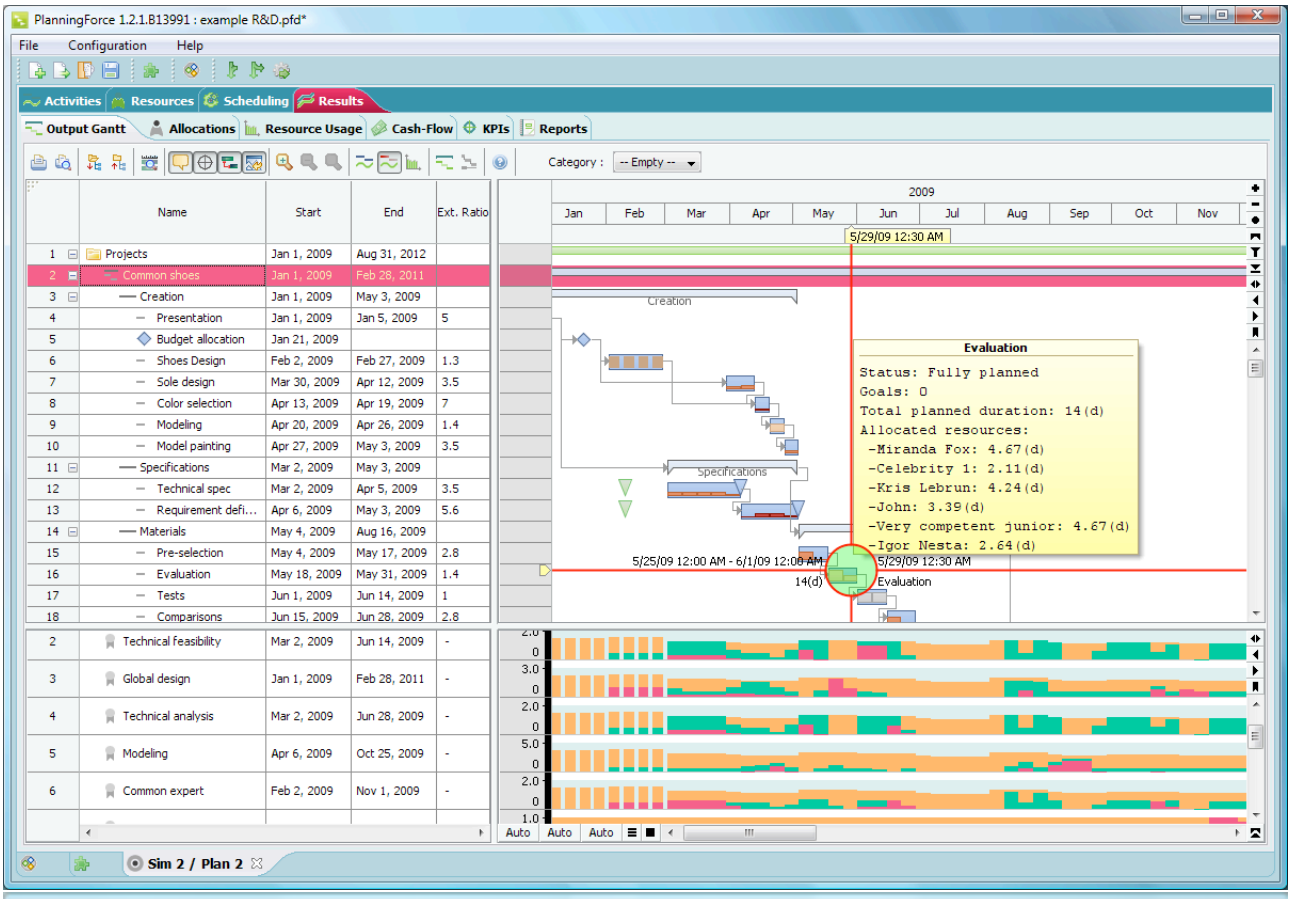
FlexGantt - Release 1

Getting Started



Dirk Lemmermann Software & Consulting
Asylweg 28
8134 Adliswil
Switzerland
www.dlsc.com

The image on the title page shows the application „PlanningForce“, which makes heavy use of FlexGantt. For more information on this product go to http://www.planningforce.com

FlexGantt Release 1.1.2

Document last updated on October 1st, 2008

Table of Content

Introduction

This document aims at providing enough information to get started using the FlexGantt component in your application.

