# IBM ILOG Views

# Gantt V5.3

# User's Manual

**June 2009**

# Copyright notice



## Trademarks

IBM, the IBM logo, ibm.com, Websphere, ILOG, the ILOG design, and CPLEX are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

## Notices

For further information see <*installdir*>/license/notices.txt in the installed product.

# *Table of Contents*

# About This Manual

This manual describes how to use the set of Gantt chart classes to manage the displaying and editing of scheduling data. These classes are provided in a library based on the following IBM® ILOG® Views packages:

- Foundation
- Gadgets
- Managers
- Grapher

## What You Need to Know

This manual assumes that you are familiar with the PC or UNIX® environment in which you are going to use IBM® ILOG® Views, including its particular windowing system. Since IBM ILOG Views is written for C++ developers, the documentation also assumes that you can write C++ code and that you are familiar with your C++ development environment so as to manipulate files and directories, use a text editor, and compile and run C++ programs.

## Manual Organization

This manual is organized as follows:

- **Chapter 1, *Gantt Chart Functionalities*** describes the main functionalities of the Gantt package.
- **Chapter 2, *Using the Gantt Chart Through Examples*** illustrates how to use the Gantt package through concrete examples.
- **Appendix A, *Overview of Gantt Samples*** lists the samples provided with the Gantt package.

## Notation

### Typographic Conventions

The following typographic conventions apply throughout this manual:

- Code extracts, file names, and entries to be made by the user are written in `courier` typeface.

### Naming Conventions

Throughout this manual, the following naming conventions apply to the API.

- The names of types, classes, functions, and macros defined in the IBM® ILOG® Views libraries begin with `Ilv`.
- The names of classes as well as global functions are written as concatenated words with each initial letter capitalized:

```
class IlvDrawingView;
```

- The names of virtual and regular methods begin with a lowercase letter; the names of static methods start with an uppercase letter:

```
virtual IlvClassInfo* getClassInfo() const;

static IlvClassInfo* ClassInfo*() const;
```

C H A P T E R

# 1

# Gantt Chart Functionalities

Scheduling is the process of assigning resources to activities in time. IBM® ILOG® Views provides a set of Gantt chart classes to manage the displaying and editing of scheduling data, of which `IlvGanttChart` and `IlvGanttChartForm` are the most important.

This chapter gives an overview of the functionalities provided by the Gantt package. The following topics are discussed:

- *High Level Gantt Chart Classes*
- *User Interfaces*
- *Gantt Abstract Data Classes*
- *Gantt Graphic Models*
- *Gantt Grid Objects*
- *Gantt Interactors*
- *Time Scales and IlvGanttChartForm*

## High Level Gantt Chart Classes

The Gantt package of IBM® ILOG® Views provides two high level UI classes to present scheduling data. Before writing any code, you need to choose the one that best fits your

needs. The first class is `IlvGanttChart`, which has a user interface shown in Figure 1.1. The second class is `IlvGanttChartForm`, shown in Figure 1.2.

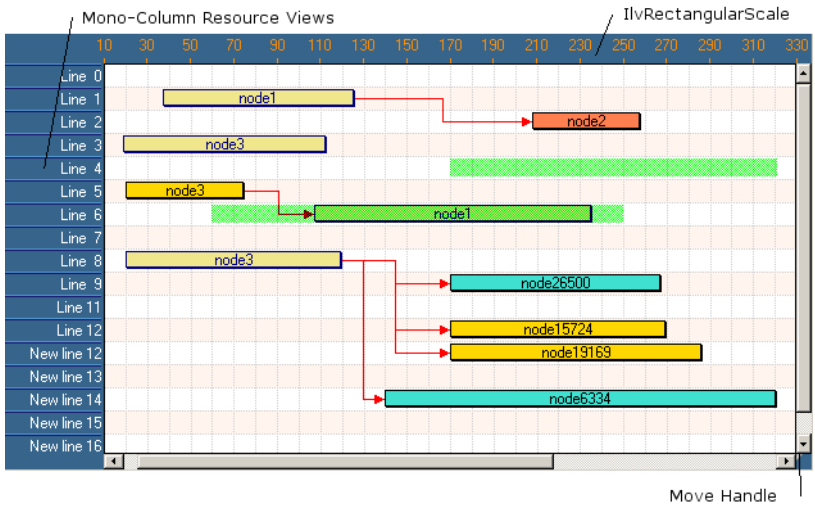



***Figure 1.1*** *The User Interface of IlvGanttChart*

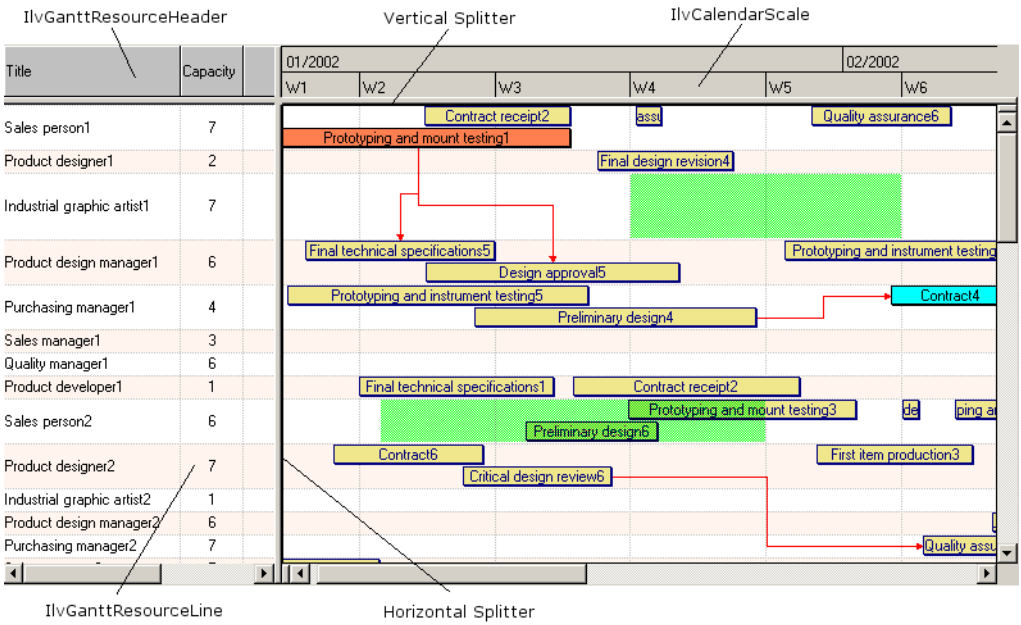



***Figure 1.2*** *The User Interface of IlvGanttChartForm*

`IlvGanttChartForm` is a subclass of `IlvGanttChart`. Most of their functionalities are common or similar. The main differences between the two classes consist in how they manage scales and resources:

- `IlvGanttChart` can only use `IlvRectangularScale` as scales, whereas `IlvGanttChartForm` can use any `IlvGraphic` object as scales. See the example in `<ILVHOME>/samples/gantt/calendar` to see how to use `IlvCalendarScale` as scales of `IlvGanttChartForm`.
- `IlvGanttChart` can only show resources in one column, whereas `IlvGanttChartForm` can display resources in a table-like manner. `IlvGanttChartForm` has an `IlvGanttResourceHeader` object that you can use to resize the columns.
- The width of the resource views are fixed in `IlvGanttChart`, whereas it can be changed in `IlvGanttChartForm` by moving the first horizontal splitter. You should also notice that `IlvGanttChart` uses a move handle to resize views while `IlvGanttChartForm` uses vertical and horizontal splitters.

Therefore, if you need to show your resources in a multi-column manner or you need more sophisticated scales, the `IlvGanttChartForm` class is the best choice.

## User Interfaces

The Gantt chart can display and edit data consistently by means of the Gantt chart manager and grapher views. These views are created inside the Gantt chart object and attached to a manager (two views for Gantt lines), and a grapher (four views for Gantt nodes and links).

### Manager Views

The 2 manager views, which are on the left of the user interface, are created to display `IlvGanttLine` objects. They are indexed as follows:

- `0 top`
- `1 bottom`

Grids (`IlvGanttLineGrid` objects) can be displayed in the manager views.

### Grapher Views

The 4 grapher views, which are on the right of the user interface, are created to display `IlvGanttNode`, `IlvGanttSubNode`, and `IlvGanttLink` objects. The grapher views are indexed as follows:

- `0 topLeft`

- 1 `topRight`
- 2 `bottomLeft`
- 3 `bottomRight`

By default, the four grapher views are visible. In the example provided (*Using the Gantt Chart Through Examples*) one view is displayed while the other ones are hidden. The user can scroll these views by using the keyboard or mouse. Scrolling can be done step-by-step or page-by-page.

Grids can also be drawn in the grapher views.

## Scales

Scales are associated with the grapher views, that is, there are two scales at the top of the views: `topleft` (index `0`) and `topright` (index `1`). In `IlvGanttChart`, the scales are `IlvRectangularScale` objects. Special interactors are installed on the scales and allow you to horizontally zoom the grapher views.

In `IlvGanttChartForm`, the built-in `IlvRectangularScale` scales are not used. Users can set customized scales to `IlvGanttChartForm`. The sample `<ILVHOME>/samples/gantt/calendar` shows a way of using time scales in `IlvGanttChartForm`.

Figure 1.3 shows the main components of a Gantt chart:

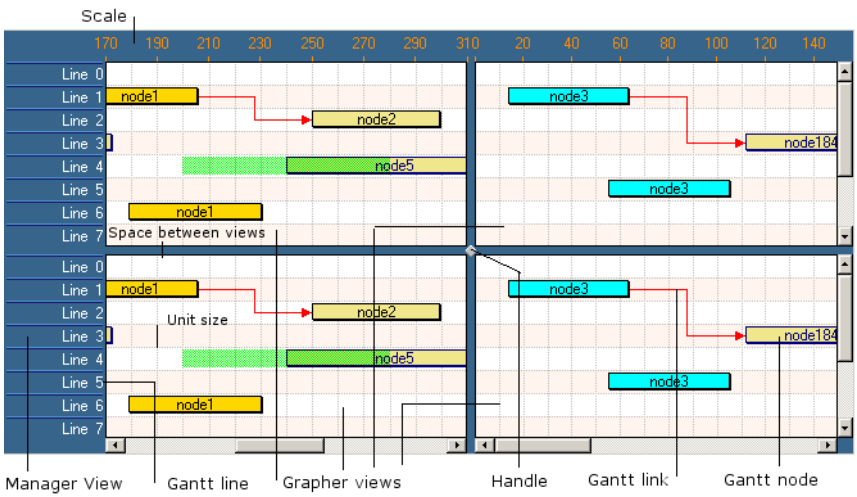



***Figure 1.3*** *Description of the Gantt Chart Found in the Month Sample*

## Gantt Abstract Data Classes

Gantt data classes are objects used to represent user's schedule data. The following Gantt abstract data objects are provided:

- *Resources* - The general inputs of a scheduling problem are as follows:
  - Each *resource* is identified by its name.
  - A resource has a given capacity that describes the number of activities it can perform at one time.
  - A resource with a capacity equal to `1` is called a *unary* resource.
  - Resources are described in the Gantt chart through `IlvGanttLine` class objects.
- *Activities and subactivities* - Each activity is identified by its name. It is composed of one or many *subactivities*, each of which has a minimum and maximum start-time, and a minimum and maximum end-time. A subactivity has a given capacity that indicates its needed resource capacity. Activities and subactivities are described in the Gantt chart through `IlvGanttNode` and `IlvGanttSubNode` class objects.
- *Breaks* - Note that the `IlvGanttNode` and `IlvGanttSubNode` objects might have a flag indicating whether it is a break activity or not.
- *Precedence constraints* - Some activities must start or end before other activities begin or end. These constraints are described in the Gantt chart through objects of class `IlvGanttLink`. A *link* has a starting activity, an ending activity, and a delay. The delay is the time that elapses between the two given activities.

All these objects inherit from the abstract class `IlvGanttAbstractObject`.

### Populating Gantt Charts

When adding Gantt data objects to the `IlvGanttChart` or `IlvGanttChartForm`, graphic objects are created to present the data objects. The following methods of `IlvGanttChart` are provided to populate the Gantt chart:

- Use the methods IlvGanttChart::`addLine()`, IlvGanttChart::`insertLine()`, and IlvGanttChart::`removeLine()` of the `IlvGanttChart` to manage `IlvGanttLine` objects.
- Use the methods IlvGanttChart::`addNode()`, IlvGanttChart::`addSubNode()`, IlvGanttChart::`removeNode()`, and IlvGanttChart::`removeSubNodes()` to manage `IlvGanttNode` and `IlvGanttSubNode` objects of the `IlvGanttChart`.
- Use the method IlvGanttChart::`setSubNodeValues()` to update the values of the subnode already added to the Gantt chart.

