Introduction to FlexGantt

Topic: Layers

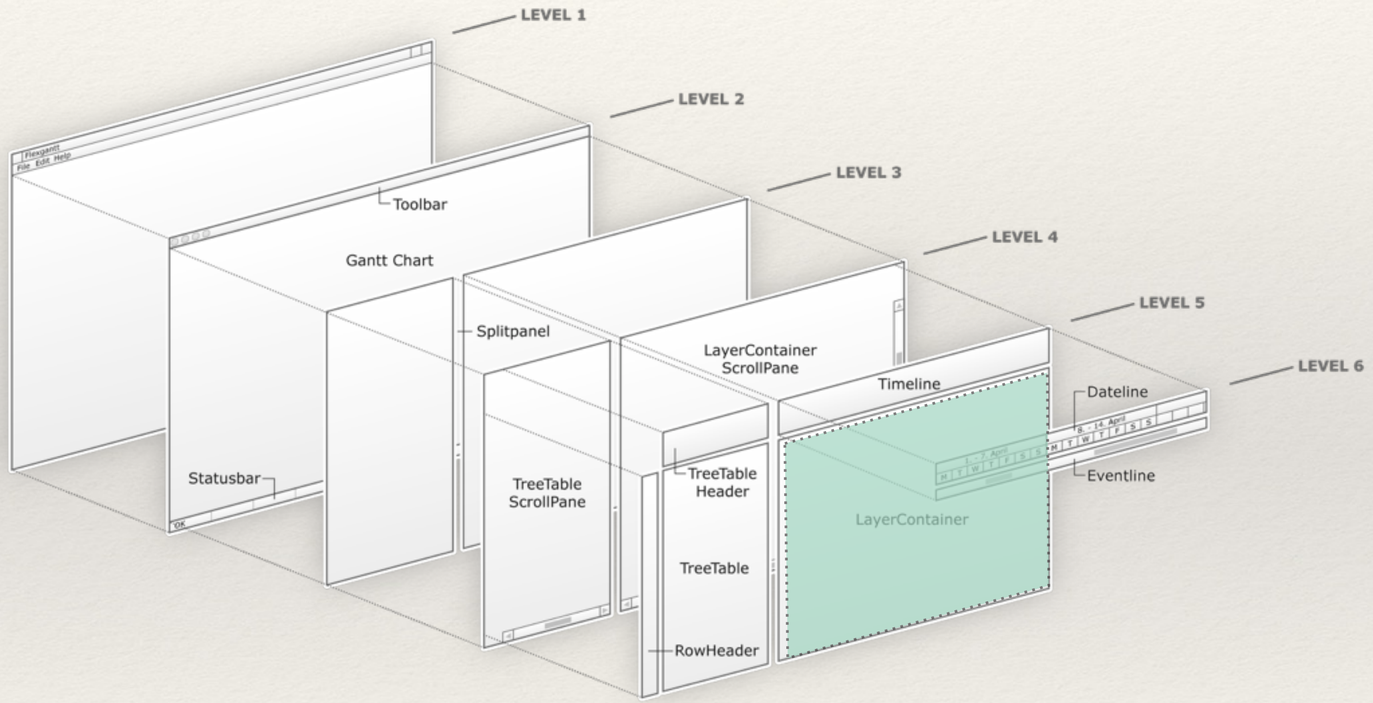
Dirk Lemmermann
Software & Consulting
Zurich, Switzerland

Content

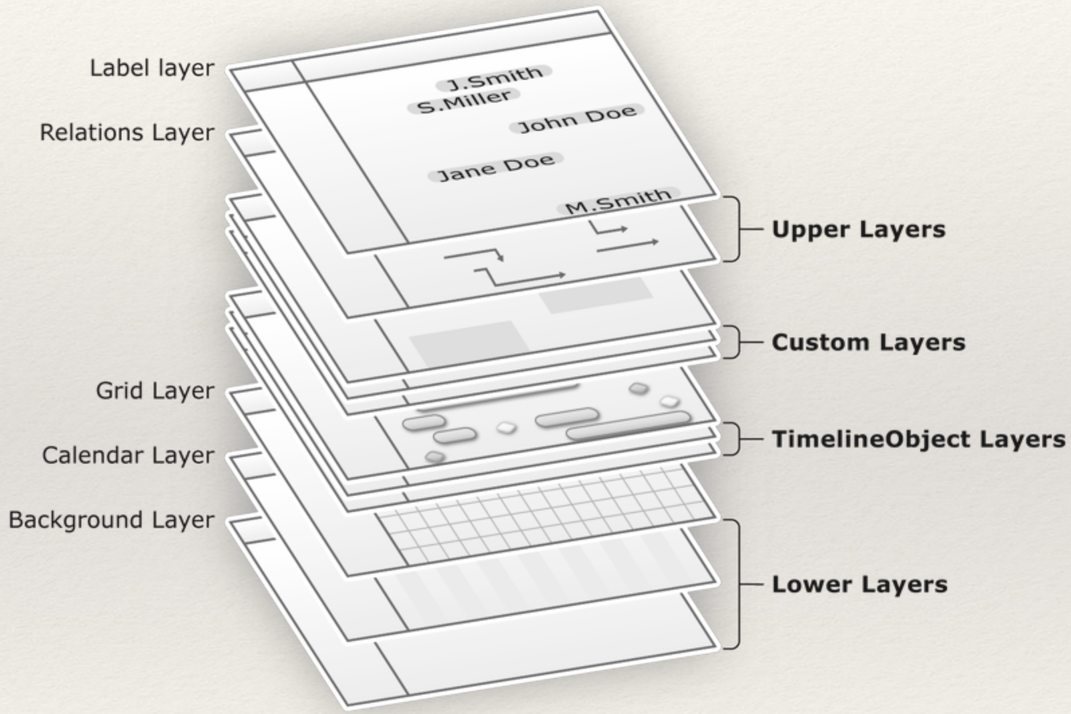
- Layer Container
- Layer Factory
- Timeline Object Layers
- System Layers
- Custom Layers

COMPONENTS

Architecture



Layer Container



Layer Model Object

```
/**
 * An interface for objects that represent a layer.
 */
public interface ILayer extends INamedObject {

	/**
	 * An enumerator with values that describe features that a layer can have or
	 * not have.
	 */
	public enum Feature {

		TIMELINE_OBJECT_DESCRIPTIONS,

		RELATIONSHIPS,

		DELETION,

		TRANSPARENCY,

		OVERVIEW,

		TIMELINE_OBJECT_CREATION,

		SHOW_IN_PALETTE
	}

	/**
	 * Returns TRUE if the layer will be visualized by a custom layer (see
	 */
	boolean isCustomLayer();

	/**
	 * Determines whether a feature is currently supported / required by a layer
	 * or not.
	 */
	boolean isFeatureEnabled(Feature feature);
```

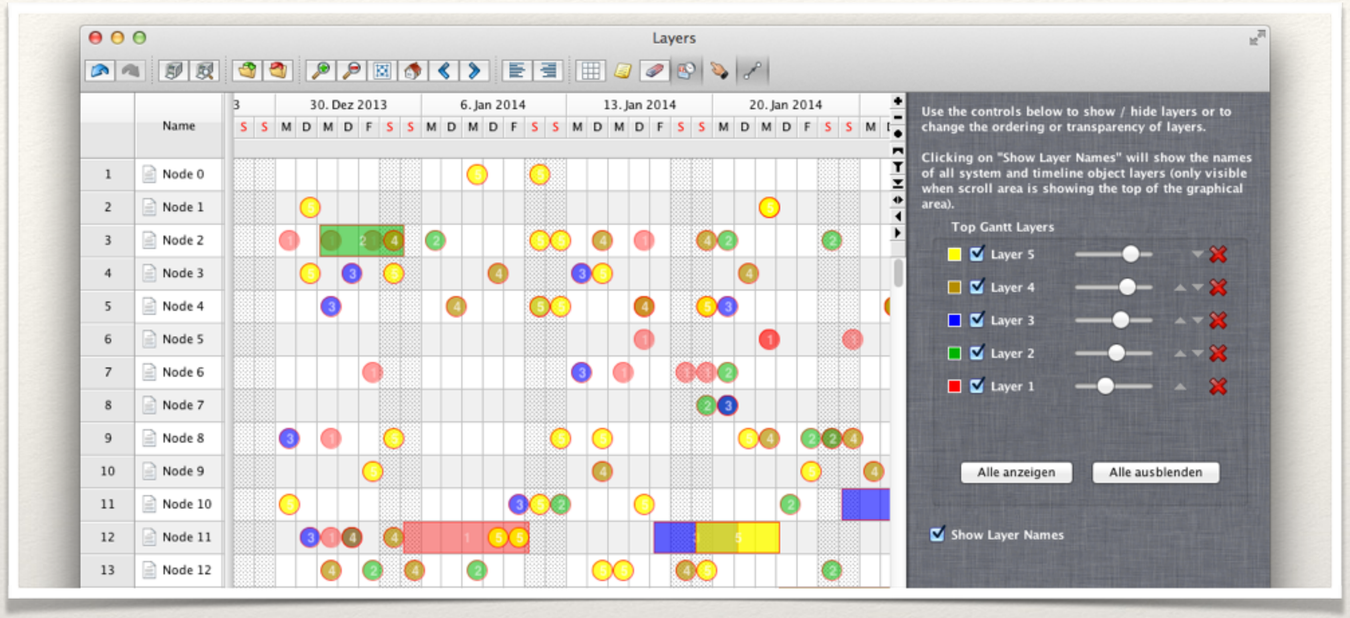
Layer Factory

```
public interface ILayerFactory {

    <T extends AbstractSystemLayer> T createSystemLayer(LayerContainer lc,
            Class<T> layerType);

    TimelineObjectLayer createTimelineLayer(LayerContainer lc, ILayer layer);

    AbstractCustomLayer createCustomLayer(LayerContainer lc, ILayer layer);
}
```

Custom TimeNowLayer

```
public <T extends AbstractSystemLayer> T createSystemLayer(
        LayerContainer lc, Class<T> layerType) {
    if (layerType == TimeNowLayer.class) {
        return (T) new MyTimeNowLayer(lc);
    }
    return super.createSystemLayer(lc, layerType);
}

class MyTimeNowLayer extends TimeNowLayer {

    public MyTimeNowLayer(LayerContainer lc) {
        super(lc);
    }

    @Override
    protected void paintTimeNow(Graphics g, int x) {
        Color c = new Color(255, 100, 100);
        g.setColor(c);
        g.drawLine(x - 1, 0, x - 1, getHeight());
        g.setColor(c.darker());
        g.drawLine(x, 0, x, getHeight());
        g.setColor(c);
        g.drawLine(x + 1, 0, x + 1, getHeight());
    }
}
```

Timeline Object Layers



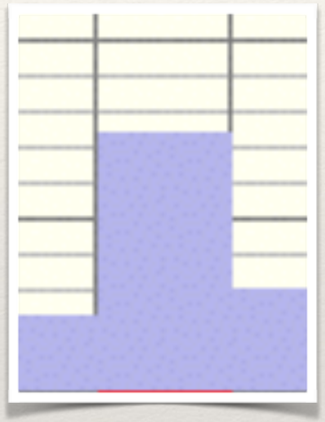
Timeline Object Layer

Renders timeline objects / bars / activities in rows with optional inner lines.