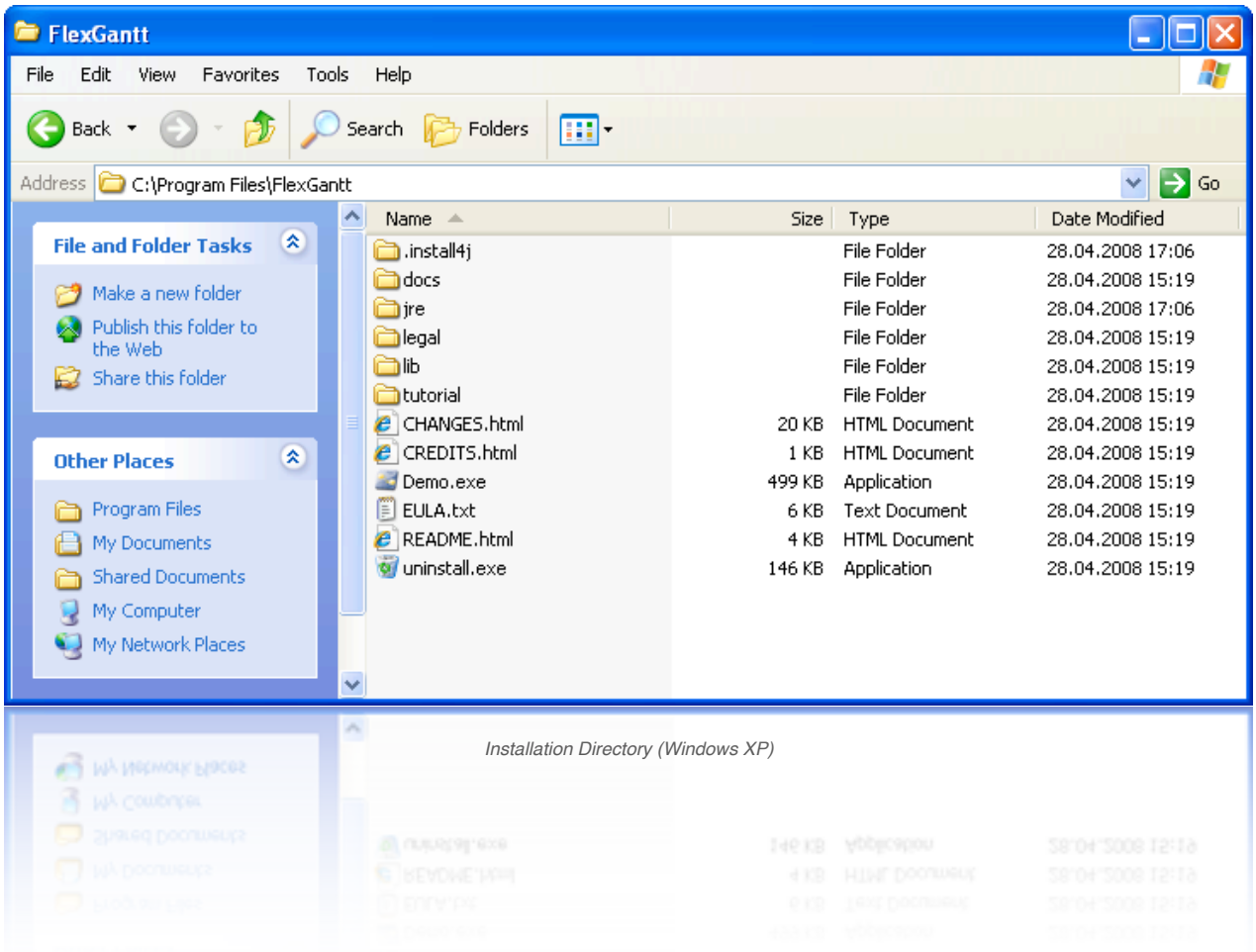
Contacting support via Email: dlemmermann@gmail.com
Contacting support via online forum: http://groups.google.com/group/flexgantt
FlexGantt product site: http://www.flexgantt.com
Bug Reports: http://www.dlsc.com:7070 (JIRA Bug Tracking Server)

Setting up FlexGantt

Classpath

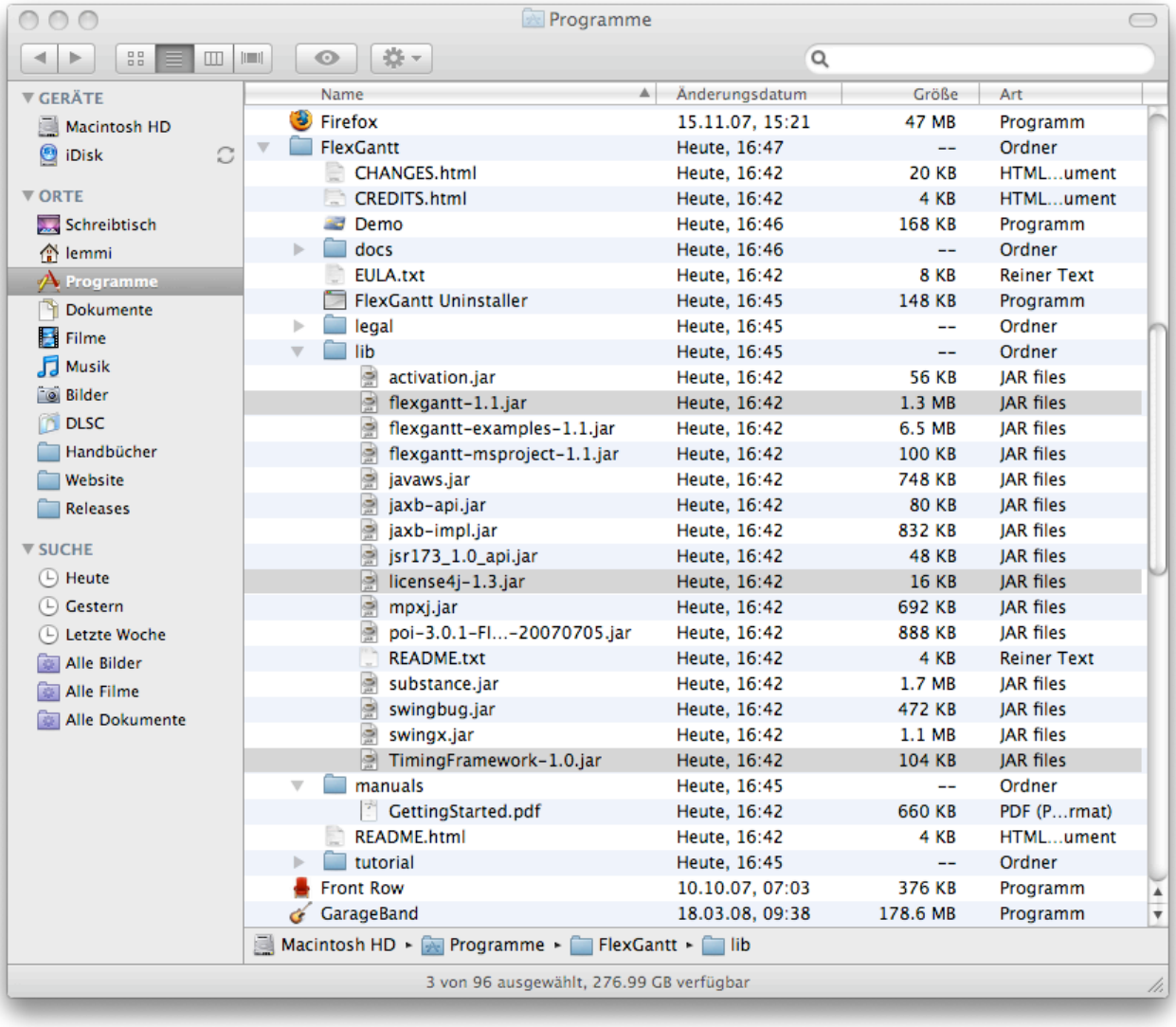
In order to use the Gantt chart component your application needs to find it. How to do this depends entirely on your environment but normally you add it to your classpath, possibly in the project settings in your IDE. The files to make available are *flexgantt-1.1.jar*, *TimingFramework-1.0.jar* and *license4j-1.4.jar*.