- Use the methods IlvGanttChart::`addLink()` and IlvGanttChart::`removeLink()` to manage `IlvGanttLink` objects in the Gantt chart.

## Gantt Graphic Models

Every Gantt abstract object is associated with a graphic representation composed of graphic objects of any subclass of `IlvGraphic`. When a line, node, subnode, or link is added to the Gantt chart, a default graphic representation provided by the Gantt library is used to display it.

To customize a Gantt chart, the user can define one graphic model for each abstract object to create its own graphic representation. Each time an instance of an abstract object for which the user has defined a graphic model is added to the chart, a copy of this model is made. The position and size of the resulting graphic object, that is, its bounding box, are computed by the Gantt chart according to their specific internal values. For example, the bounding box of nodes and subnodes depends on the maximum and minimum start and end-times.

### Specifying Graphic Models

The `IlvGanttChart` class provides member functions for setting and retrieving graphic models for lines, nodes, subnodes, and links:

- IlvGanttChart::`getLineGraphicModel`
- IlvGanttChart::`setLineGraphicModel`
- IlvGanttChart::`getLinkGraphicModel`
- IlvGanttChart::`setLinkGraphicModel`
- IlvGanttChart::`getSubNodeGraphicModel`
- IlvGanttChart::`setSubNodeGraphicModel`
- IlvGanttChart::`getBreakGraphicModel`
- IlvGanttChart::`setBreakGraphicModel`

See the example provided (*Using the Gantt Chart Through Examples*). In this example, the graphic model used to represent lines and nodes is an `IlvGraphicSet`, which is composed of two `IlvMessageLabel` objects.

## Gantt Grid Objects

The `IlvGanttChart` class has an integrated grid mechanism that draws grids for the Gantt chart when the grapher views are being refreshed. Note that this grid mechanism exists only

for grapher views. This integrated grid can be enabled or disabled by calling the method IlvGanttChart::`showGrid()`. The grid is drawn by the method IlvGanttChart::`drawGrid()`, which is a virtual method that can be overridden if the default implementation does not meet your needs. The grid drawing is invoked by `IlvGanttGridViewHook`(s) that are installed by default to all grapher views of the Gantt chart.

The integrated grids draw only when the grapher views of the Gantt chart is refreshed. Therefore, they cannot be dumped when you want to print the contents of the grapher. To resolve this problem, several grid classes are implemented as `IlvGraphic` objects and can be dumped with the contents of the associated grapher.

### Predefined Grid Classes

`IlvGanttHorizontalGridImpl` is an abstract class that implements the basic functions of the horizontal grids. It is designed to be subclassed to implement concrete grids. `IlvGanttHorizontalGrid` is the basic class for most of the grid classes.

`IlvGanttLineGrid` and `IlvGanttLineReliefGrid` are grids implemented for Gantt lines. Use the method `IlvGanttChart::setLineGrid()` to specify the grid to be used. The sample `<ILVHOME>/samples/gantt/simple` uses an `IlvGanttLineReliefGrid`.

`IlvGanttResourceGrid` is also a line grid but it is designed only for an `IlvGanttChartForm`. The sample `<ILVHOME>/samples/gantt/calendar` uses this grid.

Grids for Gantt rows are also provided. `IlvGanttRowGrid` is a grid designed for Gantt rows. Use the method `IlvGanttChart::setRowGrid()` to specify the grid to be used for Gantt rows. The sample `<ILVHOME>/samples/gantt/simple` uses this grid.

A special grid, `IlvGanttCalendarGrid`, is also provided for Gantt rows. This grid draws not only the horizontal grid for Gantt rows but also the vertical grid, which dynamically adjusts its graduation according to the current used zoom level. The sample `<ILVHOME>/samples/gantt/calendar` uses this grid.

## Gantt Interactors

The Gantt package provides a set of interactors that allow users to manipulate the components or the data of the Gantt chart.

### Select and Move Interactors

- `IlvGanttSelectInteractor` allows users to select and move Gantt lines, nodes, and subnodes. It can also be used to select Gantt links.

- `IlvGanttHandleInteractor` allows users to resize the grapher views of the Gantt chart.

### Creation Interactors

- `IlvGanttAddNodeInteractor` allows users to add Gantt nodes and breaks.
- `IlvGanttAddLinkInteractor` allows users to add Gantt links.
- `IlvGanttDragDropInteractor` allows users to duplicate Gantt subnodes between grapher views.

### Zoom Interactors

- `IlvGanttPanZoomInteractor` allows users to use the left button to pan, and use the middle button or the mouse wheel to zoom the grapher views.
- `IlvGanttScaleInteractor` allows users to zoom with the `IlvRectangularScale`(s) of the Gantt chart.
- `IlvGanttZoomInteractor` allows users to zoom the manager views of the Gantt chart.

## Time Scales and IlvGanttChartForm

The `IlvGanttChartForm` class can use external graphic objects as scales. The sample `<ILVHOME>/samples/gantt/calendar` shows how to use an `IlvCalendarScale` object as the scale of the `IlvGanttChartForm`. The external scale can be specified by calling the method `IlvGanttChartForm::setCompositeScale()`.

### Positioning the Time Scales

When you create an `IlvCalendarScale`, the horizontal position and size of the scale must correspond to the start and end parameters of the Gantt chart. The height of the scale must be the same scale height parameter you used for creating the Gantt chart. The Y position of the scale is supposed to be `-100`, so the bounding box of your scale is defined by the following `IlvRect`:

```
IlvRect(start, -100, end-start, scaleHeight)
```

In cases where your time scale is not created with the position and dimension mentioned above, the `IlvGanttChartForm` places the time scale at the correct position and dimension in the Gantt chart form when you call the method IlvGanttChartForm::`setCompositeScale()`.

### Using Time Converters

The time scales, such as `IlvCalendarScale`, use `IlvTimeConverter` to convert the dates to coordinates used to draw the scales. The `IlvGanttCalendarGrid`, used in the sample `<ILVHOME>/samples/gantt/calendar`, also uses an `IlvTimeConverter`. To make the time scale and the calendar grid synchronous, we suggest that the time scale and the calendar grid use the same time converter.

If you use dates to define the start and end times of the activities, you need to convert them to grapher coordinates in order to add them to the Gantt chart. We suggest you use the same time converter as the time scale and the calendar grid to make the conversions.