Timeline Object Renderer

DefaultActivityObjectRenderer

DefaultTimelineObjectRenderer

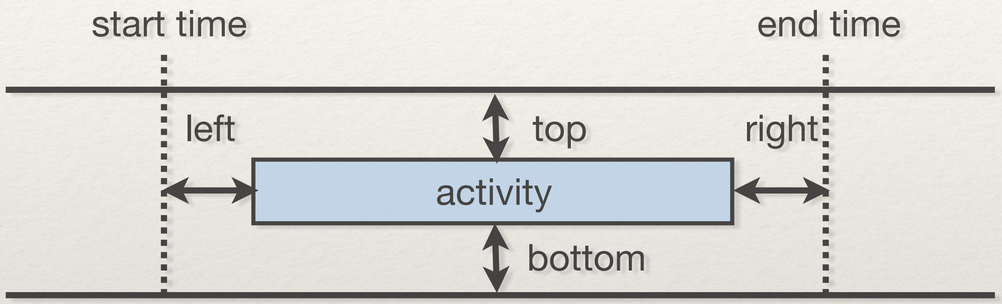


DefaultCapacityObjectRenderer

Creating Timeline Object Renderers

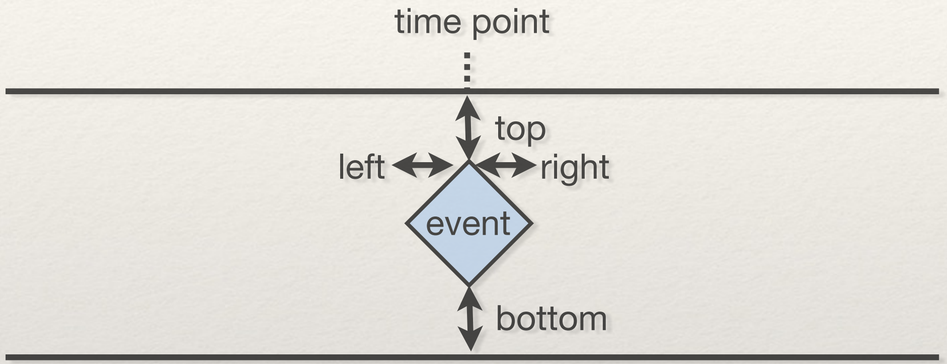
```
public interface ITimelineObjectRenderer extends IRenderer {

   Component getTimelineObjectRendererComponent(
        TimelineObjectLayer layer,
        TimelineObjectPath path,
        boolean selected,
        boolean focus,
        boolean highlighted,
        int row);

   Insets getTimelineObjectRendererInsets(int x, int y, int width,
int height);

}
```

Timeline Object Insets (Activity)



Example: insets = (top, left, bottom, right) = (5, 5, 5, 5)

Timeline Object Insets (Event)



Example: insets = (top, left, bottom, right) = (5, -5, 5, -5)

Registering Timeline Object Renderers

```
  public void
LayerContainer.setTimelineObjectRenderer(Class
objectType,
      ITimelineObjectRenderer renderer) {
    if (objectType == null) {
      throw new IllegalArgumentException(
          "the timeline object type can not be NULL“);
    }
    rendererCache.clear();
    rendererMap.put(objectType, renderer);
  }
```

Selection Model

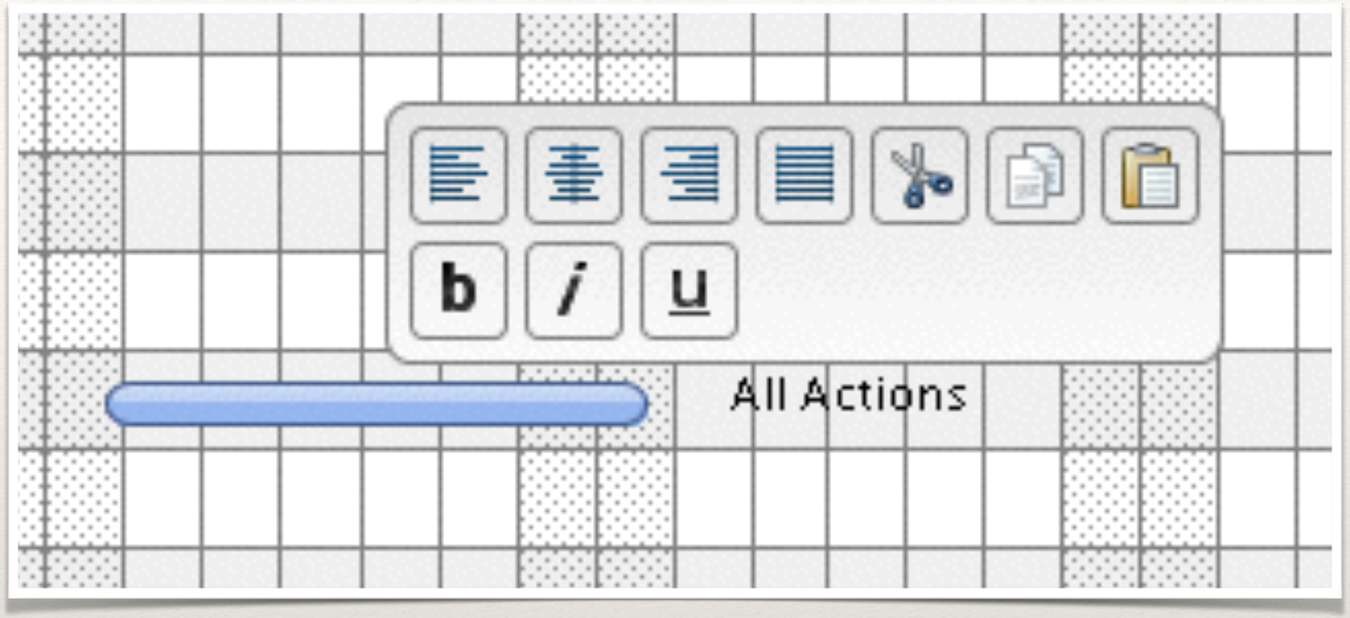
- Each timeline object layer has its own selection model.
- Model stores timeline object paths.

```
    public ITimelineObjectSelectionModel
                LayerContainer.getSelectionModel(ILayer
layer)
```

System Layers

System Layers

- Action Layer
- Background Layer
- Calendar Layer
- Crosshair Layer
- Editing Layer
- Dateline Layer
- Drag Layer
- Drop Shadow Layer
- Eventline Layer
- Grid Layer
- Label Layer
- Lasso Layer
- Link Layer
- Popup Layer
- Relationship Layer
- Row Layer
- Selection Layer
- Spreadsheet Layer
- Time Now Layer



System Layer

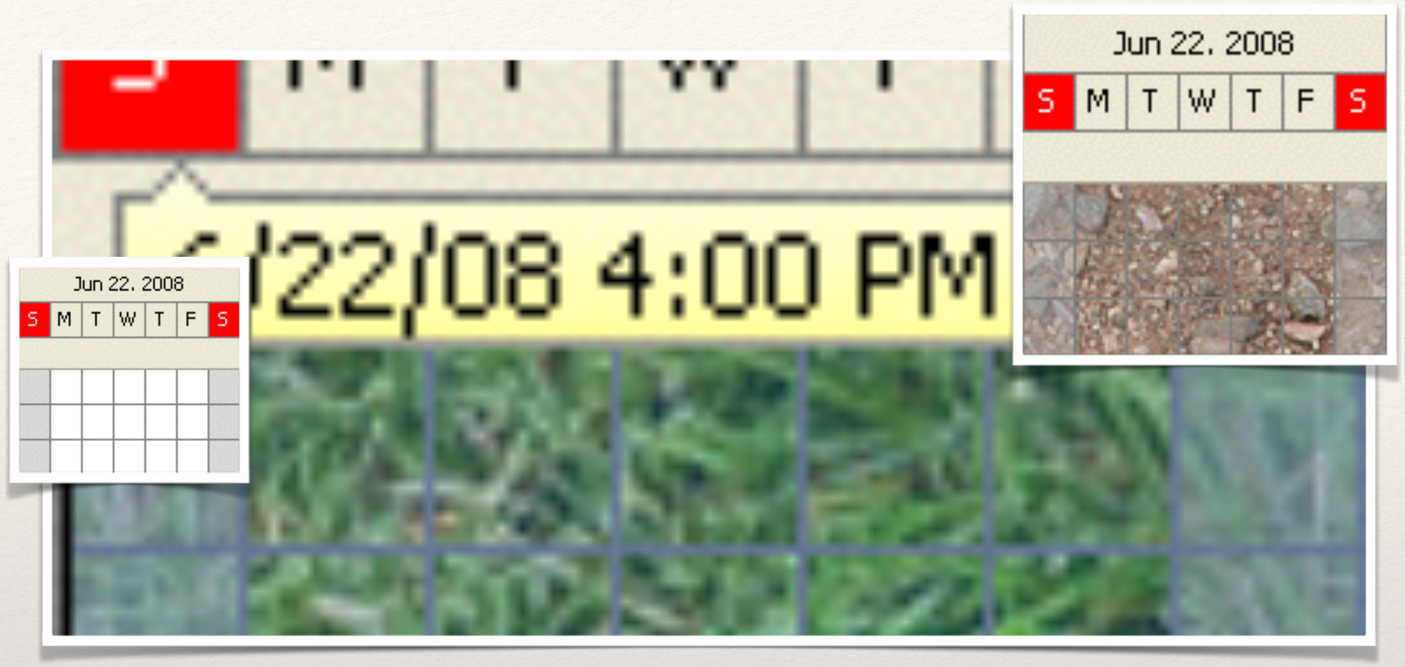
ActionLayer

Shows actions that can be executed on a timeline object.