The JAR files can be found in the „lib“ subfolder within the installation directory. On Windows XP this looks like the following:



Installation Directory (Windows XP)

On a MacIntosh it looks like this:



Installation Directory (MacOS X)

The JAR files can be found in the „lib“ folder. It contains a long list of other files but you will only need the before mentioned three archives.

Add License Code

Since the product you are using is commercial it will need to be fed a license code before it can be used. This license code needs to be set before any FlexGantt classes are used or the framework will enter trial / evaluation mode.This is normally done in your application's *main()* method or some other place that is run very early in the startup sequence. This is done by calling:

```
com.dlsc.flexgantt.util.FlexGanttLicenseManager.setLicenceKey(String);
```

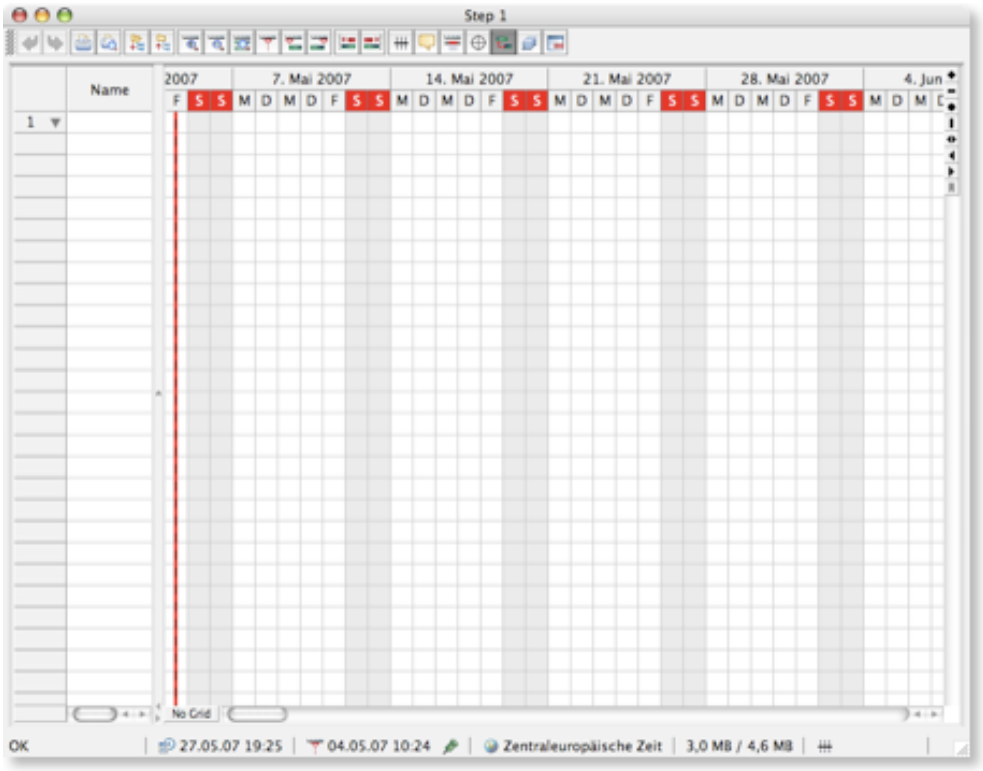
Setting up a Swing Application

In order to show the FlexGantt Gantt chart component you will have to create a a normal Swing application. How to do that is outside the scope of this document but it can be done in just a few lines. See the Swing trail in the Java Tutorial at Sun Microsystems site. It is currently located at: http://java.sun.com/docs/books/tutorial/uiswing/index.html

Tutorial

This tutorial tries to get you started on bringing up your first Gantt chart as quickly as possible.