C H A P T E R

# 2

# Using the Gantt Chart Through Examples

In this chapter, we show how to use the Gantt package through concrete examples. C++ sources code is provided to show how to use the API of the Gantt package.

The following sections provide comments on the sample files provided in the `<ILVHOME>/samples/gantt/load` and `<ILVHOME>/samples/gantt/month` directories.

To simplify the use of this sample, only the main header and source files will be considered:

- `include/month.h`, `src/month.cpp`

  The `month` files (found in the `<ILVHOME>/samples/gantt/month` directory) define the main user interface. The top-level window is a subtype of `IlvGadgetContainer`.

- `include/loadgantt.h`, `src/loadgantt.cpp`

  The `loadgantt` files (found in the `<ILVHOME>/samples/gantt/load` directory) are the description of the `LoadGanttChart` class, which is the intermediate class between the `IlvGanttChart` superclass and the `MonthGanttChart` class described in this section.

- `include/monthgantt.h`, `src/monthgantt.cpp`

  The `monthgantt` files (found in the `<ILVHOME>/samples/gantt/month` directory) contain the definition and the declaration of the `MonthGanttChart` class, a subclass of the `IlvGanttChart` class.

This example shows some functionalities of the `IlvGanttChart` class. It covers the following topics:

- *The Entry Point of the Sample*
- *Subclassing IlvGanttChart*
- *Customizing Gantt Scales*
- *Modifying the Start and End Times*
- *Automatically Updating Scales*
- *Customizing Gantt Chart Grids*
- *Defining Graphic Models for Lines and Nodes*
- *Defining a Graphic Model for Links*
- *Registering Gantt Data Change Hooks*
- *Using the Handle*

## The Entry Point of the Sample

The `main` function of the sample is defined in `<ILVHOME>/samples/gantt/month/src/month.cpp`. If you look at the source code, you will find the follows tasks accomplished in the `main` function:

- The display is created.
- The main window of the sample is created.
- The customized Gantt chart object is created and added to the main window.
- The toolbar manipulating the Gantt chart is created.
- The graphic models of the Gantt chart are specified.
- The Gantt chart object is populated with data.
- The line grid is created for the Gantt chart.
- The colors of the Gantt chart are customized.
- The handle of the Gantt chart is hidden to make a mono view Gantt chart.

```
// --------------------------------------------------------------------------
int
main(int argc, char* argv[])
{
    // Create the display
    IlvDisplay* display
        = new IlvDisplay("IBM ILOG Views Gantt Chart", 0, argc, argv);
```

```
if (!display || display->isBad()) {
    IlvFatalError("Couldn't create display");
    delete display;
    return -1;
}

// Add the path to toolbar files
AppendGanttDataPath(display);

// Creating top view
IlvGadgetContainer* top = new IlvGadgetContainer
    (display, "GanttChart", "IBM ILOG Views Gantt Chart Month Demo",
    IlvRect(200, 100, 720, 376), IlFalse);
top->setDestroyCallback(Quit);

IlInt start, end;
IlUShort step, day;
IlvGanttComputeDay(3, 2002, day, start, end, step);

// Creating Chart
MonthGanttChart* gantt =
    new MonthGanttChart(display, IlvRect(0, 32, 720, 344),
                        start, end, step, day, 160);

// Create the toolbar
IlvGanttToolBar* toolbar = new MonthToolBar(display, gantt);
top->addObject(toolbar);

// Specify the new graphic models
gantt->setSubNodeGraphicModel(CreateSubNodeGraphicModel(display));
gantt->setLineGraphicModel(CreateLineGraphicModel(display));
gantt->setLinkGraphicModel(CreateDoubleLinkGraphicModel(display));
gantt->setBreakGraphicModel(CreateBreakGraphicModel(display));

// Populate the Gantt chart with data
PopulateGanttChart(gantt, start, end, 12L);

// Add horizontal grids
IlvColor* steelblue3 = display->getColor("steelblue3");
IlvColor* midnightblue = display->getColor("midnightblue");
IlvPalette* gridPal = display->getPalette(steelblue3, midnightblue);
IlvGanttLineGrid* linGrid = new IlvGanttLineReliefGrid(gantt, gridPal);
gantt->setLineGrid(linGrid, IlFalse, 0);

// Attach the toolbar
IlvSetAttachment(toolbar, IlvTop, IlvFixedAttach, 0);
IlvSetAttachment(toolbar, IlvLeft, IlvFixedAttach);
IlvSetAttachment(toolbar, IlvRight, IlvFixedAttach);
IlvSetAttachment(toolbar, IlvHorizontal, IlvElasticAttach);

// Attach the Gantt chart
IlvSetAttachment(gantt, IlvTop, IlvFixedAttach, 32);
IlvSetAttachment(gantt, IlvBottom, IlvFixedAttach);
IlvSetAttachment(gantt, IlvLeft, IlvFixedAttach);
IlvSetAttachment(gantt, IlvRight, IlvFixedAttach);
IlvSetAttachment(gantt, IlvVertical, IlvElasticAttach);
IlvSetAttachment(gantt, IlvHorizontal, IlvElasticAttach);
```

```
    // Add the Gantt chart
    top->addObject(gantt);
    // Showing top view
    top->show();

    // Color customizations
    IlvColor* cyan = display->getColor("cyan");
    IlvColor* darkorange = display->getColor("darkorange");
    IlvColor* steelblue4 = display->getColor("steelblue4");
    IlvPalette* pal = display->getPalette(cyan, darkorange);
    gantt->getScale(0)->setPalette(pal);
    gantt->getScale(1)->setPalette(pal);
    gantt->setBackground(steelblue4);
    gantt->getManagerView(0)->setBackground(steelblue4);
    gantt->getManagerView(1)->setBackground(steelblue4);
    IlvColor* antiquewhite = display->getColor("antiquewhite");
    gantt->getGridPalette()->setBackground(antiquewhite);

    // Enable the default grid
    gantt->showGrid(IlTrue, IlTrue);

    // Enable double buffering
    gantt->setDoubleBuffering(IlTrue);

    // Make only one view visible
    gantt->moveHandle(IlvPoint(2000, 2000));

    // Running main loop
    IlvMainLoop();
    return 0;
}
```

## Subclassing IlvGanttChart

Some methods of the `IlvGanttChart` class are virtual. These functions are called when the internal states of the Gantt chart are being changed or special actions are taken on the Gantt chart. Users are invited to subclass the `IlvGanttChart` class for two purposes:

- The virtual functions are called to indicate internal events of the Gantt chart.
- The function can be implemented by users to meet their special needs. The new implementation changes the behavior of the Gantt chart.