Action Provider

```
/**
 * An action provider determines which quick actions are available for a given
 * timeline object. These actions can then be made accessible via buttons
 * displayed by the {@link ActionLayer}.
 */
public interface IActionProvider {

    /**
     * Returns the actions that are available for the timeline object given by
     * the path.
     */
    List<Action> getTimelineObjectActions(LayerContainer lc,
            TimelineObjectPath path);
}
```

```
/**
 * Sets the action provider that is used by this layer to lookup the
 * available actions for a given timeline object.
 */
public void ActionLayer.setActionProvider(IActionProvider actionProvider) {
    this.actionProvider = actionProvider;
}
```



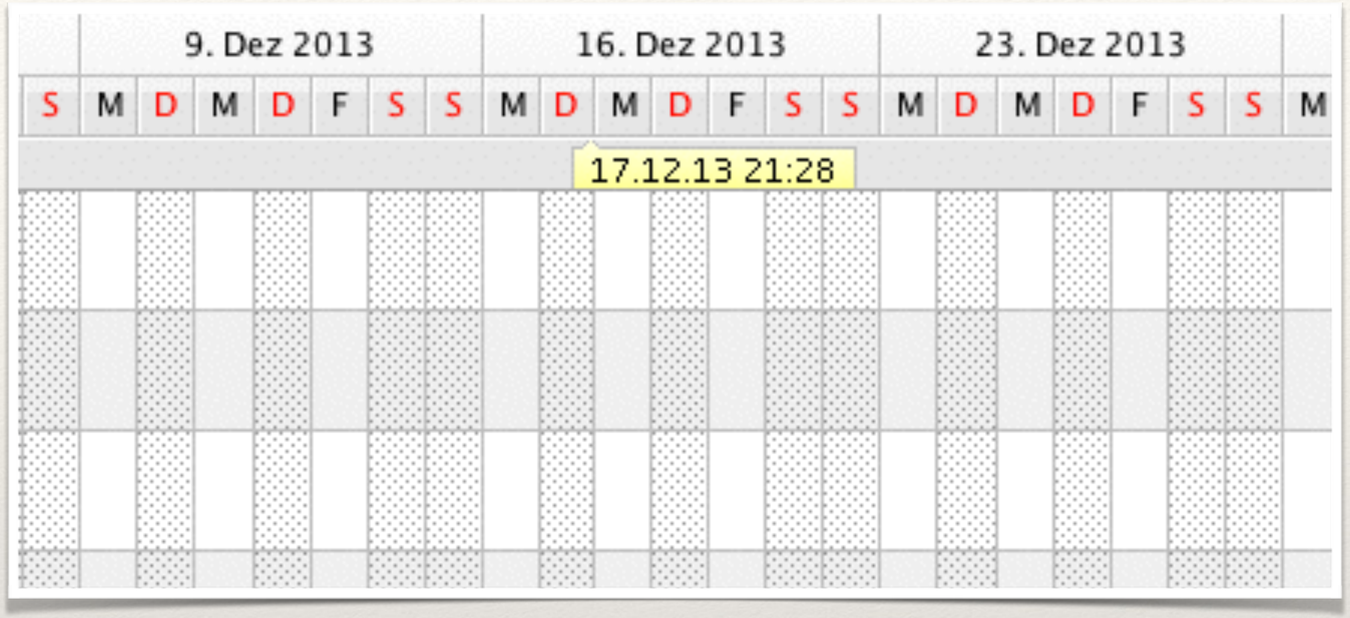
System Layer

BackgroundLayer

Fills the row backgrounds with a primary and a secondary (alternating) row color or with a texture.

Background Layer

```
   /**
    * Sets an image that will be used as a texture to fill the background
of
    * the layer container. The given image will be repeatedly painted
    * horizontally and vertically until the entire background is covered.
    * Textures should make sure that their edges connect seamlessly. A
texture
    * is purely optional. If a texture is not provided the background layer
    * will fill itself with the current background color. Additionally if
an
    * alternating background color is specified the layer will use that
color
    * for every row with an odd row number (assuming row numbers start with
0).
    */
   public void setTexture(Image texture);
```



System Layer

CalendarLayer

Visualizes information provided by a calendar model, used for displaying weekends, holidays, shifts.

Calendar Model

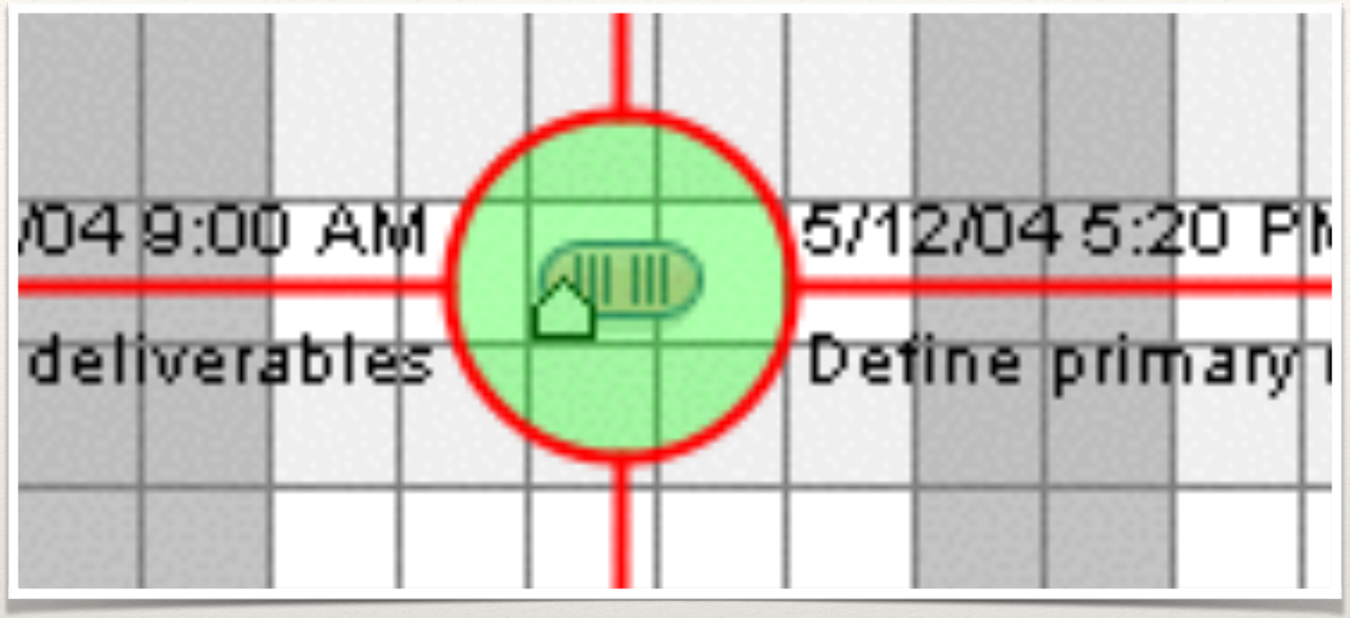
```
/**
 * A calendar model definition that can be used by the Gantt chart to visualize
 * weekends, holidays, or any other special day based on the result of an
 * on-the-fly computation. Objects returned by the model are often not
 * explicitly added to the model first but computed on demand instead.
 * Implementations of this interface have to make sure that they perform well.
 */
public interface ICalendarModel<T, S> {

    /**
     * Returns an iterator for iterating over all calendar entries within the
     * given time span.
     */
    Iterator<S> getCalendarEntries(IDatelineModel model, ITimeSpan span);

    /**
     * Returns an iterator for iterating over all calendar entries of the given
     * node within the given time span.
     */
    Iterator<S> getCalendarEntries(IDatelineModel model, T node, ITimeSpan span);

    ...
}
```

Calendar Entry Renderers

```
/**
 * Maps the implementation of a calendar entry renderer to a calendar object
 * definition.
 *
 * @param objectType
 *            the type of the calendar entries where the visual
 *            representation will be rendered with the given renderer
 * @param renderer
 *            an implementation that will be used to renderer instances of
 *            the given calendar entry type
 */
public void setCalendarEntryRenderer(Class objectType,
        ICalendarEntryRenderer renderer) {

    rendererMap.put(objectType, renderer);

}
```

Examples:

```
    setCalendarEntryRenderer(Weekend.class, new WeekendCalendarEntryRenderer());
    setCalendarEntryRenderer(Holiday.class, new HolidayCalendarEntryRenderer());
    setCalendarEntryRenderer(Maintenance.class, new MaintenanceCalendarEntryRenderer());
```



