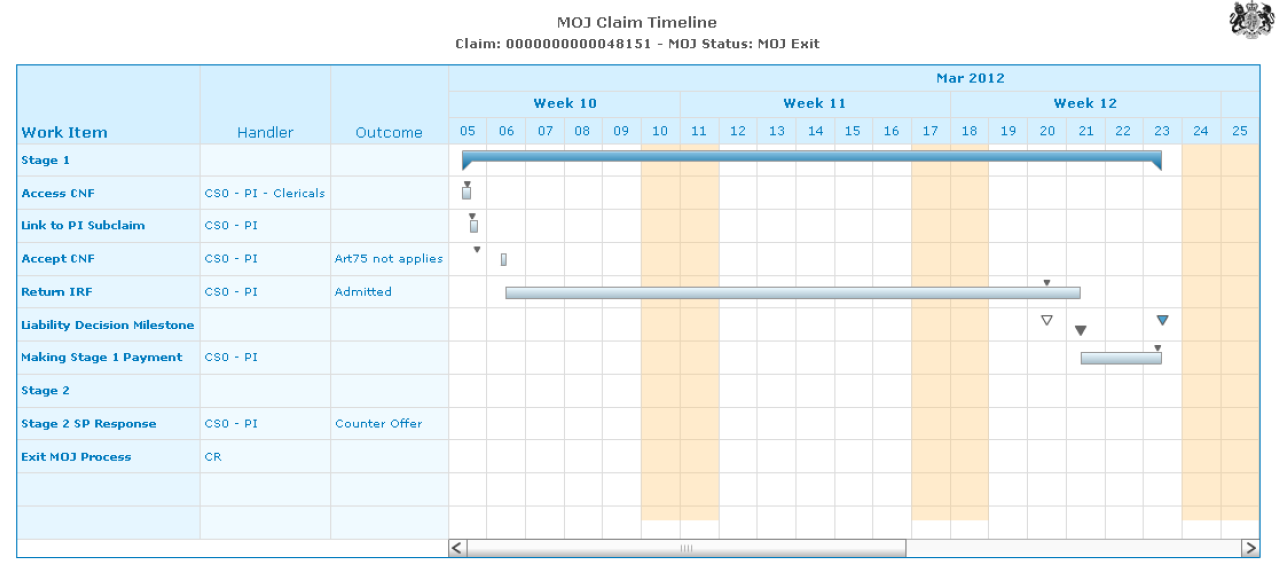
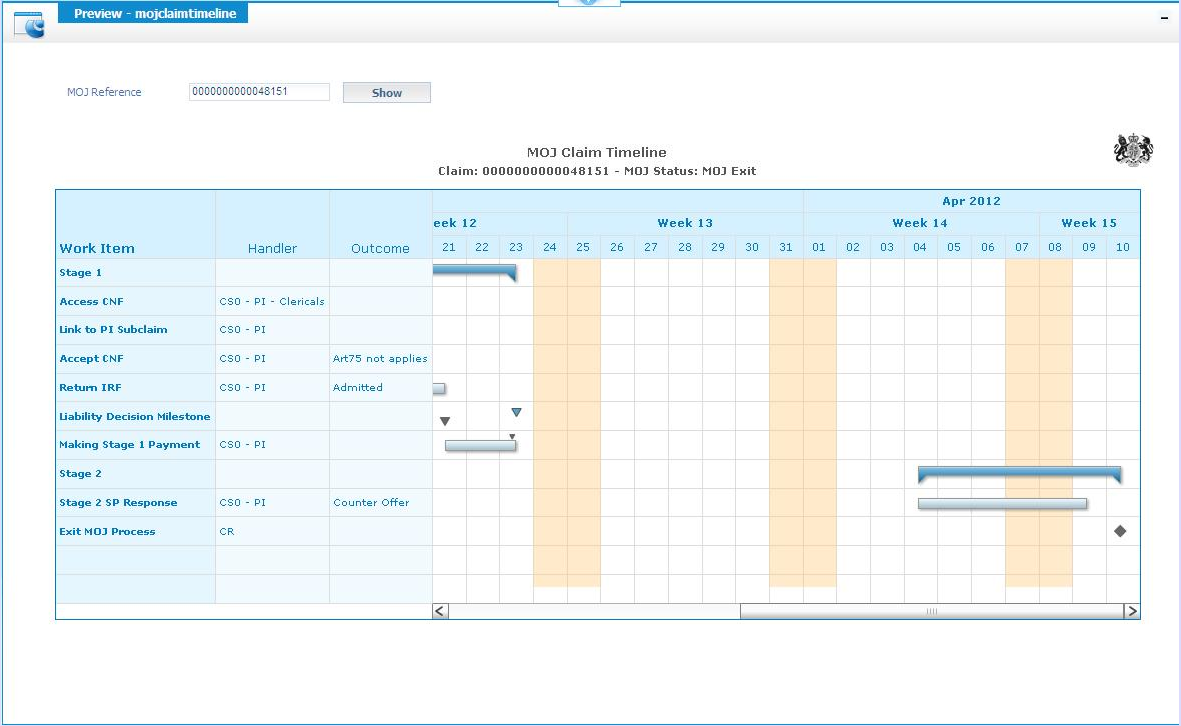
**Charting API – Gantt chart  
cordysganttchart project -** [**http://code.google.com/p/cordysganttchart/**](http://code.google.com/p/cordysganttchart/)