In the `month` example, we show how to override some virtual functions of the `IlvGanttChart` to customize the grid of the Gantt chart. Here is the definition of the `MonthGanttChart` class:

```
class MonthGanttChart
: public LoadGanttChart
{
public:
    MonthGanttChart(IlvDisplay* display,         const IlvRect& rect,
                    IlvInt      start,           IlvInt         end,
                    IlvUShort   step,            IlvUShort      day,
```

```
                    IlvDim      rowsize,         IlvDim         unitsize=40,
                    IlvUShort   gridThickness=4, IlvPalette*    palette = 0)
        : LoadGanttChart(display, rect, start, end, start, start+7*24,
                         step, rowsize, unitsize, gridThickness, palette),
          _day(day) { makeHolidayPalette(display); }
    virtual ~MonthGanttChart();
    // ____________________________________________________
    virtual void drawGrid(IlvPort* dst, IlvUShort index,
                          IlvBoolean skipCompute=IlFalse,
                          const IlvRegion* clip=0,
  IlvTransformer* t=0,
  IlvDirection d=(IlvDirection)(IlvHorizontal|IlvVertical));
    virtual IlvScale* createScale(IlvDisplay*, const IlvPoint& orig,
                                  IlvDim size, IlvInt start, IlvInt end,
                                  IlvUShort step, IlvPosition p=IlvTop) const;
    virtual void  updateScale(IlvScale*, IlvBoolean redraw=IlTrue) const;
    void computeScaleLabels(IlvInt start, IlvInt end, IlvUShort step,
                            IlvUShort& count, char** labels) const;
    IlvUShort getDay() const { return _day; }
    DeclareTypeInfo();
    DeclareIOConstructors(MonthGanttChart);
protected:
    IlvUShort   _day;
    IlvPalette* _holidayPalette;
    void makeHolidayPalette(IlvDisplay*);
};
```

As you may have noticed, the virtual functions `drawGrid()`, `createScale()`, and `updateScale()` are overridden.

The height and position of the scales can be specified in the Gantt chart constructor. By default, the height is `32` and the location is at the top of the Gantt chart.

## Customizing Gantt Scales

The scale is actually created by the `makeView` call, using the virtual method `IlvGanttChart::createScale`. This method creates and returns the scale graphic object at the given `origin` point. The limits of the time shown are given by `start` and `end`. The `size` parameter indicates the scale size. You can redefine this method to initialize user-defined scale internal values or to customize your own scale.

In the `month` example, the `createScale` method is redefined as follows:

```
IlvScale*
MonthGanttChart::createScale(IlvDisplay* display, const IlvPoint& orig,
                             IlvDim size, IlvInt start, IlvInt end,
                             IlvUShort step, IlvPosition position) const
{
    IlvUShort count;
    char* labels[32];

    // Internal method to create every label displayed on the scale
    computeScaleLabels(start, end, step, count, labels);
```

```
    // Create the scale object
    IlvRectangularScale* scale =
        new IlvRectangularScale(display, orig, size, labels,
                                IlvHorizontal, position, count,
                                12, 10, 5, getPalette());

    // Labels will be centered
    scale->centerLabels(IlTrue);

    // Labels cannot overlap
    scale->drawOverlappingLabels(IlFalse);

    // Clean memory
    for (IlvUShort i = 0; i < count; ++i)
        delete labels[i];

    return scale;
}
```

## Modifying the Start and End Times

To modify the displayed start and end of a scale, use the method `IlvGanttChart::showInterval`. This member function modifies the values of the scale specified by the value `index`, either the left one (`index=0`) or the right one (`index=1`). To prevent the contents of the scale and grapher views from being refreshed, set the `redraw` parameter to `IlFalse`.

## Automatically Updating Scales

To update a scale, use the virtual method `IlvGanttChart::updateScale`. This member function is called when a new transformation is applied to a corresponding grapher view, that is, a zoom or a horizontal translation. You can set the `redraw` parameter to `IlFalse` to prevent the contents of the scale and grapher views from being refreshed.

In the `month` example, the `updateScale` method is redefined as follows:

```
void
MonthGanttChart::updateScale(IlvScale* scale, IlvBoolean redraw) const
{
    IlvContainer* container = getGadgetContainer();
    IlvUShort count;

    // Get index of the scale to be updated (0 if it is the left scale,
    // 1 if it is the right scale)
    IlvUShort hi = (IlvUShort)((scale == getScale(0)) ? 0 : 1);

    // Get the labels to be displayed
    char* labels[32];
    computeScaleLabels(getShownStart(hi), getShownEnd(hi),
                       getStep(hi), count, labels);
```

```
    MONTHGANTTDATA arg;
    arg.count  = count;
    arg.labels = labels;
    container->applyToObject(scale, SetScaleLabels, &arg, redraw);

    // Clean memory
    for (IlvUShort i = 0; i < count; ++i)
        delete labels[i];
}
```

## Customizing Gantt Chart Grids

The `IlvGanttChart` class has methods for dealing with a chart grid. Using these methods, you can show or hide the grid, compute its points, and draw it.

In the grapher views, you can choose to show the grid or not using the method `IlvGanttChart::showGrid`. If the `redraw` parameter is `IlTrue`, the grapher views will be redrawn.

You can specify the grid points using the method `IlvGanttChart::computeGridPoints`. This method computes the grid points for the grapher view specified by the `index` parameter (`0`, `1`, `2`, or `3`). It stores them in two internal data members (one each for the x and y coordinates): `_gridXPoints` and `_gridYPoints`. There are two internal data members for every grapher view.

You can draw the grid with the virtual method `IlvGanttChart::drawGrid`. This method draws the grid in the given destination port. Its `index` parameter indicates for which grapher view the grid points are computed. Its `skipCompute` parameter indicates whether this function calls the `computeGridPoints` member function. This increases performance; for example, it is useless to call `computeGridPoints` if there is no change in the point values and there is just a simple redraw.