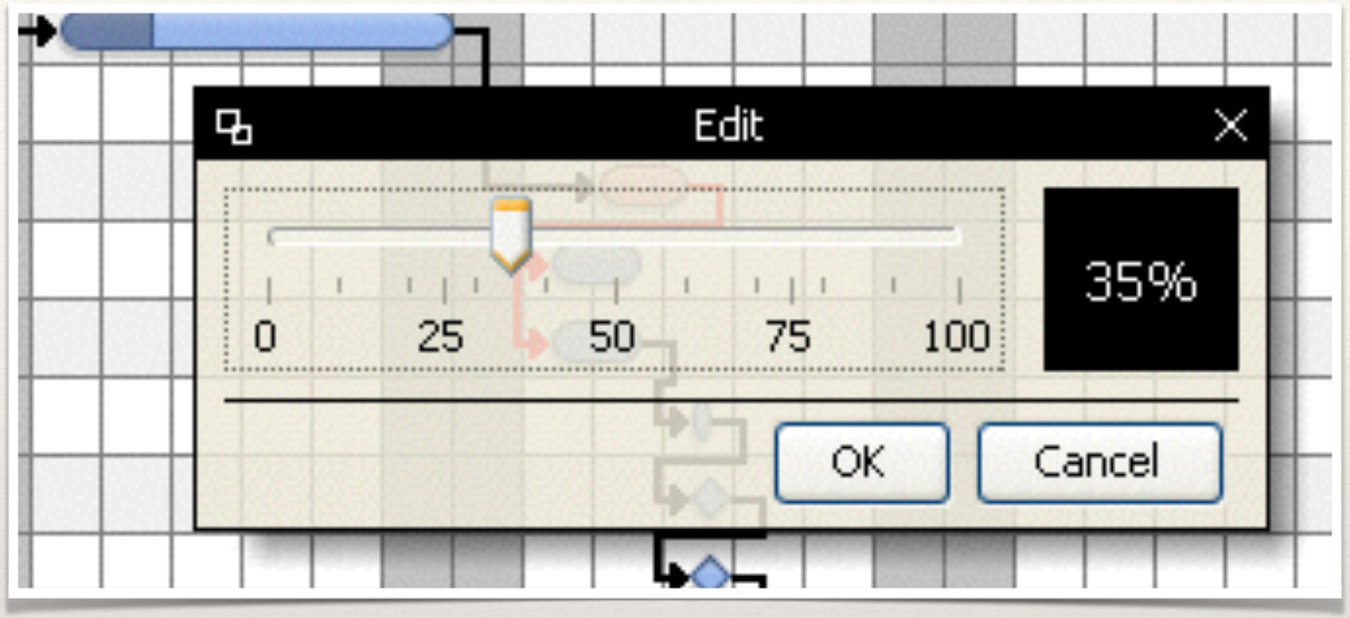
System Layer

CrosshairLayer

Displays a crosshair for presentation purposes.

Crosshair Policy

```
/**
 * A policy used for the crosshair feature. The policy provides labels for the
 * four corners around the crosshair circle.
 */
public interface ICrosshairPolicy extends IPolicy {

    enum LabelPosition {
        UPPER_LEFT,
        UPPER_RIGHT,
        LOWER_LEFT,
        LOWER_RIGHT
    }

    /**
     * Determines whether a label is visible at all.
     */
    boolean isLabelVisible(LabelPosition position);

    /**
     * Returns a label that will be shown to the left/right and top/bottom of
     * the crosshair location.
     */
    String getLabel(Object node, TimelineObjectPath timelineObject,
            IGanttChartModel model, long time, LabelPosition position);
}
```



System Layer

EditingLayer

Handles the display of lightweight editors directly inside the Gantt chart.

Timeline Object Editor

```
/**
 * An interface for editors that can be used for in-place editing of timeline
 * objects. The editor component will not be shown in a dialog but directly
 * inside the Gantt chart.
*/
public interface ITimelineObjectEditor {

    /**
     * Returns the actual editor component for the given tree node and timeline
     * object.
     */
    Component getTimelineObjectEditorComponent(EditingLayer layer,
            TimelineObjectPath path);

    /**
     * Callback method for the timeline layer to indicate to the editor that the
     * user has finished the editing of the timeline object.
     */
    boolean stopEditing();

    /**
     * Callback method for the timeline layer to indicate to the editor that the
     * user has canceled the editing of the timeline object.
     */
    void cancelEditing();

    …
}
```

Editor Registration

```
   /**
    * Maps the implementation of a timeline object editor to a class
    * definition.
    *
    * @param objectType
    *            the type of those timeline objects that will be rendered
with
    *            the given renderer
    * @param editor
    *            an implementation that will be used to edit instances of
the
    *            given object type
    */
   public void LayerContainer.setTimelineObjectEditor(Class objectType,
         ITimelineObjectEditor editor) {
      editorMap.put(objectType, editor);
   }
```

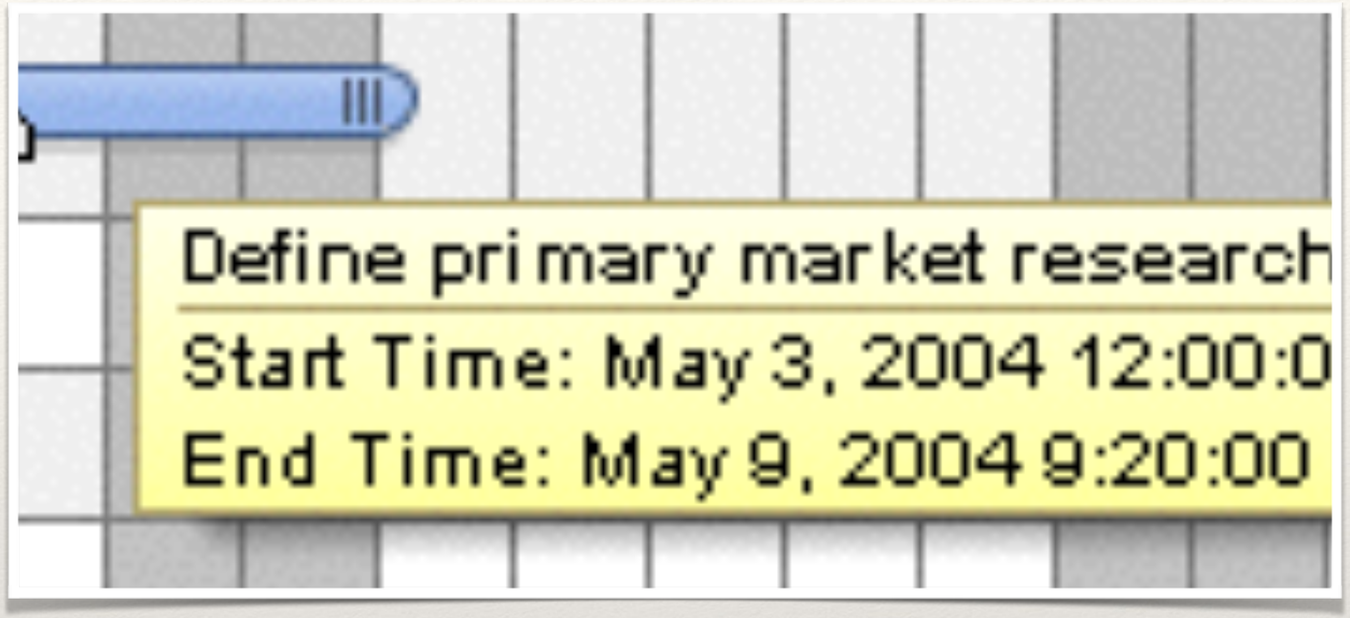
System Layer

DatelineLayer

Visualizes the focused time span and time span selections made by the user in the dateline.

Dateline Layer Options

```
   public void setHorizonLimitsVisible(boolean
visible);



   public void setFocusedTimeVisible(boolean
visible);

   public void setFocusedTimeSpanLinePaint(Paint
color);

   public void setFocusedTimeSpanFillPaint(Paint
fillPaint);
```



System Layer

DragLayer

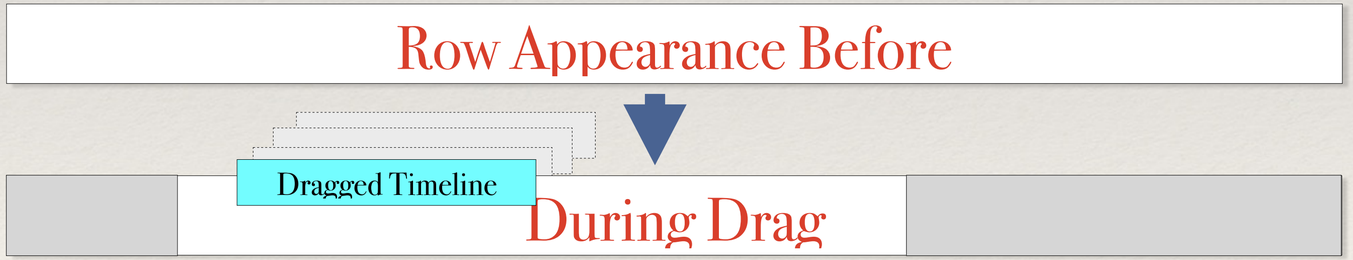
Handles dragging timeline objects. Displays drag popup information. Shows possible drop locations.

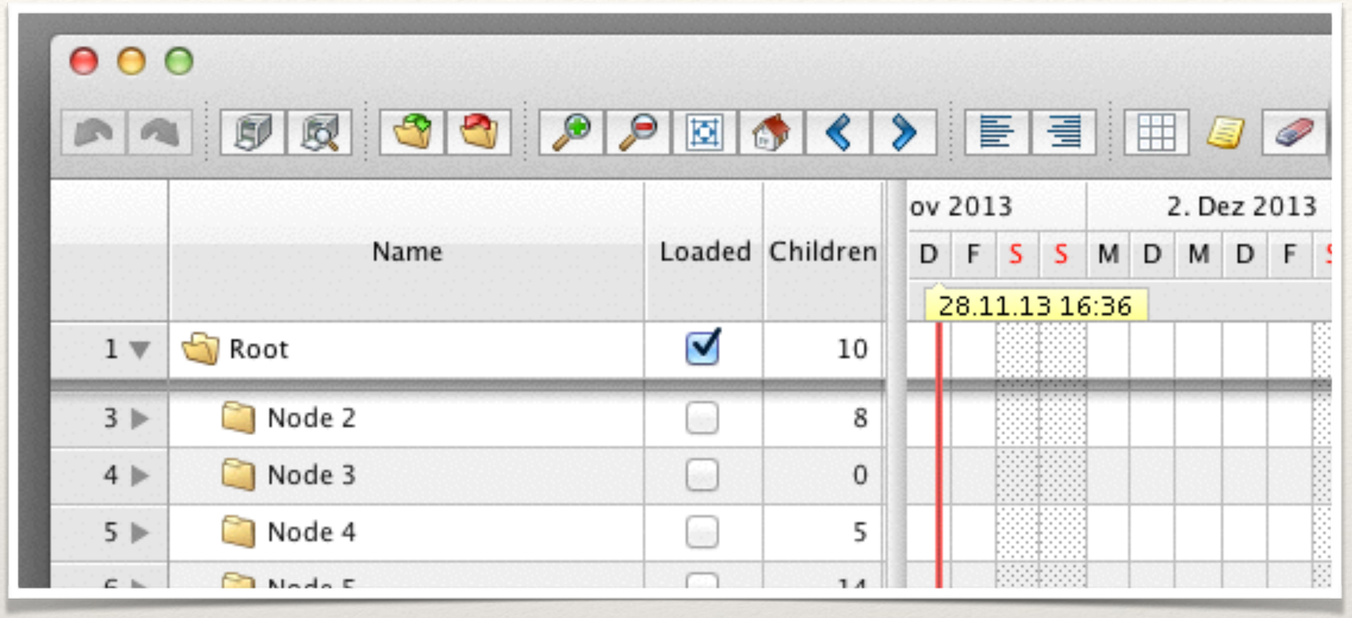
Drag Info

```
   /**
    * Maps the implementation of a drag info renderer to a timeline object
    * type.
    *
    * @param objectType
    *            the type of the timeline object for which the info will
be
    *            rendered with the given renderer
    * @param renderer
    *            a renderer implementation that will be used to renderer
    *            instances of the given timeline object type
    */
   public void setDragInfoRenderer(Class objectType, IDragInfoRenderer
renderer) {
      dragInfoRendererMap.put(objectType, renderer);
   }
```

