug-ins selected below only*. Click *Deselect All*, check the *Workspace* item in the tree, and click *Add Required Plug-ins*.
5. Click *Apply* to save your changes and then *Debug* to start an Eclipse instance to test with.

Test the layouter in your new Eclipse instance:

1. Click *New > Project... > General > Project* and set the project name to something like *Test*.
2. Right-click the new project and click *New > Empty KEG Graph*. Enter a meaningful name and click *Finish*.
3. Put a few nodes into the diagram. To properly test your code, you will want to vary the sizes of the nodes. It may also be a good idea to get into the habit of giving each node a different name, such as *N1*, *N2*, etc. This will help you later if you have to debug your algorithm.
4. Open the *Layout* view through *Window > Show View > Other... > KIELER Layout > Layout*.
5. With your KEG diagram selected, set the *Layout Algorithm* option in the *Layout* view to your new algorithm.
6. Save your KEG diagram.
7. Trigger automatic layout by clicking the layout button in the toolbar, or by hitting *Ctrl+R L* (first *Ctrl+R*, then *L*).



Tip

You can see the direct output of your algorithm and the time it took to run it through the *Layout Graph* and *Layout Time* views. The views are available through the `de.cau.cs.kieler.kiml.debug` plug-in, which can be found in the `plugins-dev` folder of the KIML repository. You will learn more about debugging layout algorithms in a layout tutorial or presentation.

Once you're satisfied with your node placement code, it's time to take care of edge routing.

1. Add a new method that will implement the edge routing using the following skeleton code:

```
/**
 * Routes the edges connecting the nodes in the given graph.
 *
 * @param parentNode the graph whose edges to route.
 * @param yStart y coordinate of the start of the edge routing area.
 * @param objectSpacing the object spacing.
 * @return height used for edge routing.
 */
private float routeEdges(final KNode parentNode, final float yStart, final float objectSpacing) {
    // TODO: Implement edge routing

    return 0;
}
```

2. Add a call to `routeEdges(...)` in your `doLayout(...)` method and implement the latter.

Tips

Here's a few tips for implementing the edge routing:

- Each edge shall be drawn with three orthogonal line segments: one vertical segment below the start node, one vertical segment below the target node, and a horizontal segment that connects the two.
- The horizontal segments of two different edges shall not have the same y-coordinate. Two neighboring horizontal segments shall be placed at a distance of `objectSpacing`.
- See the screenshot at the top of the tutorial for an example.
- Find the edges in a graph by calling `getOutgoingEdges()` or `getIncomingEdges()` on the nodes.
- You can add bend points to edges through the edge's edge layout:

```
KEdgeLayout edgeLayout = edge.getData(KEdgeLayout.class);
KPoint bendPoint = KLayoutDataFactory.eINSTANCE.createKPoint();
edgeLayout.getBendPoints().add(bendPoint);
```

- You will want to clear the list of bend points of each edge layout before adding bend points to it. This will remove all bend points the edge had prior to invoking your layout algorithm.
- Set the values of the points returned by `getSourcePoint()` and `getTargetPoint()` according to the positions where an edge leaves its source node and reaches its target node.
- If you want, you can improve the edge routing code by allowing horizontal segments to share the same y-coordinate if that doesn't make them overlap. Your goal could be to produce an edge routing that uses as little space as possible.
- If that's not enough yet: can you find a way to find an order of the horizontal segments such that as few edge crossings as possible are produced?

Once you're done implementing the edge routing code, test it by running your debug configuration again, as before.

Before You Run Away...

...don't forget to commit your layout algorithm to your repository, and to push your commits to the tutorial repository on our server. If it's not there, we won't be able to see your work! 😊