The `month` example shows how to customize the drawing of the grid. The weekend periods are highlighted by drawing additional filled rectangles:

```
void
MonthGanttChart::drawGrid(IlvPort* dst, IlvUShort ix,
                          IlvBoolean skipCompute, const IlvRegion* clip,
  IlvTransformer*, IlvDirection)
{
    if (!isShowingGrid() || !getGadgetContainer() || rows() == 0)
        return;
    if (clip)
        _holidayPalette->setClip(clip);

    // The time graduation (step) is 1 hour, 24 means 24 hours per day
    IlvUShort dayInWeek,
              startDay = (IlvUShort)
                         ((getShownStart((IlvUShort)(ix&1))-getStart())/24),
              endDay = (IlvUShort)
```

```
                    ((getShownEnd((IlvUShort)(ix&1))-getStart())/24);
    IlvUShort fvRow = firstVisibleRow((IlvUShort)(ix/2));
    IlvUShort lvRow = lastVisibleRow((IlvUShort)(ix/2));
    IlvUShort toRow = IlvMin((IlvUShort)(lvRow+2), rows());

    // Gets the current grapher view transformer
    IlvTransformer* t = getGrapher()->getTransformer(getGrapherView(ix));
    IlvRect rect;

    // In the visible part, check if there is a Sunday or a Saturday
    // and fill the grapher view regions that correspond to these
    // days with a specific pattern (_holidayPalette)
    IlvRect fromRect, toRect;
    rowBBox(fvRow, fromRect);
    rowBBox(toRow, toRect);
    for (IlvUShort day = startDay; day < endDay; ++day) {
        dayInWeek = (IlvUShort)(((IlvInt)day + (IlvInt)_day - 1) % 7);
        if ((dayInWeek == 6) || (dayInWeek == 0)) {
            rect.moveResize(day*24, fromRect.y(),
                            (IlvDim)getStep((IlvUShort)(ix&1)),
                            (IlvDim)(toRect.y()-fromRect.y()));
            if (t)
                t->apply(rect);
            dst->fillRectangle(_holidayPalette, rect);
        }
    }
    LoadGanttChart::drawGrid(dst, ix, skipCompute, clip);
    if (clip)
        _holidayPalette->setClip();
}
```

## Defining Graphic Models for Lines and Nodes

This code is used to set graphic models to the Gantt objects (lines and nodes):

```
gantt->setSubNodeGraphicModel(CreateSubNodeGraphicModel(display));
gantt->setLineGraphicModel(CreateLineGraphicModel(display));
```

In the `month` example, external functions `CreateSubNodeGraphicModel()` and `CreateLineGraphicModel()`, defined in `<ILVHOME>/samples/gantt/common/utils.cpp`, are used to create the graphic model used for lines as well as for nodes.

```
// --------------------------------------------------------------------------
IlvGraphic*
CreateSubNodeGraphicModel(IlvDisplay* display)
{
    IlvColor* khaki = display->getColor("khaki");
    IlvColor* navy = display->getColor("navy");
    IlvPalette* palette = display->getPalette(khaki, navy);
    palette->lock();
    IlvGraphic* model = new IlvShadowLabel(display, "", IlvRect(0, 0, 100, 20),
                                           1, IlvBottomRight, palette);
    palette->unLock();
    RegisterGanttNameProperty(model);
    return model;
```

```
}
```

The subnode graphic model used is an `IlvShadowLabel` object.

```
// --------------------------------------------------------------------------
IlvGraphic*
CreateLineGraphicModel(IlvDisplay* display)
{
    IlvColor* white = display->getColor("white");
    IlvColor* black = display->getColor("black");
    IlvPalette* palette = display->getPalette(black, white);
    palette->lock();
    IlvMessageLabel* model = new IlvMessageLabel(display,  "",
        IlvRect(0, 0, 100, 20), IlvRight, 0, palette);
    palette->unLock();
    model->setOpaque(IlFalse);
    RegisterGanttNameProperty(model);
    return model;
}
```

The line graphic model used is an `IlvMessageLabel` object.

## Defining a Graphic Model for Links

To represent links, the user can also define a graphic model. To set the link graphic model, use the following method as indicated, after having defined the model:

```
gantt->setLinkGraphicModel(CreateDoubleLinkGraphicModel(display));
```

In the `month` example, a new class (`DoubleLink` class) is defined for a specific kind of link and this new class is used as the Gantt link model.

This model is the result of the local function `CreateDoubleLinkGraphicModel`:

```
// --------------------------------------------------------------------------
IlvGraphic*
CreateDoubleLinkGraphicModel(IlvDisplay* display)
{
    IlvColor* red = display->getColor("red");
    IlvColor* black = display->getColor("black");
    IlvPalette* palette = display->getPalette(black, red);
    palette->lock();
    IlvGraphic* model = new DoubleLink(display, IlTrue, 0,0, palette);
    palette->unLock();
    return model;
}
```

## Registering Gantt Data Change Hooks

This section shows how to handle the name property of the Gantt data object. We show first how to change the name property and then how to get notified when the name property of the Gantt data objects is being changed.

### Name Property

The name of every abstract object is specified in the constructor and can be updated by means of the member function `IlvGanttChart::setObjectName`. This member function sets the name of the Gantt object (line, node, or link) `object` to `name`. It returns `IlTrue` if successful. The object names must be unique in the scope of a given Gantt chart. You can delete the name of an object by setting the `name` parameter to `0` when calling this function. An attempt to erase the name of an unnamed object is an error. Use the `redraw` parameter to specify whether you want the views to be redrawn (`IlTrue`) or not (`IlFalse`).

The subnodes of one activity have the same name but different indexes in the list managed by the node. Thus, subnodes can be distinguished graphically by displaying their indexes.

The Gantt library provides hooks to insert a call to any user-defined function at various points, as follows:

- When a line is inserted into the Gantt chart.
- When a subnode is added to a node.
- When a link is added to the Gantt chart.
- When the `setObjectName` method is called to set or to update any Gantt abstract object name.