Introduction

To some extend, the Cordys platform offers support for charting (single/multi-series, gauge, BAM charts). Your project though might require extended support/other types of graphs. For example, the requirement might come up to depict the timeline of an executed BPM/Case process into a Gantt Chart.

Example:

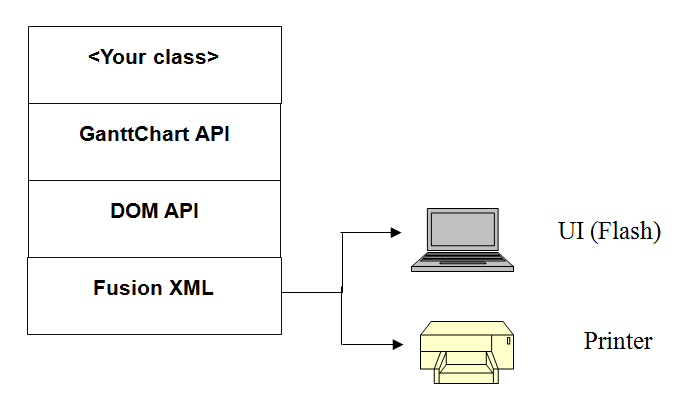




The Cordys platform comes with a version of Fusion Charts (<Install>\Web\wcp\flash\fusion). One of the Fusion charts is the Gantt Chart widget. The Charting API in this opensource project gives a high level API on top of this Fusion widget. The API is designed in such a way that in future any other chart types and any other low level charting technology (apart from Fusion) can be added.

Design

The Gantt Chart is implemented as a server-side component, so it comes as a Java API. Layering:



The DOM API is set up along the lines as applied in the gwt-fusionchart project (<http://code.google.com/p/gwt-fusionchart>).