Basic Setup



Tutorial Application after Basic Setup

The first thing needed is an instance of *GanttChart* (or *DualGanttChart* if split view functionallity is needed). In most cases it is sufficient to use the default empty constructor. There are several other constructors that allow you to pass in a model or a component factory or both. The default constructor uses an instance of *DefaultGanttChartModel* and *DefaultComponentFactory*.

```
GanttChart gc = new GanttChart();
```

This component could now be added to any window (frame or dialog) but FlexGantt provides a specialized frame called *GanttChartFrame* that adds out-of-the-box functionality that most scheduling applications need anyway. They are: busy cursor management, toolbar support, status bar support. We use this frame type because this is a quickstart guide and we are trying to get started as quickly as possible.

```
GanttChartFrame<GanttChart> frame = new GanttChartFrame<GanttChart>("Step 1", gc);
```

To ensure that the virtual machine exits when the frame gets closed we set the default close operation to EXIT_ON_CLOSE.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Now we only need to make the frame visible in order to see our first Gantt chart on the screen.

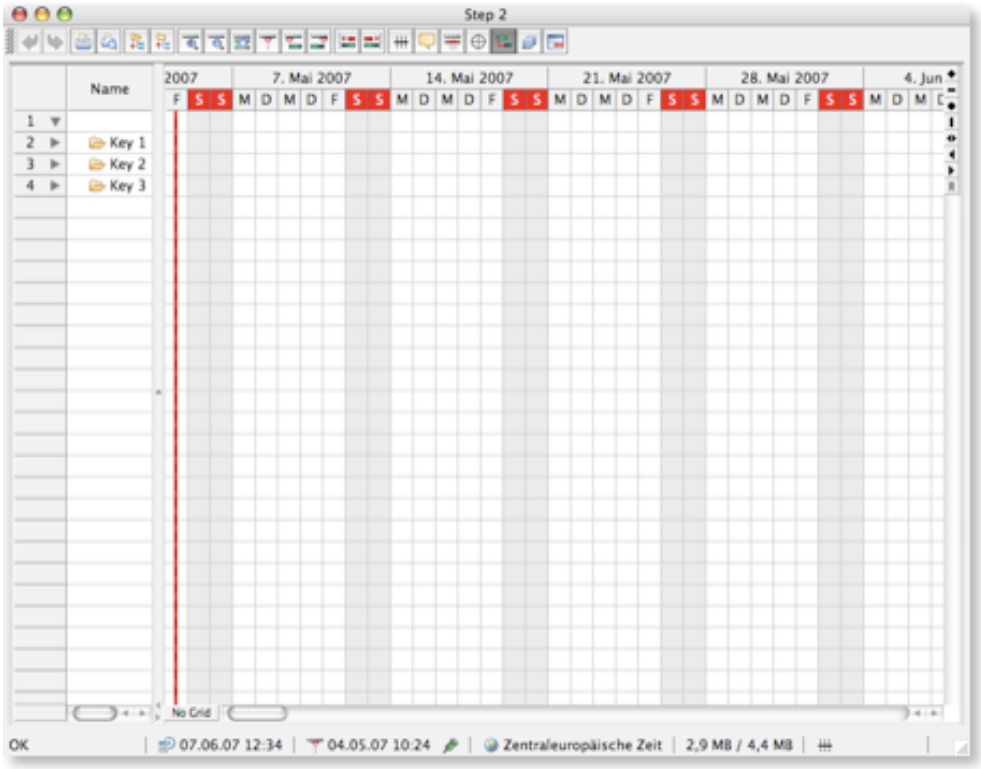
```
frame.setVisible(true);
```

All of the above steps combined and wrapped in a *main()* method result in the following short program:

```
/**
 * Copyright 2006 - 2008
 * Dirk Lemmermann Software & Consulting
 * http://www.dlsc.com
 */
package com.dlsc.flexgantt.examples.jumpstart;
```

```
import javax.swing.JFrame;

import com.dlsc.flexgantt.swing.GanttChart;
import com.dlsc.flexgantt.swing.GanttChartFrame;

/**
 * One of the steps used for the 'Jumpstart' tutorial. The step will create a
 * basic Gantt chart. The chart will be populated with a default model and a
 * single default node, which displays a default value as its key.
 *
 * @author Dirk Lemmermann
 */
public class Step1_Basic_Setup {

    /**
     * The example's main method.
     */
    public static void main(String[] args) {

        /*
         * Create a basic Gantt chart, which will use the default Gantt chart
         * model. The default model itself will use the default Gantt chart node
         * as a root object.
         */
        GanttChart gc = new GanttChart();

        /*
         * Add the Gantt chart to a specialized frame class. The frame will
         * automatically add a status bar and a glass pane (used for updating
         * the cursor when commands get executed).
         */
        GanttChartFrame<GanttChart> frame = new GanttChartFrame<GanttChart>("Step 1", gc);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /*
         * Show the frame. The split panel inside the Gantt chart will adjust to
         * the preferred size of the left-hand side (the tree table).
         */
        frame.setVisible(true);
    }
}
```