Drag Row Renderer

```
/**
 * Maps the implementation of a drag row renderer to a tree node type.
 *
 * @param objectType
 *            the type of the tree node for which the row will be rendered
 *            with the given renderer
 * @param renderer
 *            a renderer implementation that will be used to renderer the
 *            row with drag and drop information
 */
public void setDragRowRenderer(Class objectType, IDragRowRenderer renderer) {
    dragRowRendererMap.put(objectType, renderer);
}
```

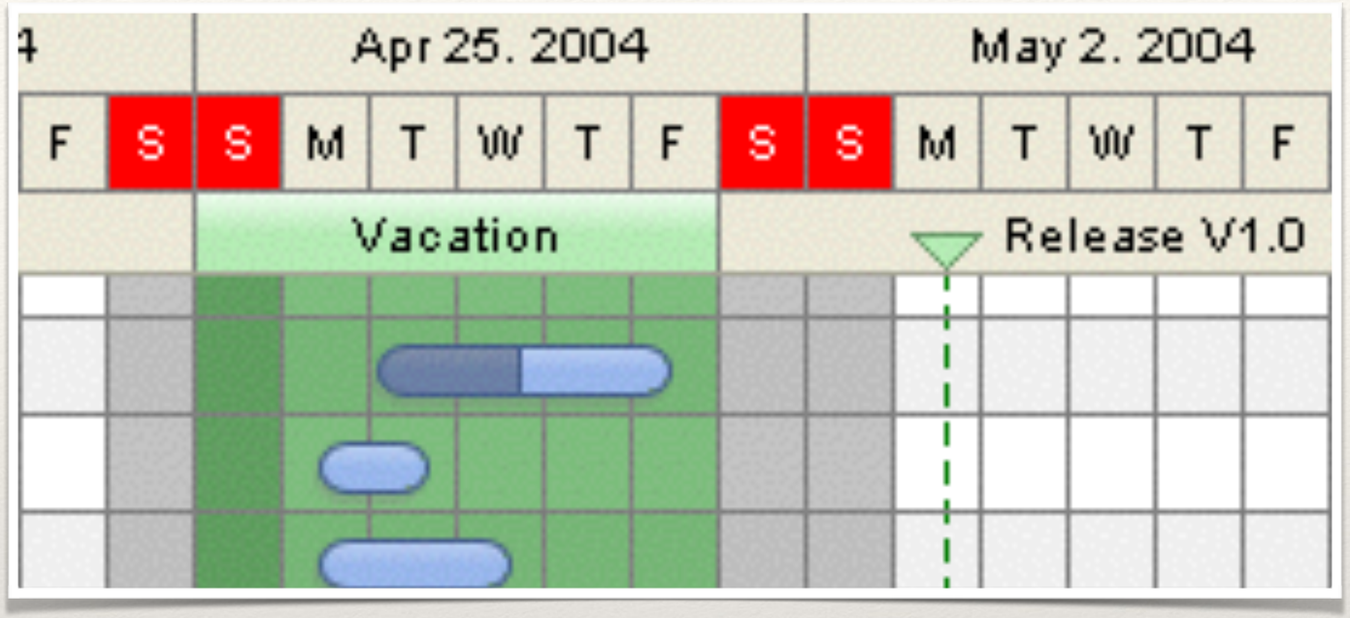




System Layer

DropShadowLayer

Temporarily draws a shadow when tree nodes are expanded / collapsed.



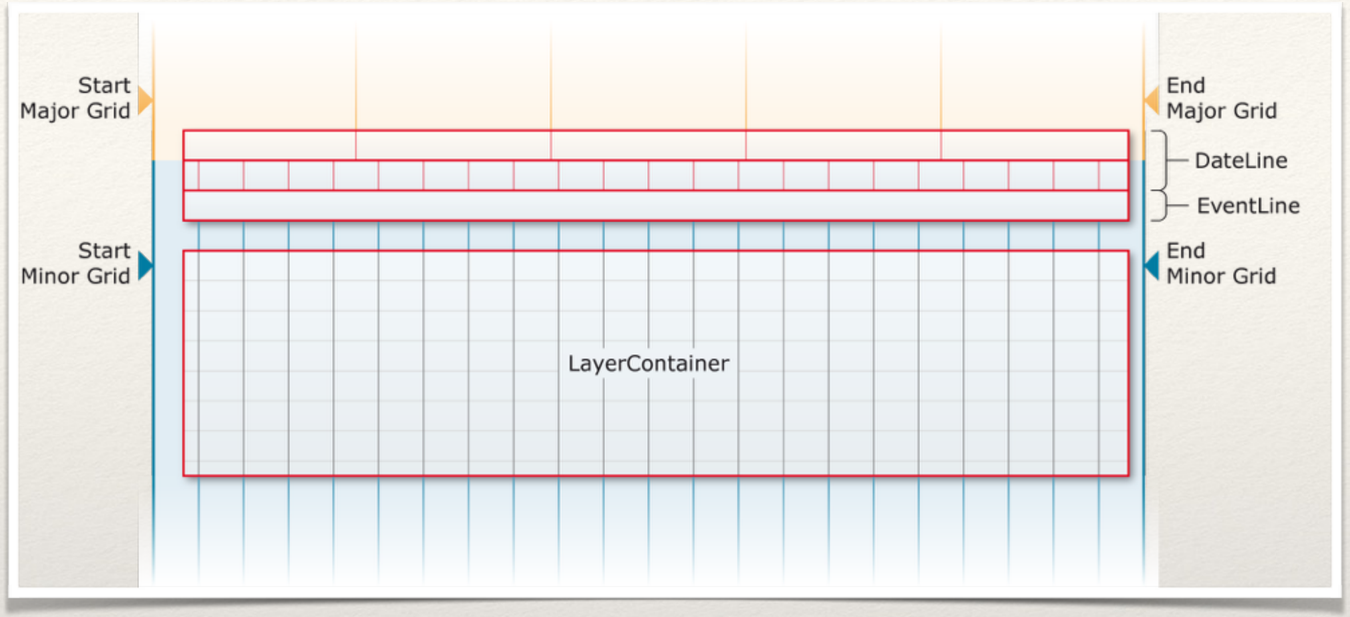
System Layer

EventlineLayer

Renders eventline model information, e.g. company-wide holidays.

Eventline Layer Settings

```
  public void setPaint(Class eventlineObjectType,
Paint paint);

  public void setStroke(Class eventlineObjectType,
Stroke paint);


  public void setPaintingActivities(boolean
paint);

  public void setPaintingEvents(boolean paint);
```



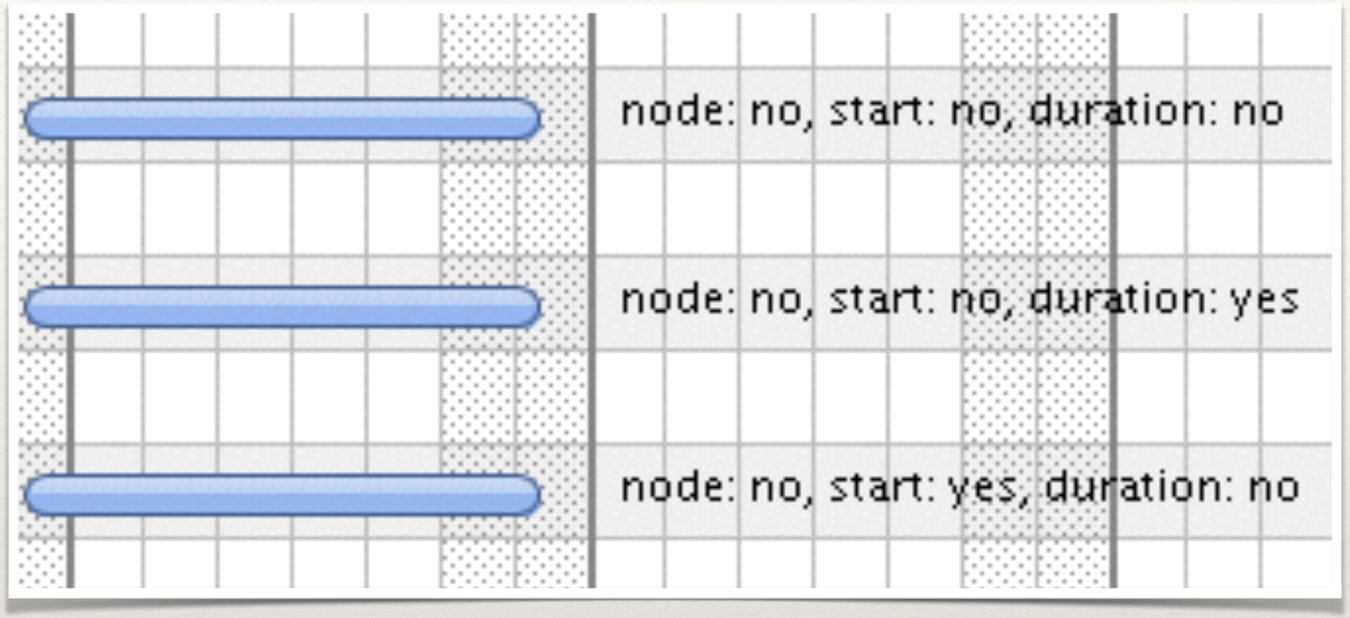
System Layer

GridLayer

Draws major and minor grid lines at the locations that were calculated by the dateline model.