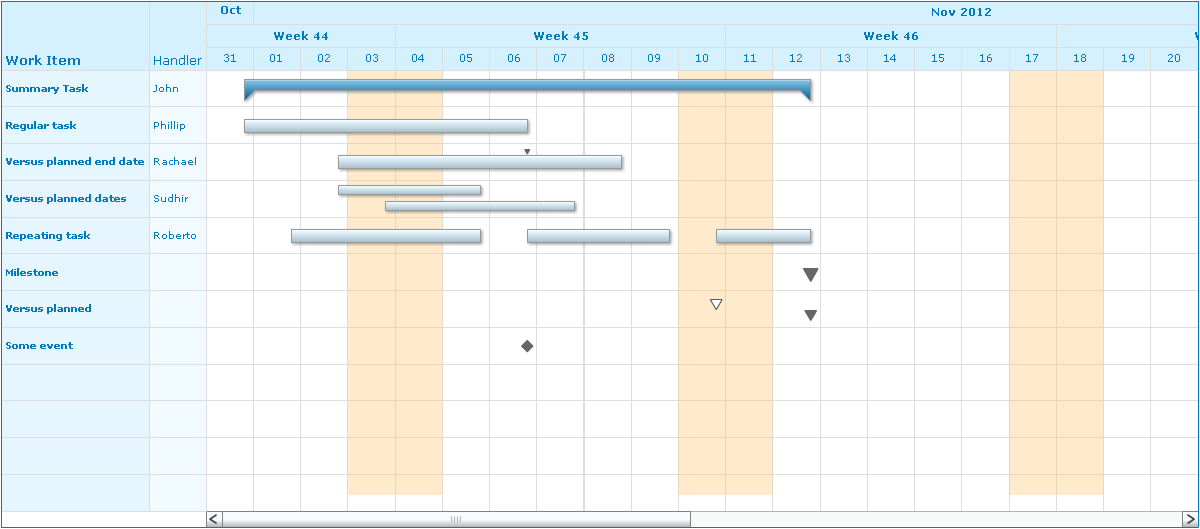
Install/config

The cordysganttchart project comes as an ISVP and as a Cordys project. The Cordys project you can check out from your Cordys workspace by configuring next url: <http://cordysganttchart.googlecode.com/svn/trunk/src/cws>. To run it, attach the chartingWSAppService.chartingWSAppInterface to your WsApps service group.

The project has an example Java class to illustrate the usage of the API (Chart.java in com.ekemait.charting package). It also has an example XForm to illustrate on how to request for the chart from server and populate it client side (User Interfaces/com/ekemait/charting/ExampleGanttChart).

Functionality

Below screendump illustrates the supported elements, varying from depicting summary tasks to events.



The calendar on top supports 3 optional timescales: month, weeks and days. On the left, 1 or more columns can be added.

API Usage

Next are the relevant classes to be used:

* GanttChart.java (com.ekemait.charting.chart.ganttchart). This is the main class. An instance will have references to ChartCalendar, Columns and Rows instances.
* ChartCalendar.java (com.ekemait.charting.chart.ganttchart.calendar).
* Columns.java (com.ekemait.charting.chart.ganttchart.column). Container class for multiple columns.
* Rows.java (com.ekemait.charting.chart.ganttchart.row). Container class for multiple rows. Each row can have 1 or multiple items, where an item is a task, a milestone or an event
* Task.java, Event.java, Milestone.java (com.ekemait.charting.chart.ganttchart.row.item)

Mostly, a row will have one item only, however multiple items are possible to support for example the repeating tasks as from above screenshot.

The GanttChart class has a method to render the chart to a Fusion gantt chart. By this, a fusion XML can be responded back to the client.

For details, see the JavaDoc.

Below shows how the API can be used to populate a gantt chart – the code is from the example class. It generates the gantt chart as in above screenshot.

/\*\*

\* Get an example gantt chart

\*

\* **@return** a fusion gantt chart in XML string representation

\*/

**public** **static** com.ekemait.charting.Chart getGanttChart()

{

// ininialize new gantt chart

GanttChart ganttChart = **new** GanttChart();

ganttChart.setCaption("Claim Timeline");

ganttChart.setSubcaption("Claim: 00000000000235626 - Status: For Approval");

ganttChart.setNumberVisibleDays(*DEFAULT\_NUMBER\_VISIBLE\_DAYS*);

// set export parameters

ganttChart.setExportHandler("fcExporter");

ganttChart.setExportFileName("GanttChartExample");

// set the chart calendar

ChartCalendar chartCalendar = ganttChart.getChartCalendar();

Calendar calDate = Calendar.*getInstance*();

chartCalendar.setStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, *DEFAULT\_NUMBER\_VISIBLE\_DAYS* \* 2);

chartCalendar.setEndDate(calDate.getTime());

chartCalendar.setNonWorkableDayOfWeek(Calendar.*SATURDAY*);

chartCalendar.setNonWorkableDayOfWeek(Calendar.*SUNDAY*);

chartCalendar.createTimeScale(TimeScale.*TU\_MONTHS*);

chartCalendar.createTimeScale(TimeScale.*TU\_WEEKS*);

chartCalendar.createTimeScale(TimeScale.*TU\_DAYS*);