Tree Nodes



Tutorial Application after Tree Nodes

A Gantt chart consists of a left-hand side and a right-hand side. The left-hand side displays a tree table. In this step we are creating data that we use to fill the table. Any kind of object can be shown in the tree table but the easiest way to get to a working Gantt chart is to use the default model classes provided by FlexGantt. They are *DefaultGanttChartModel* and *DefaultGanttChartNode*. Since we have used the default *GanttChart* constructor in step 1 we already know that the model used by the Gantt chart is the default Gantt chart model, so we can retrieve it like this:

```
DefaultGanttChartModel model = (DefaultGanttChartModel) gc.getModel();
```

The root node of the default model is an instance of *DefaultGanttChartNode*:

```
DefaultGanttChartNode root = (DefaultGanttChartNode) model.getRoot();
```

Now we create our own nodes and add them to the root. We have to specify key values, which will show up in the key column of the tree table.

```
DefaultGanttChartNode node1 = new DefaultGanttChartNode("Node 1");
DefaultGanttChartNode node2 = new DefaultGanttChartNode("Node 2");
DefaultGanttChartNode node3 = new DefaultGanttChartNode("Node 3");
node1.setKey("Key 1");
node2.setKey("Key 2");
node3.setKey("Key 3");
root.add(node1);
root.add(node2);
root.add(node3);
```

Now we need to inform the model that the root node has changed its structure (new children). This will cause events to be fired by the model. These events are received by the tree table, which will then update itself.

```
model.nodeStructureChanged(root);
```

Let's add a few more nodes to *node2* so that the example is more interesting.

```
DefaultGanttChartNode node21 = new DefaultGanttChartNode("Node 21");
DefaultGanttChartNode node22 = new DefaultGanttChartNode("Node 22");
DefaultGanttChartNode node23 = new DefaultGanttChartNode("Node 23");
DefaultGanttChartNode node24 = new DefaultGanttChartNode("Node 24");
DefaultGanttChartNode node25 = new DefaultGanttChartNode("Node 25");
node21.setKey("Key 21");
node22.setKey("Key 22");
```

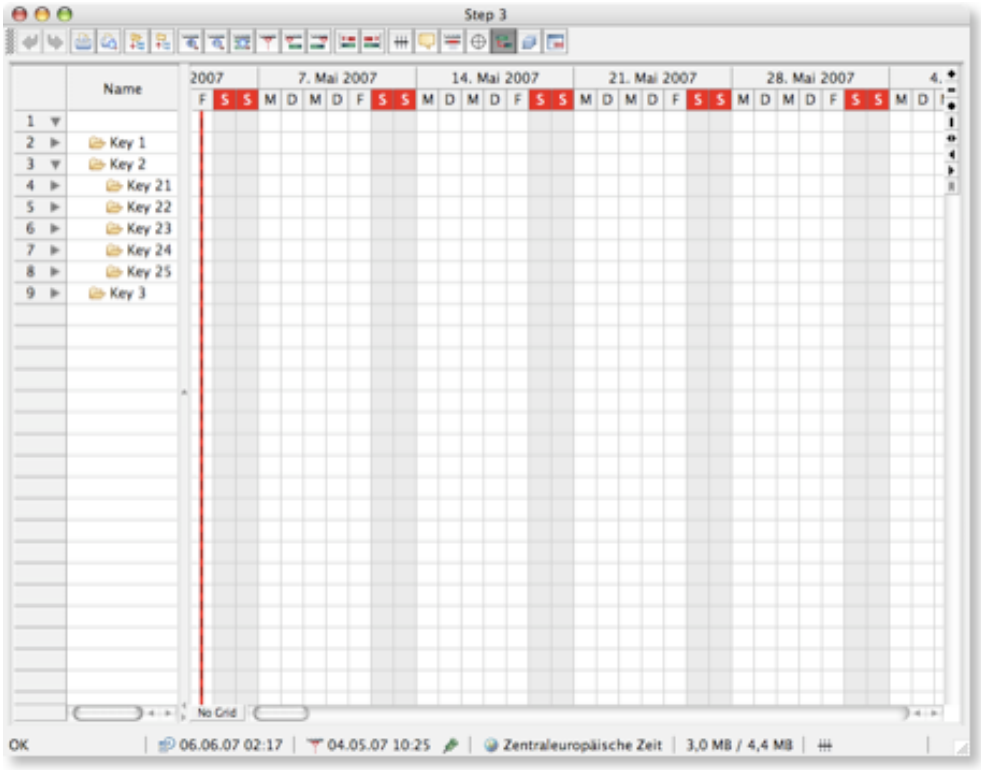
```
node23.setKey("Key 23");
node24.setKey("Key 24");
node25.setKey("Key 25");
node2.add(node21);
node2.add(node22);
node2.add(node23);
node2.add(node24);
node2.add(node25);
model.nodeStructureChanged(node2); // another event
```