GridLayer Settings

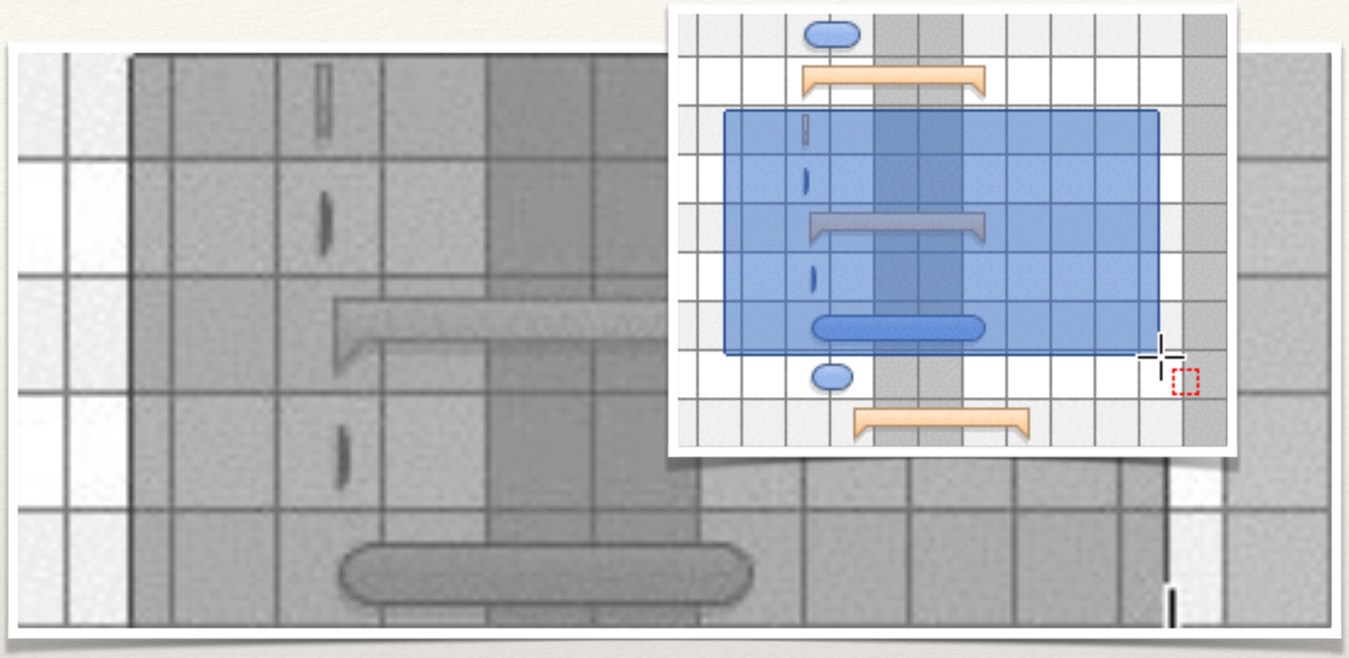
```
  public void setHorizontalLineColor(Color color);

  public void setHorizontalLinesVisible(boolean visible);

  public void setVerticalLinesOnTop(boolean
verticalLinesOnTop);

  public void setMajorGridColor(Color color);

  public void setMinorGridColor(Color color);

  public void setThickMajorGridLines(boolean
thickMajorGridLines);
```



System Layer

LabelLayer

Draws text descriptions behind timeline objects.



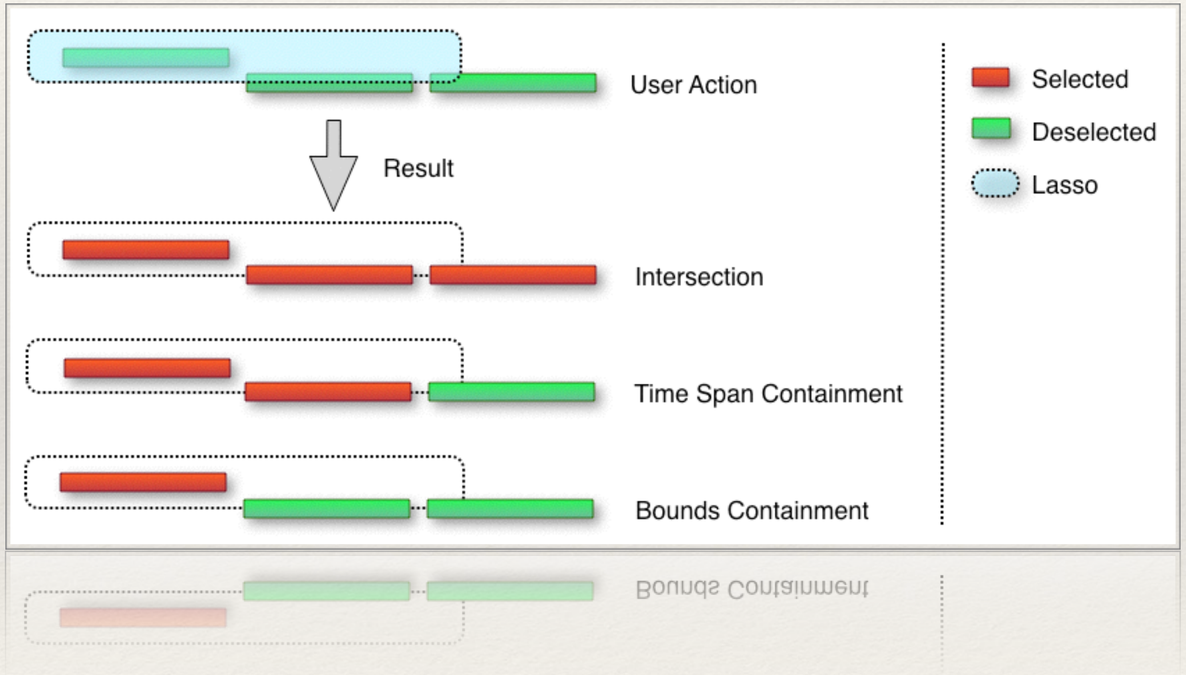
System Layer

LassoLayer

Draws the blue lasso selection box. Supports different selection types and the creation of new timeline objects.

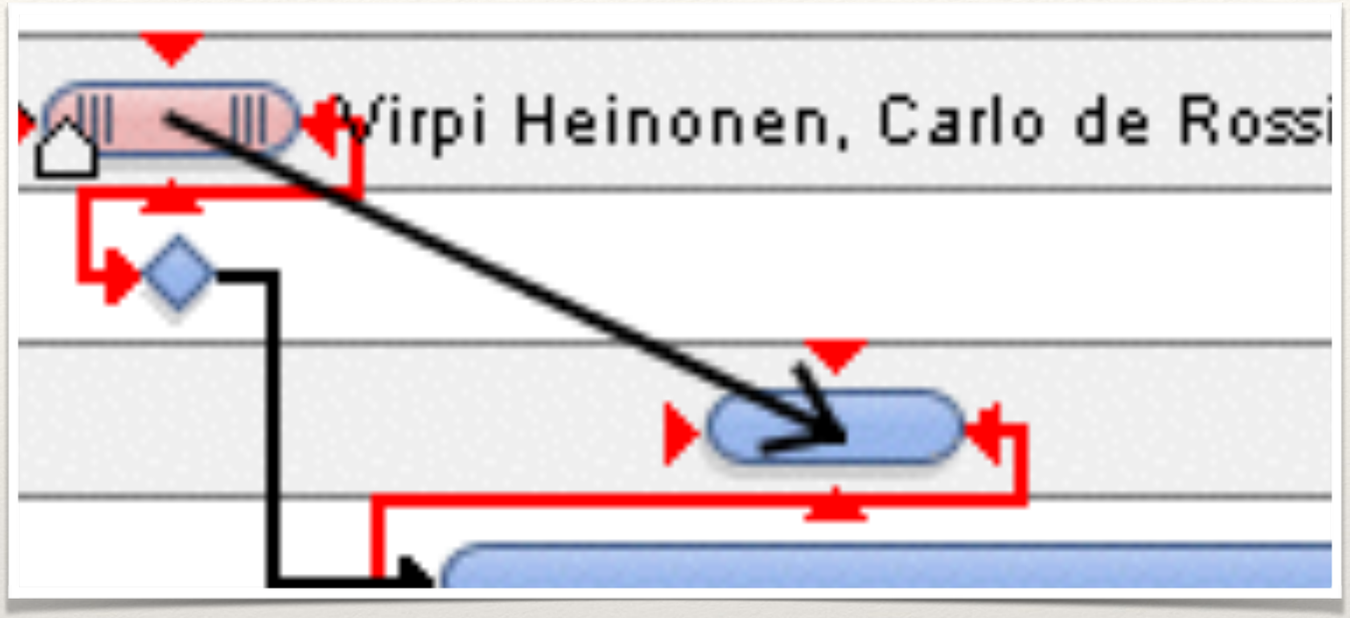
Lasso Layer Behavior

```
public void LassoLayer.setSelectionBehaviour(SelectionBehaviour
behaviour);
```