// create the needed gantt chart columns

Columns columns = ganttChart.getColumns();

columns.createColumn("Handler");

// create the gantt chart rows

Rows rows = ganttChart.getRows();

*createChartRows*(rows);

// generate the fusion chart

com.ekemait.charting.fusion.ganttchart.GanttChart fcGanttChart = ganttChart.generateFusionGanttChart();

// any customization of the generated chart

// add a trendline for today

// Trendlines trendlines = fcGanttChart.createTrendlinesNode();

// Line line = trendlines.createLineNode();

// line.setStart(Util.formatDate(new Date(), GanttChart.FC\_INPUTDATE\_JAVAFORMAT));

// line.setDisplayValue("Now");

// set the return xml

Chart chart = **new** Chart();

String chartXML = fcGanttChart.toString();

chart.setChartXML(chartXML);

// return the result

**return** chart;

}

/\*

\* Add example chart rows

\*/

**private** **static** **void** createChartRows(Rows rows)

{

**int** numberRows = 0;

// add new row. Each row can have 1 or more tasks

Row row = rows.createRow();

row.setColumnText("Handler", "John");

// summary task

Task task = row.createTask();

task.setDescription("Summary Task");

Calendar calDate = Calendar.*getInstance*();

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 12);

task.setActualEndDate(calDate.getTime());

task.setIsSummaryTask(**true**);

numberRows++;

// regular task

row = rows.createRow();

row.setColumnText("Handler", "Phillip");

task = row.createTask();

task.setDescription("Regular task");

calDate = Calendar.*getInstance*();

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 6);

task.setActualEndDate(calDate.getTime());

numberRows++;

// regular task with planned end date

row = rows.createRow();

row.setColumnText("Handler", "Rachael");

task = row.createTask();

task.setDescription("Versus planned end date");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 4);

task.setPlannedEndDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

task.setActualEndDate(calDate.getTime());

numberRows++;

// regular task with planned start/end

row = rows.createRow();

row.setColumnText("Handler", "Sudhir");

task = row.createTask();

task.setDescription("Versus planned dates");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

task.setPlannedStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 1);

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

task.setPlannedEndDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

task.setActualEndDate(calDate.getTime());

numberRows++;

// repeating task, so one row with multiple tasks

row = rows.createRow();

row.setColumnText("Handler", "Roberto");

task = row.createTask();

task.setDescription("Repeating task");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 1);

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 4);

task.setActualEndDate(calDate.getTime());

task = row.createTask();

task.setDescription("Repeating task");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 6);

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 3);

task.setActualEndDate(calDate.getTime());

task = row.createTask();

task.setDescription("Repeating task");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 10);

task.setActualStartDate(calDate.getTime());

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

task.setActualEndDate(calDate.getTime());

numberRows++;

// milestone

row = rows.createRow();

Milestone milestone = row.createMilestone();

milestone.setDescription("Milestone");

//milestone.setPlannedEventDate(rowItem.plannedDate);

//milestone.setExternalPlannedEventDate(rowItem.externalPlannedDate);

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 12);

milestone.setActualEventDate(calDate.getTime());

numberRows++;

// planned/actual milestone

row = rows.createRow();

milestone = row.createMilestone();

milestone.setDescription("Versus planned");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 10);

milestone.setPlannedEventDate(calDate.getTime());

//milestone.setExternalPlannedEventDate(rowItem.externalPlannedDate);

calDate.add(Calendar.*DAY\_OF\_MONTH*, 2);

milestone.setActualEventDate(calDate.getTime());

numberRows++;

// event

// can be used for example when a claim is rejected

row = rows.createRow();

Event event = row.createEvent();

event.setDescription("Some event");

calDate = Calendar.*getInstance*();

calDate.add(Calendar.*DAY\_OF\_MONTH*, 6);

event.setActualEventDate(calDate.getTime());

numberRows++;

// if less than minimum number of rows, fill up with empty rows

// to prevent a to big row height

**for** (**int** rn=numberRows; rn < *MINIMUM\_NUMBER\_ROWS*; rn++)

{

rows.createRow();

}

}