### Name Property Change Hook

This hook, which is also useful when graphic models are defined for customization, must have the following signature:

```
typedef void (* IlvGanttSetGraphicName)(IlvGanttAbstractObject* object,
                                        IlvGraphic*             model,
                                        IlvGanttObjectType      type,
                                        IlvAny                  arg);
```

The `object` parameter is an instance of the `IlvGanttLine`, `IlvGanttNode`, or `IlvGanttLink` classes. The `model` parameter is the graphic object used as a model to create the `IlvGanttAbstractObject` object. The `type` parameter is one of the values defined by the `IlvGanttObjectType` enum, and `arg` is additional information (which can be useful, for example, when the `IlvGanttAbstractObject` object is an `IlvGanttSubNode` object).

```
typedef enum IlvGanttObjectType {IlvGanttIsLine,
                                 IlvGanttIsNode,
                                 IlvGanttIsLink};
```

There is a user-defined example in `<ILVHOME>/samples/gantt/common/src/utils.cpp`:

```
// --------------------------------------------------------------------------
void
OnGanttObjectNameChanged(IlvGanttAbstractObject* object,
                         IlvGraphic*             graphic,
                         IlvGanttObjectType      type,
                         IlvAny                  index)
{
    if (graphic->getClassInfo()->isSubtypeOf(IlvMatrix::ClassInfo())) {
        if (type == IlvGanttIsLine ) {
           IlvMatrix* matrix = (IlvMatrix*)graphic;
           IlvAbstractMatrixItem*  item;
           item = matrix->getItem(0,0);
           item->setLabel(object->getName());
           item = matrix->getItem(1,0);
           char buffer[16];
           sprintf(buffer, "%d", ((IlvGanttLine*)object)->getCapacity());
           item->setLabel(buffer);
        }
    } else if(graphic->getClassInfo()->isSubtypeOf
              (IlvMessageLabel::ClassInfo())) {
        IlvMessageLabel* label = (IlvMessageLabel*)graphic;
        label->setLabel(object->getName());
    } else if(graphic->getClassInfo()->isSubtypeOf
              (IlvShadowLabel::ClassInfo())) {
        IlvShadowLabel* label = (IlvShadowLabel*)graphic;
        label->setLabel(object->getName());
    }

    if (type == IlvGanttIsNode ) {
        IlvGanttNode* node = (IlvGanttNode*)object;
        IlvGanttSubNode* subnode = node->getSubNode(*((IlUShort*)index));
        graphic->setNamedProperty(new IlvGanttSubNodeToolTip(subnode));
    }
}
```

This user-defined function of type `IlvGanttSetGraphicName` will automatically be called in the cases previously mentioned.

---

### Registering Name Property Hooks

After have defined the hook function, it must be registered before being called by the Gantt chart object. The registration is based on the IBM® ILOG® Views class property feature, in which a property is registered at the class level.

In the Gantt library, the key for allowing the property value to be retrieved is an `IlvSymbol` defined in the `IlvGanttChart` class:

```
IlvSymbol* IlvGanttChart::nameProperty() const;
```

where IlvSymbol is defined as GanttName. Here is an example of registering the hook function we defined above. You can find this function in <ILVHOME>/samples/gantt/common/src/utils.cpp:

```
// --------------------------------------------------------------------------
void RegisterGanttNameProperty(IlvGraphic* graphic)
{
    // Test if the Gantt name property is added to the IlvGraphic.
    IlvGanttSetGraphicName function =
        IL_FPTRTOANYCAST(IlvGanttSetGraphicName)
        (graphic->getClassProperty(IlvGanttChart::nameProperty(), IlTrue));
    if (function)
        return;

    graphic->AddProperty(IlvGanttChart::nameProperty(), (IlAny)
        IL_FPTRTOANYCAST(IlvGanttSetGraphicName)(OnGanttObjectNameChanged));
}
```

## Using the Handle

As the default, a Gantt chart is created with four views and a diamond positioned in the middle of the grapher views. This diamond allows users to adjust the size of the manager and the grapher views by using the mouse.

The IlvGanttChart::getHandle member function retrieves this handle, and IlvGanttChart::moveHandle moves the handle box to the given position and resizes the manager and grapher views accordingly.

### Mono View Gantt Chart

In the month example, the handle box is moved to the right bottom corner of the screen so that only one grapher view is currently visible:

```
// Make only one grapher view visible
gantt->moveHandle(IlvPoint(display->screenWidth(),
                           display->screenHeight()));
```

### Hiding the Handle

To hide the handle box, you should write:

```
    gantt->getGadgetContainer()->setVisible(gantt->getHandle(), IlFalse);
```

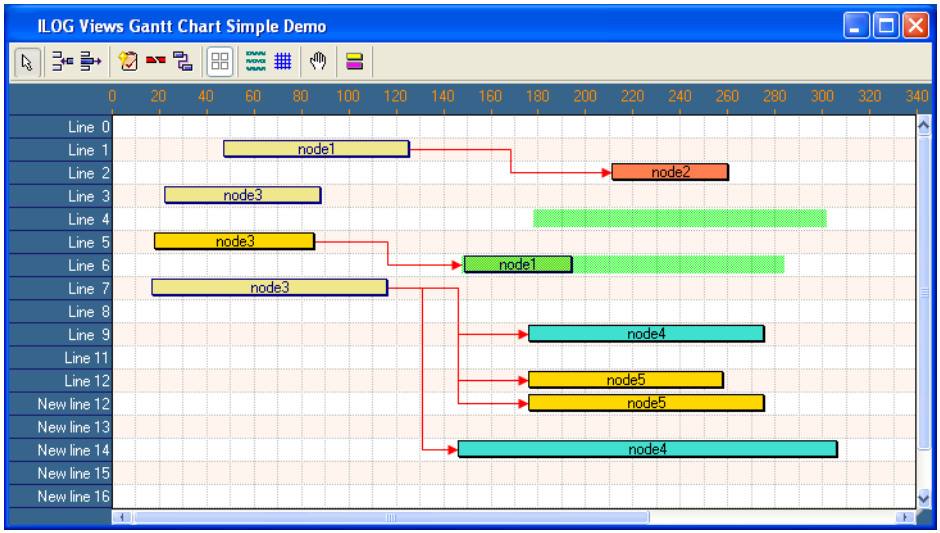
# A

# Overview of Gantt Samples

The best way to learn how to use the Gantt package is to compile and to run the sample programs provided with the Gantt package. Four samples are provided to show various functionalities of the Gantt package and different ways of customizing the Gantt chart. You can find the samples in `<ILVHOME>/samples/gantt`. Refer to the `index.html` files in these directories for detailed explanations.