Lasso Mode

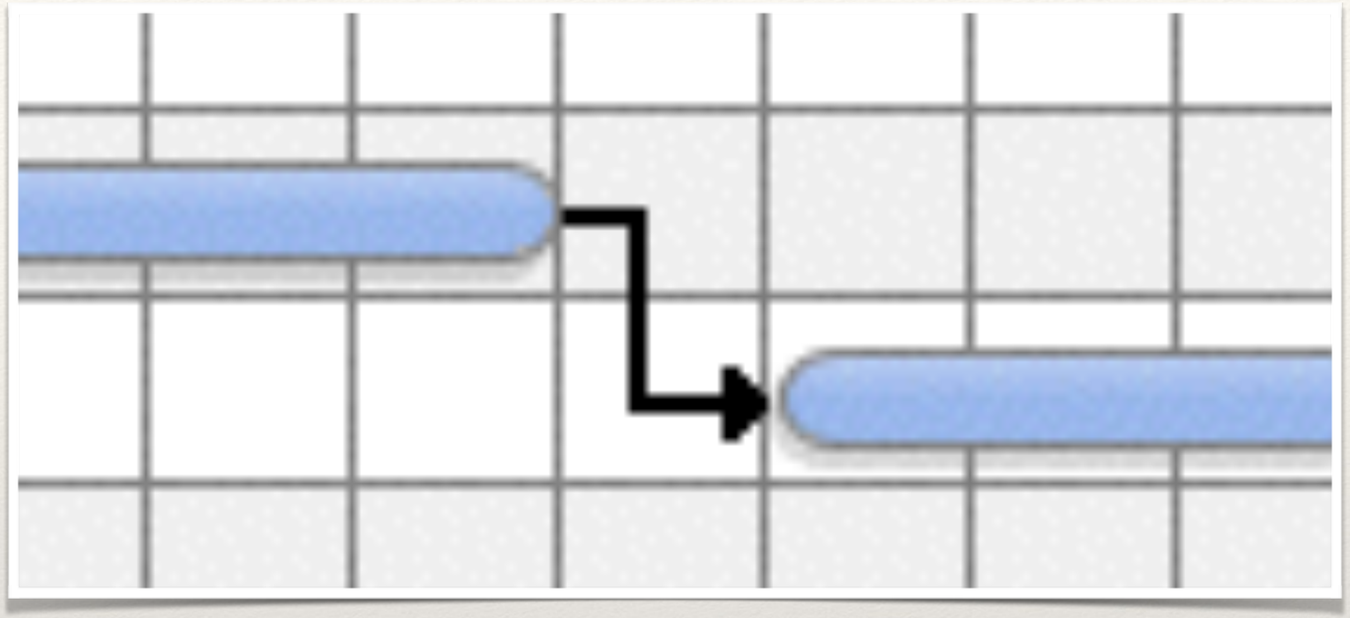
```
public enum LassoMode {

    /**
     * A mode used to select several timeline objects at once. Any timeline
     * object with a time span that intersects with the lasso when the lasso
     * operation finishes will be selected.
     */
    SELECT_TIMELINE_OBJECTS,

    /**
     * A mode used to create timeline objects.
     */
    CREATE_TIMELINE_OBJECTS,

    /**
     * A mode used to select several time spans at once. Selected time spans
     * will be added to the selection model of the layer container.
     */
    SELECT_TIME_SPANS,
}

public void setLassoMode(LassoMode mode);

public void setSupportedLassoModes(LassoMode... modes);
```



System Layer

LinkLayer

Supports the creation of relationships (links / lines) between two timeline objects.



System Layer

RelationshipLayer

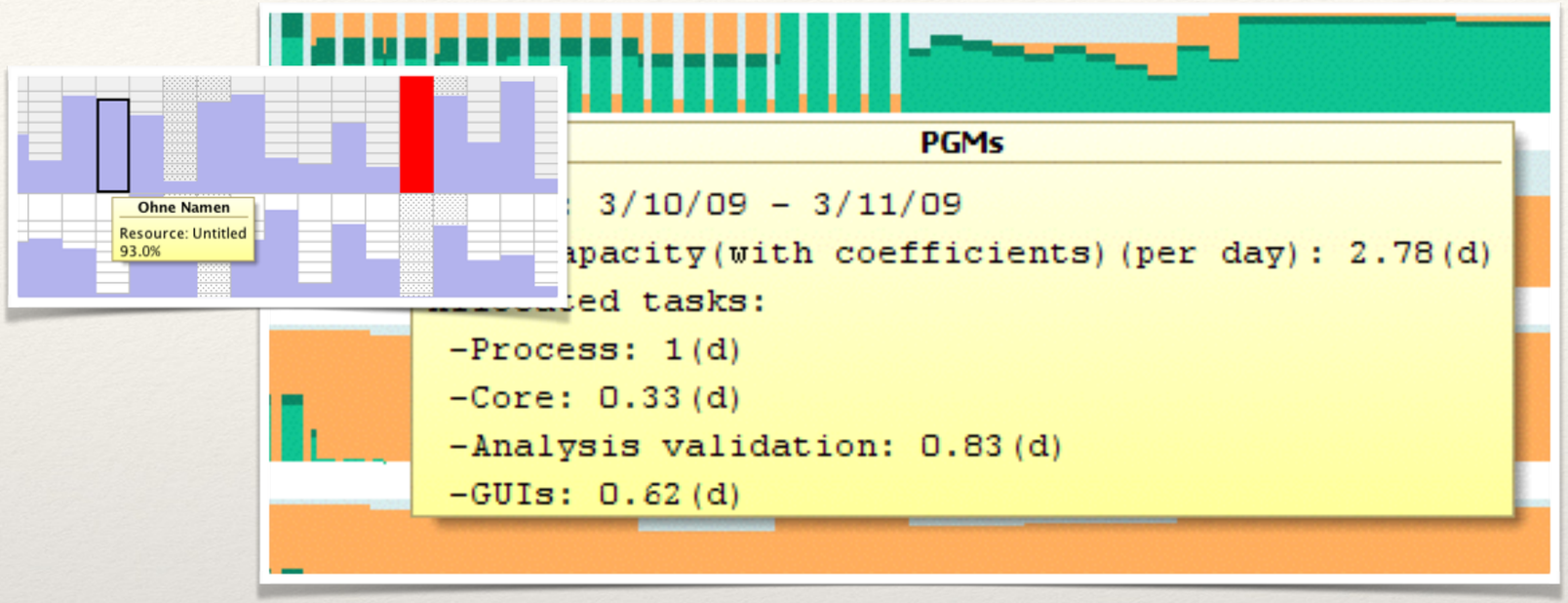
Renders the relationships (links / lines) between timeline objects.

Relationship Renderer

```
public void RelationshipLayer.setRelationshipRenderer(
                      Class<? extends IRelationship> relationshipType,
                      IRelationshipRenderer renderer);


/**
 * A relationship renderer is used to draw a relationship between two timeline
 * objects.
 */
public interface IRelationshipRenderer extends IRenderer {

   /**
    * Draws a relationship between two timeline objects. This is usually a line
    * that starts at the source object and ends at the target object (with an
    * arrow head pointing towards the target).
    */
   GeneralPath drawRelationship(RelationshipLayer layer, Graphics g,
         IRelationship relationship, boolean focused, boolean selected);
}
```

Default Relationship Renderer

```
/**
 * Constructs a new relationship renderer.
 *
 * @param calculator
 *            the calculator used to compute the path to draw
 */
public DefaultRelationshipRenderer(PathCalculator calculator) {
   this.calculator = calculator;
   …
}

public class PathCalculator {
       public GeneralPath calculatePathEndToEnd(Rectangle sourceRect, Rectangle targetRect);
       public GeneralPath calculatePathEndToStart(Rectangle sourceRect, Rectangle
targetRect);
       public GeneralPath calculatePathStartToEnd(Rectangle sourceRect, Rectangle
targetRect);
       public GeneralPath calculatePathStartToStart(Rectangle sourceRect, Rectangle
targetRect);
       …
}
```



System Layer

PopupLayer

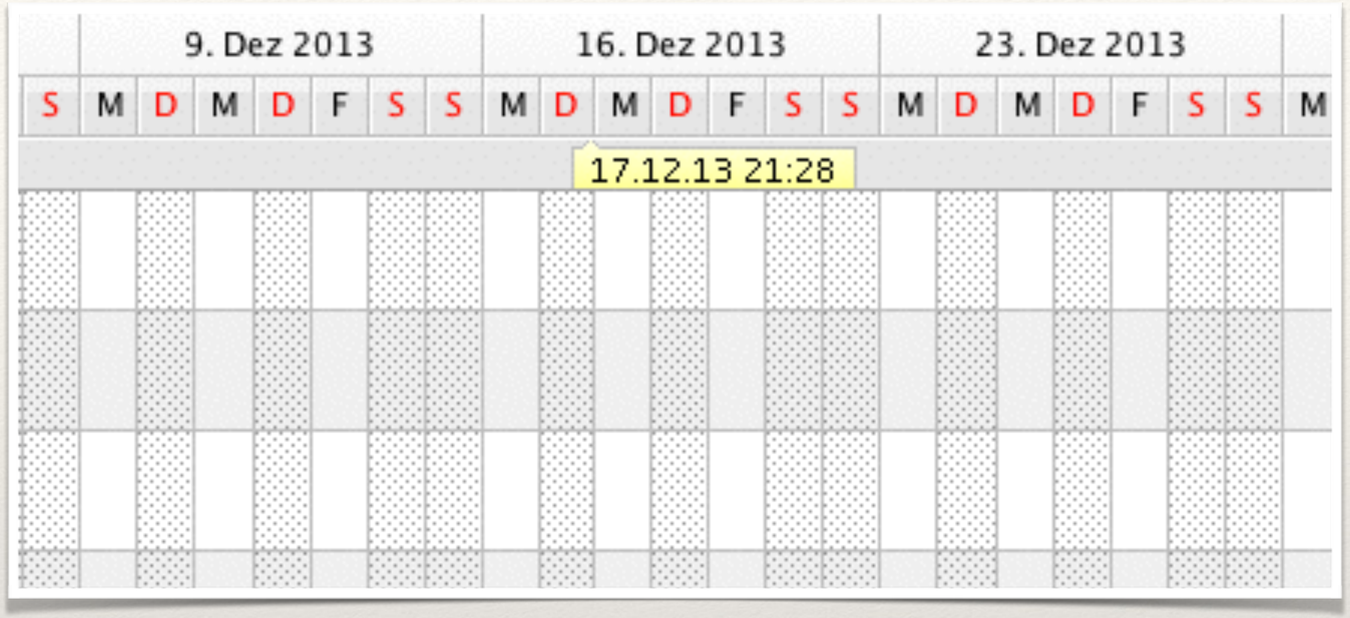
Displays details about timeline objects, an advanced version of a tooltip.