Our tutorial application now looks like this:

```
/**
 * Copyright 2006 - 2008
 * Dirk Lemmermann Software & Consulting
 * http://www.dlsc.com
 */
package com.dlsc.flexgantt.examples.jumpstart;

import javax.swing.JFrame;
import com.dlsc.flexgantt.model.gantt.DefaultGanttChartModel;
import com.dlsc.flexgantt.model.gantt.DefaultGanttChartNode;
import com.dlsc.flexgantt.swing.GanttChart;
import com.dlsc.flexgantt.swing.GanttChartFrame;

/**
 * One of the steps used for the 'Jumpstart' tutorial. The step will create a
 * couple of tree nodes and add them to the Gantt chart.
 *
 * @author Dirk Lemmermann
 */
public class Step2_Tree_Nodes {

    /**
     * The application's entry point.
     */
    public static void main(String[] args) {

        /*
         * Create a basic Gantt chart, which will use the default Gantt chart
         * model. The default model itself will use the default Gantt chart node
         * as a root object.
         */
        GanttChart gc = new GanttChart();

        /*
         * Retrieve the default model and node from the Gantt chart.
         */
        DefaultGanttChartModel model = (DefaultGanttChartModel) gc.getModel();
        DefaultGanttChartNode root = (DefaultGanttChartNode) model.getRoot();

        /*
         * Add new nodes to the model and trigger an event so that the UI will
         * update its state.
         */
        DefaultGanttChartNode node1 = new DefaultGanttChartNode("Node 1");
        DefaultGanttChartNode node2 = new DefaultGanttChartNode("Node 2");
        DefaultGanttChartNode node3 = new DefaultGanttChartNode("Node 3");
        node1.setKey("Key 1");
        node2.setKey("Key 2");
        node3.setKey("Key 3");
        root.add(node1);
        root.add(node2);
        root.add(node3);
        model.nodeStructureChanged(root); // the event

        /*
         * Add more nodes to node2.
         */
        DefaultGanttChartNode node21 = new DefaultGanttChartNode("Node 21");
        DefaultGanttChartNode node22 = new DefaultGanttChartNode("Node 22");
        DefaultGanttChartNode node23 = new DefaultGanttChartNode("Node 23");
        DefaultGanttChartNode node24 = new DefaultGanttChartNode("Node 24");
        DefaultGanttChartNode node25 = new DefaultGanttChartNode("Node 25");
        node21.setKey("Key 21");
        node22.setKey("Key 22");
        node23.setKey("Key 23");
        node24.setKey("Key 24");
        node25.setKey("Key 25");
        node2.add(node21);
```

```
        node2.add(node22);
        node2.add(node23);
        node2.add(node24);
        node2.add(node25);
        model.nodeStructureChanged(node2); // another event

        /*
         * Add the Gantt chart to a specialized frame class. The frame will
         * automatically add a status bar and a glass pane (used for updating
         * the cursor when commands get executed).
         */
        GanttChartFrame<GanttChart> frame = new GanttChartFrame<GanttChart>(
                "Step 2", gc);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /*
         * Show the frame. The split panel inside the Gantt chart will adjust
         * to the preferred size of the left-hand side (the tree table).
         */
        frame.setVisible(true);
    }
}
```

Tree Path



Tutorial Application after Tree Path

Tree paths are used by the tree table to locate certain nodes within the tree hierarchy. In this step we are using a tree path to open *node2*, which we created in step 2. First we create the tree path like this:

```
TreePath path = new TreePath(new Object[] { root, node2 });
```

A path always starts with the root node. The next element has to be one of its children, the next element has to be a child of child, and so on ....

To open the specified node simply call the appropriate method on *TreeTable*:

```
TreeTable table = gc.getTreeTable();
table.expandPath(path);
```

Now that node 2 has been opened we want to make sure that all labels in the key column of the tree table are visible. For this we have to optimize the key column's width:

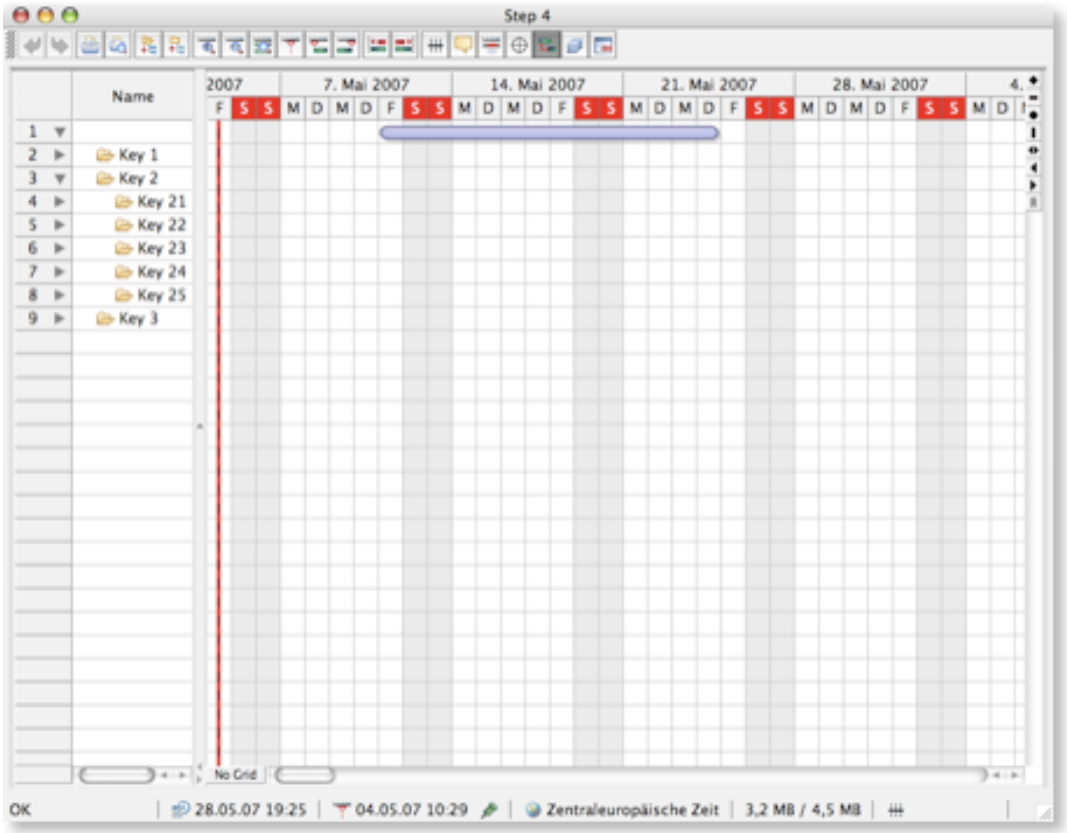
```
optimizeColumnWidth(-1);
```

Adding these lines results in the following application:

```
/**
 * Copyright 2006 - 2008
 * Dirk Lemmermann Software & Consulting
 * http://www.dlsc.com
 */
package com.dlsc.flexgantt.examples.jumpstart;

import javax.swing.JFrame;
import javax.swing.tree.TreePath;
import com.dlsc.flexgantt.model.gantt.DefaultGanttChartModel;
import com.dlsc.flexgantt.model.gantt.DefaultGanttChartNode;
import com.dlsc.flexgantt.swing.GanttChart;
import com.dlsc.flexgantt.swing.GanttChartFrame;
import com.dlsc.flexgantt.swing.treetable.TreeTable;

/**
 * One of the steps used for the 'Jumpstart' tutorial. The step will create a
 * tree path object and use it to open up a tree node.
 *
 * @author Dirk Lemmermann
 */
public class Step3_Tree_Path {

    /**
     * The application's main entry point.
     */
    public static void main(String[] args) {

        /*
         * Create a basic Gantt chart, which will use the default Gantt chart
         * model. The default model itself will use the default Gantt chart node
         * as a root object.
         */
        GanttChart gc = new GanttChart();

        /*
         * Retrieve the default model and node from the Gantt chart.
         */
        DefaultGanttChartModel model = (DefaultGanttChartModel) gc.getModel();
        DefaultGanttChartNode root = (DefaultGanttChartNode) model.getRoot();

        /*
         * Add new nodes to the model and trigger an event so that the UI will
         * update its state.
         */
        DefaultGanttChartNode node1 = new DefaultGanttChartNode("Node 1");
        DefaultGanttChartNode node2 = new DefaultGanttChartNode("Node 2");
        DefaultGanttChartNode node3 = new DefaultGanttChartNode("Node 3");
        node1.setKey("Key 1");
        node2.setKey("Key 2");
        node3.setKey("Key 3");
        root.add(node1);
        root.add(node2);
        root.add(node3);
        model.nodeStructureChanged(root); // the event

        /*
         * Add more nodes to node2.
         */
        DefaultGanttChartNode node21 = new DefaultGanttChartNode("Node 21");
        DefaultGanttChartNode node22 = new DefaultGanttChartNode("Node 22");
        DefaultGanttChartNode node23 = new DefaultGanttChartNode("Node 23");
        DefaultGanttChartNode node24 = new DefaultGanttChartNode("Node 24");
        DefaultGanttChartNode node25 = new DefaultGanttChartNode("Node 25");
        node21.setKey("Key 21");
        node22.setKey("Key 22");
        node23.setKey("Key 23");
        node24.setKey("Key 24");
        node25.setKey("Key 25");
        node2.add(node21);
        node2.add(node22);
        node2.add(node23);
        node2.add(node24);
        node2.add(node25);
        model.nodeStructureChanged(node2); // another event

        /*
         * Make sure that the children of node2 are also initially visible.
```

```
         */
        TreePath path = new TreePath(new Object[] { root, node2 });
        TreeTable table = gc.getTreeTable();
        table.expandPath(path);
        gc.optimizeColumnWidth(-1);

        /*
         * Add the Gantt chart to a specialized frame class. The frame will
         * automatically add a status bar and a glass pane (used for updating
         * the cursor when commands get executed).
         */
        GanttChartFrame<GanttChart> frame = new GanttChartFrame<GanttChart>(
                "Step 3", gc);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /*
         * Show the frame. The split panel inside the Gantt chart will adjust to
         * the preferred size of the left-hand side (the tree table).
         */
        frame.setVisible(true);
        gc.resetToPreferredSizes();
    }
}
```

Timeline Objects



Tutorial Application after Timeline Objects

Now it is time to work on the right-hand side of the Gantt chart and add a timeline object to the root node. Each timeline object has to specify a time span with a start and end time time. We want to make the time span relative to the Gantt chart's planning horizon, hence we first query the horizon and then calculate a start and end time based on the start time of the horizon. The horizon can be queried directly from the Gantt chart like this:

```
TimeSpan span = (TimeSpan) gc.getTimeSpan();
```

For the timeline object's start time we use the horizon's start time and add 7 days. For the end time we add 21 days to the horizon's end time, resulting in a time span with a duration of 14 days.

```
Calendar start = span.getStartCalendar();
start.add(Calendar.DAY_OF_YEAR, 7); // one week after model starts
Calendar end = span.getStartCalendar();
end.add(Calendar.DAY_OF_YEAR, 21); // fourteen days later
```