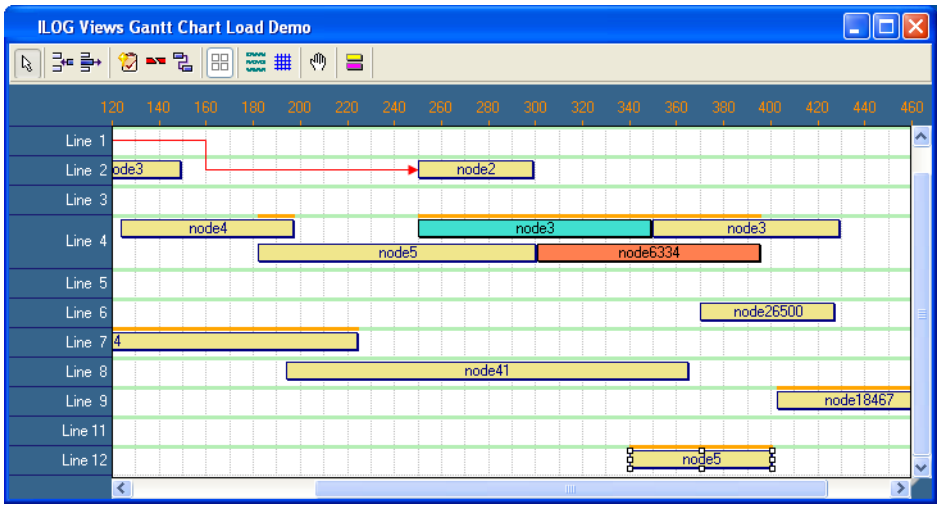
## The Simple Sample

This is a simple example showing how to create an `IlvGanttChart`:



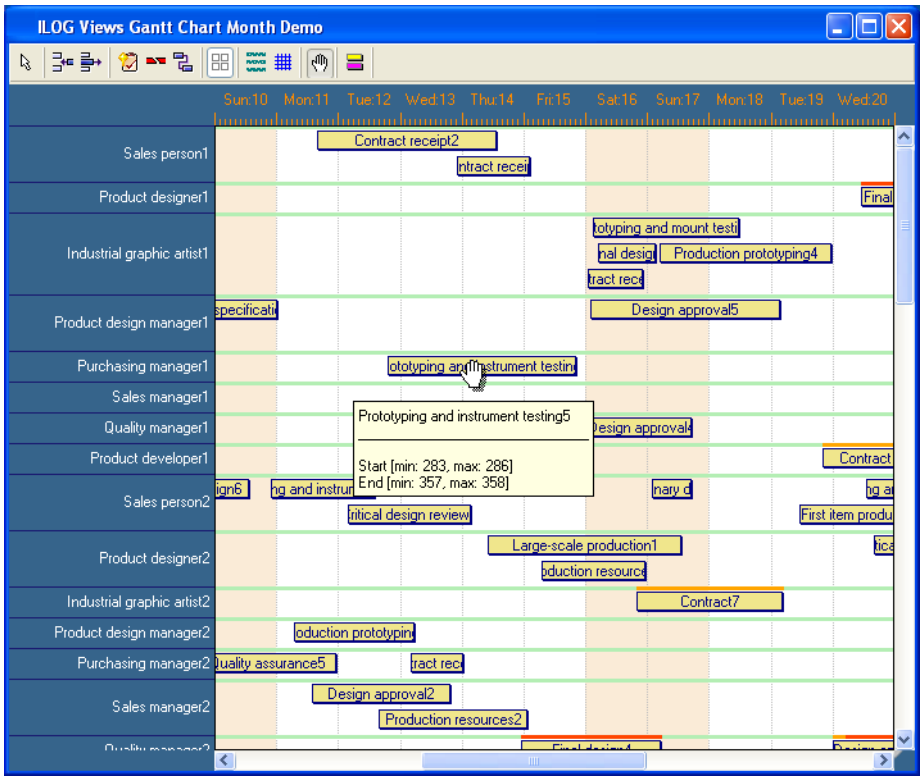

## The Load Sample

This sample shows how to customize the horizontal grid of the Gantt chart:



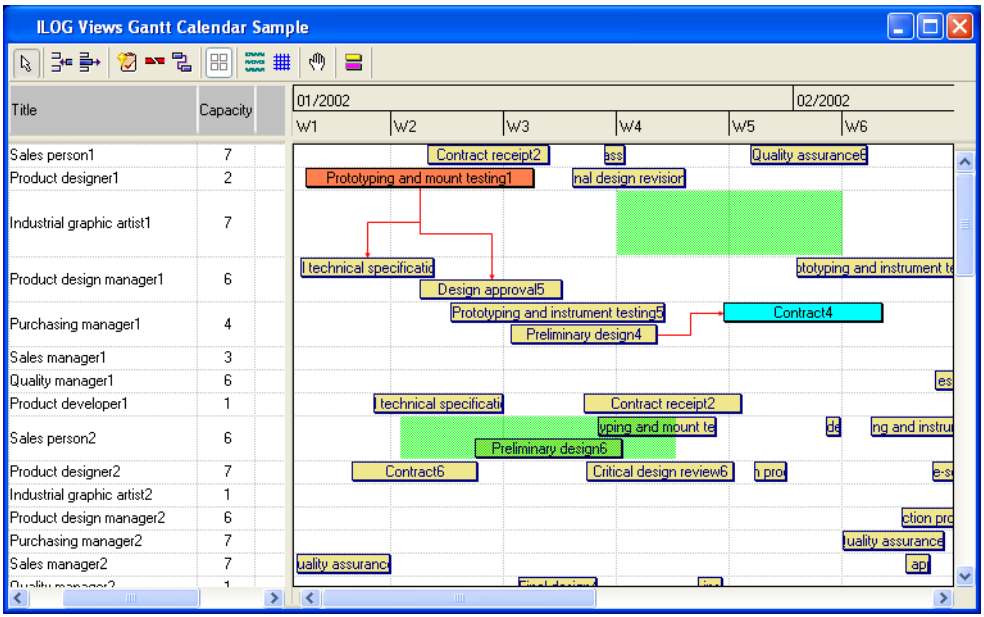
## The Month Sample

This sample shows how to customize the vertical grid of the Gantt chart and how to create a customized graphic model for Gantt links:




## The Calendar Sample

This sample shows how to use `IlvGanttChartForm`, `IlvCalendarScale`, and `IlvGanttCalendarGrid` for the Gantt chart.


ILOG Views Gantt Calendar Sample
Title
Capacity
01/2002
02/2002
W1
W2
W3
W4
W5
W6
Sales person1 7
Product designer1 2
Industrial graphic artist1 7
Product design manager1 6
Purchasing manager1 4
Sales manager1 3
Quality manager1 6
Product developer1 1
Sales person2 6
Product designer2 7
Industrial graphic artist2 1
Product design manager2 6
Purchasing manager2 7
Sales manager2 7
Contract receipt2
Quality assurance6
Prototyping and mount testing1
Design approval5
Prototyping and instrument testing5
Contract4
Preliminary design4
Contract receipt2
Preliminary design6
Contract6
Critical design review6

# Index

## M



## N



## R



## S

## T



## U