Popup Renderer

```
/**
 * A popup renderer is used to display the information that gets returned from
 * the {@link IPopupPolicy}.
 */
public interface IPopupRenderer extends IRenderer {

     /**
      * Returns the component that will be used as a popup that provides
      * additional information about a timeline object.
      */
     Component getPopupRendererComponent(PopupLayer pl, Object titleValue,
               Object popupValue, TimelineObjectPath path, IGanttChartModel model,
               boolean pinned);

     /**
      * Returns the component that will be used as a popup that provides
      * additional information about a tree table node.
      */
     Component getPopupRendererComponent(PopupLayer pl, Object titleValue,
               Object popupValue, TreePath path, IGanttChartModel model,
               boolean pinned);

     /**
      * Returns the component that will be used as a popup that provides
      * additional information about a relationship.
      */
     Component getPopupRendererComponent(PopupLayer pl, Object titleValue,
               Object popupValue, IRelationship relationship,
               IGanttChartModel model);
}
```

Standard vs. Extended Popup

- Two different popups can be shown for each timeline object, tree node, or relationship: normal or extended.
- User pressing shift while moving the mouse will display the extended version.
- Extended often requires more computation time (e.g. additional database query).



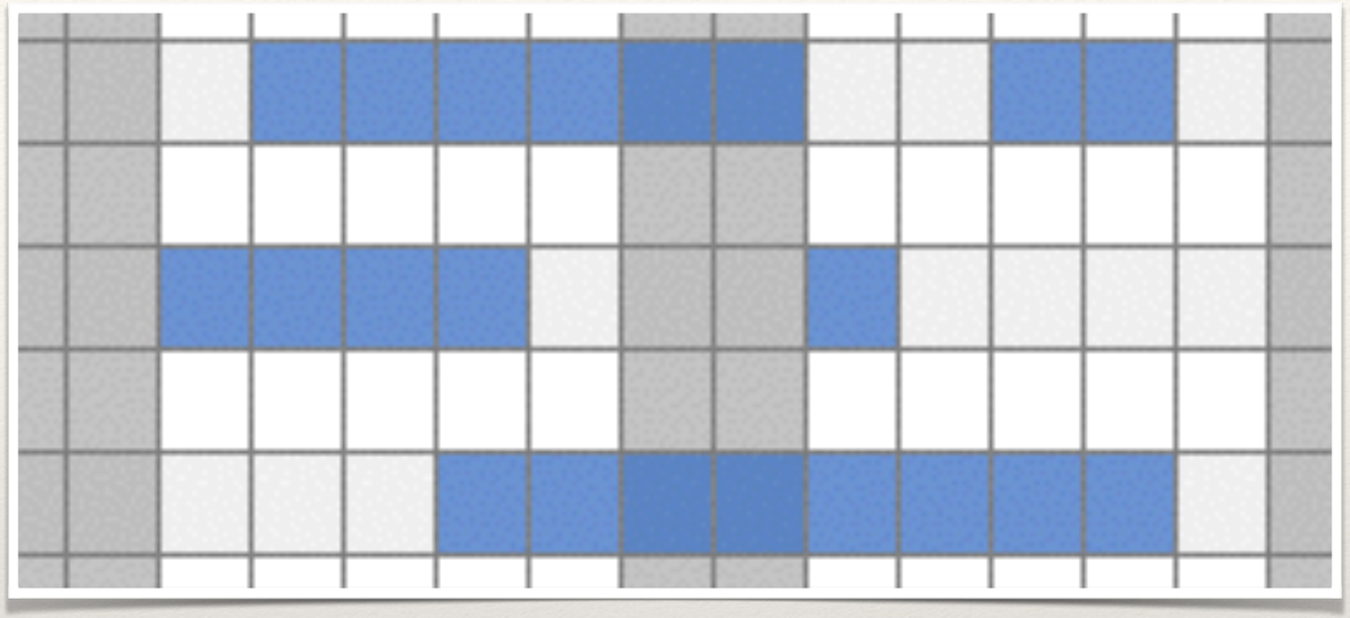
System Layer

RowLayer

Draws row-specific backgrounds.

Row Layer

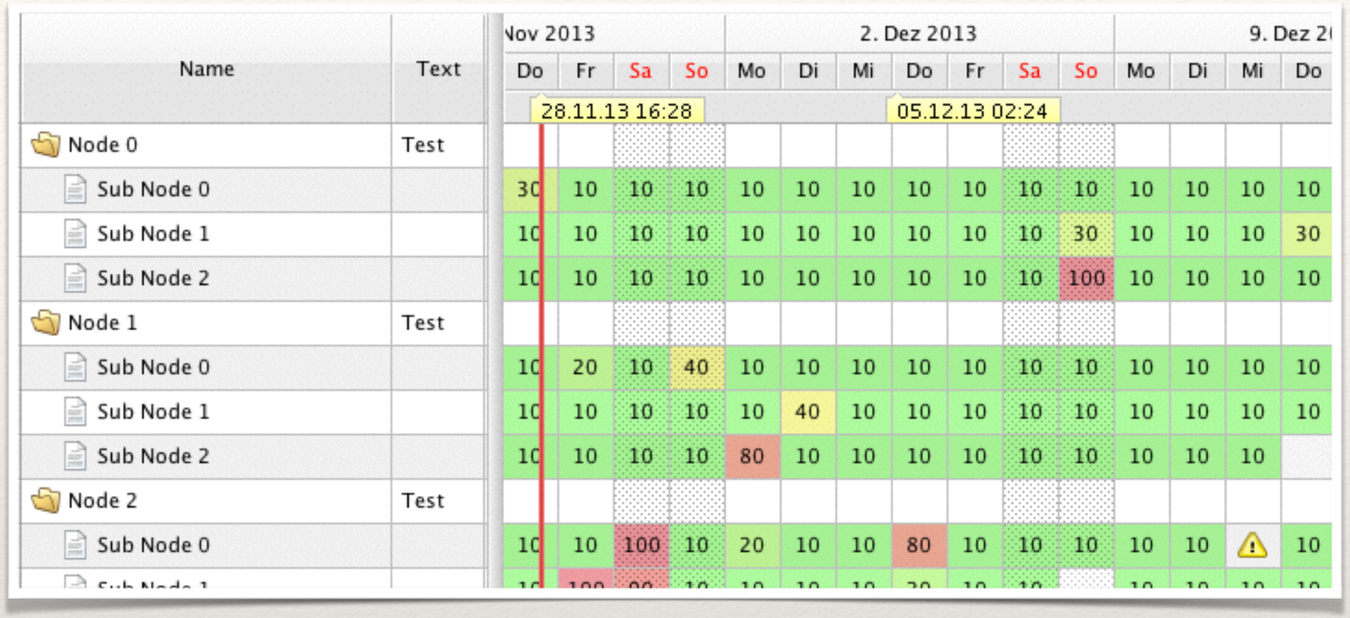
- Each row can have a custom background.
- Use case: colored rows based on some criteria.
- Use case: add text to each timeline object (e.g. departure and arrival location, airports).
- Use case: decorate timeline objects (e.g. red box around timeline objects found after a search).
- Note: nothing „clickable“ can be produced by a row renderer.



System Layer

SelectionLayer

Colors selected time intervals on each row.



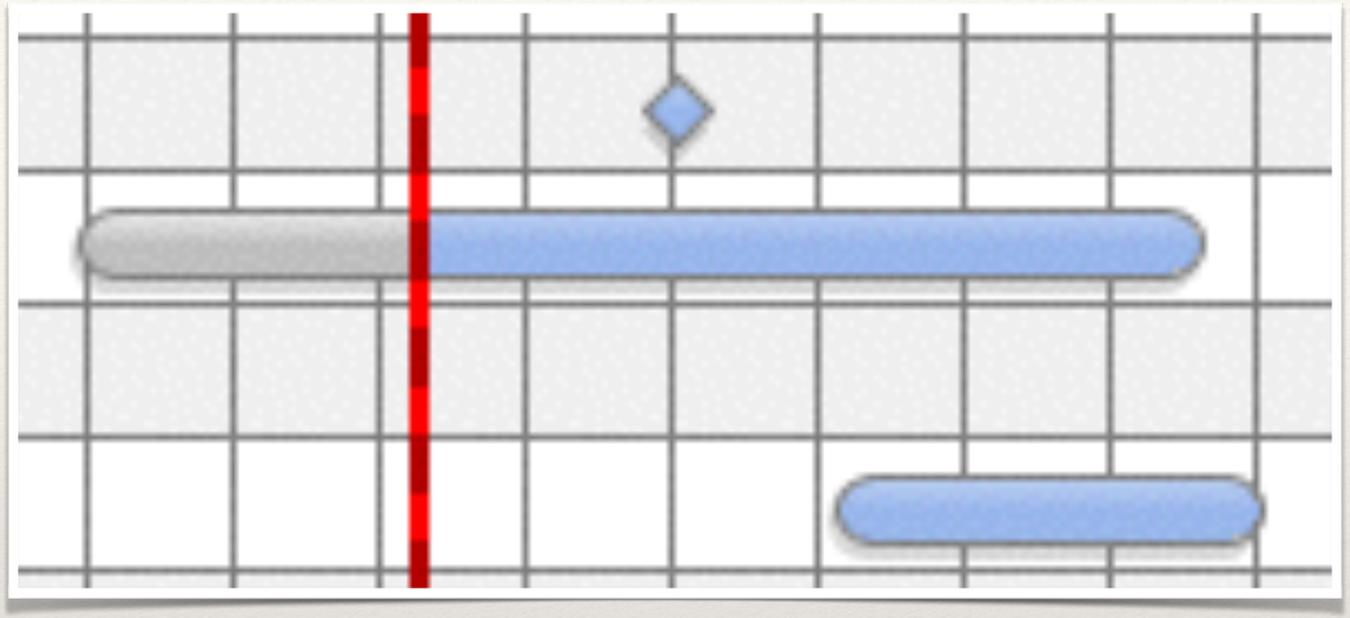
System Layer

SpreadsheetLayer

Used for displaying and editing spreadsheets.

Spreadsheet Layer

- Used for displaying and editing data in a spreadsheet-style fashion.
- Cells are based on a fixed granularity (e.g. days, hours).
- Use case: manually display and edit resource availability or target usage for a given day.
- Use case: provide an aggregated view on the timeline objects (e.g. „5“ flights of an aircraft on a given day).



System Layer

TimeNowLayer

Draws a vertical line at the location of „time now“.

Custom Layers

Custom Layers

- Used to implement application-specific graphical features.
- Call Layer.setCustomLayer(true).
- Use case: add a watermark.
- Use case: showing some kind of dependency between two or more timeline objects (draw a box around them).
- Use case: whatever FlexGantt does not cover out-of-the-box.