Now we create the timeline object, which can carry a user object of type `String`.

```
DefaultTimelineObject<String> tlo =
    new DefaultTimelineObject<String>("Root TLO", new TimeSpan(start, end));
```

Timeline objects can not be added directly to the Gantt chart model. They are always added to a layer that has to be a member of the model. Our model doesn't have any layers, yet, so we create one called "My Layer" and add it to the model first before we add the timeline object.

```
ILayer layer = new Layer("My Layer");
model.addLayer(layer);
```

Now we can add the timeline object to the root node.

```
root.addTimelineObject(layer, tlo);
```

With the above lines of code added we now have the following application:

```
/**
 * Copyright 2006 - 2008
 * Dirk Lemmermann Software & Consulting
 * http://www.dlsc.com
 */
package com.dlsc.flexgantt.examples.jumpstart;

import java.util.Calendar;
import javax.swing.JFrame;
import javax.swing.tree.TreePath;
import com.dlsc.flexgantt.model.TimeSpan;
import com.dlsc.flexgantt.model.gantt.DefaultGanttChartModel;
import com.dlsc.flexgantt.model.gantt.DefaultGanttChartNode;
import com.dlsc.flexgantt.model.gantt.DefaultTimelineObject;
import com.dlsc.flexgantt.model.gantt.ILayer;
import com.dlsc.flexgantt.model.gantt.Layer;
import com.dlsc.flexgantt.swing.GanttChart;
import com.dlsc.flexgantt.swing.GanttChartFrame;
import com.dlsc.flexgantt.swing.treetable.TreeTable;

/**
* One of the steps used for the 'Jumpstart' tutorial. The step will create a
* timeline object and add it to the root node.
*
* @author Dirk Lemmermann
*/
public class Step4_Timeline_Objects {

    /**
     * The application's main entry point.
     */
    public static void main(String[] args) {

        /*
         * Create a basic Gantt chart, which will use the default Gantt chart
         * model. The default model itself will use the default Gantt chart node
         * as a root object.
         */
        GanttChart gc = new GanttChart();

        /*
         * Retrieve the default model and node from the Gantt chart.
         */
        DefaultGanttChartModel model = (DefaultGanttChartModel) gc.getModel();
        DefaultGanttChartNode root = (DefaultGanttChartNode) model.getRoot();

        /*
         * Create a single timeline object, add it to the root node, and set its
         * time span in such a way that it begins one week after the model
         * starts and ends after 14 days.
         */
        TimeSpan span = (TimeSpan) gc.getTimeSpan();
        Calendar start = span.getStartCalendar();
        start.add(Calendar.DAY_OF_YEAR, 7); // one week after model starts
        Calendar end = span.getStartCalendar();
        end.add(Calendar.DAY_OF_YEAR, 21); // fourteen days later
```

```
        DefaultTimelineObject<String> tlo = new DefaultTimelineObject<String>(
                "Root TLO", new TimeSpan(start, end));
        ILayer layer = new Layer("My Layer");
        model.addLayer(layer);
        root.addTimelineObject(layer, tlo);

        /*
         * Add new nodes to the model and trigger an event so that the UI will
         * update its state.
         */
        DefaultGanttChartNode node1 = new DefaultGanttChartNode("Node 1");
        DefaultGanttChartNode node2 = new DefaultGanttChartNode("Node 2");
        DefaultGanttChartNode node3 = new DefaultGanttChartNode("Node 3");
        node1.setKey("Key 1");
        node2.setKey("Key 2");
        node3.setKey("Key 3");
        root.add(node1);
        root.add(node2);
        root.add(node3);
        model.nodeStructureChanged(root); // the event

        /*
         * Add more nodes to node2.
         */
        DefaultGanttChartNode node21 = new DefaultGanttChartNode("Node 21");
        DefaultGanttChartNode node22 = new DefaultGanttChartNode("Node 22");
        DefaultGanttChartNode node23 = new DefaultGanttChartNode("Node 23");
        DefaultGanttChartNode node24 = new DefaultGanttChartNode("Node 24");
        DefaultGanttChartNode node25 = new DefaultGanttChartNode("Node 25");
        node21.setKey("Key 21");
        node22.setKey("Key 22");
        node23.setKey("Key 23");
        node24.setKey("Key 24");
        node25.setKey("Key 25");
        node2.add(node21);
        node2.add(node22);
        node2.add(node23);
        node2.add(node24);
        node2.add(node25);
        model.nodeStructureChanged(node2); // another event

        /*
         * Make sure that the children of node2 are also initially visible.
         */
        TreePath path = new TreePath(new Object[] { root, node2 });
        TreeTable table = gc.getTreeTable();
        table.expandPath(path);
        gc.optimizeColumnWidth(-1);

        /*
         * Add the Gantt chart to a specialized frame class. The frame will
         * automatically add a status bar and a glass pane (used for updating
         * the cursor when commands get executed).
         */
        GanttChartFrame<GanttChart> frame = new GanttChartFrame<GanttChart>(
                "Step 4", gc);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        /*
         * Show the frame. The split panel inside the Gantt chart will adjust to
         * the preferred size of the left-hand side (the tree table).
         */
        frame.setVisible(true);
        gc.resetToPreferredSizes();
    }
}
```