



Programmer's Manual



The contents of this manual and the associated KD Gantt software are the property of Klarälvdalens Datakonsult AB and are copyrighted. Any reproduction in whole or in part is strictly prohibited without prior written permission by Klarälvdalens Datakonsult AB.

KD Gantt and the KD Gantt logo are trademarks or registered trademarks of Klarälvdalens Datakonsult AB in the European Union, the United States, and/or other countries. Other product and company names and logos may be trademarks or registered trademarks of their respective companies.

▶ Table of Contents

1. Introduction	
What You Should Know	1
The Structure of This Manual	1
What's next	1
2. Anatomy of a Gantt Chart	
Definition of a Gantt Chart	3
Gantt Chart Examples	3
What's Next	5
3. Your First Own Gantt Chart	
Procedure	6
Examples	6
What's Next	10
4. Working With Items	
Introduction to Items	11
Customizing Items	13
What's next	22
5. Working With Task Links	
Introduction to Task Links	23
Examples	23
Customizing Task Links	26
What's Next	29
6. Working With the View	
Legends	30
Timeline	32
Menus And Dialogs	35
Drawing and Updating	37
Customizing	39
Drag-and-Drop	45
What's Next	45
7. Calendar Mode	
Introduction to Calendar mode	46
Procedure	47
Example	47



List of Figures

2.1. A Basic Gantt Chart	3
2.2. An Extended Gantt Chart	4
4.1. The Different Items	11
4.2. Customizing shapes	14
4.3. Customizing Colors	15
4.4. Customizing Highlight Colors	17
4.5. Customizing Start and End Times	18
4.6. Customizing Priority	19
4.7. Customizing Texts and Icons	21
4.8. An Empty Task As Line	21
5.1. A Simple Task Link	23
5.2. Linking More Than Two Tasks	25
5.3. A Highlighted Task Link Group	26
5.4. Task Link Color	27
5.5. Task Link Highlight Color	28
5.6. Task Link Text	28
6.1. Legend Items	31
6.2. Legend Header Widgets	31
6.3. Legends as Dock Windows	32
6.4. Different Timeline Scales (Minute to Month)	33
6.5. Customized Scale Count	34
6.6. Zooming Behaviour	35
6.7. The Timeline Context Menu	36
6.8. The Gantt View Context Menu	36
6.9. Item Properties Dialog	37
6.10. Brush Example	38
6.11. Updating Error When Using Brush	39
6.12. Default Horizontal Background Lines	40
6.13. Customized Horizontal Background Lines	41
6.14. A Customized Gantt View Background	42
6.15. Customized Widget Backgrounds	43
6.16. Customized Brushes	44
7.1. "Normal" mode	46
7.2. Calendar mode	46

Chapter 1. Introduction

KD Gantt is Klarälvdalens Datakonsult AB's package for creating Gantt charts in Qt applications. It features a large number of configuration options to tailor the Gantt charts to your needs. Since all configuration settings have reasonable defaults you can usually get by with setting only a handful of parameters and relying on the defaults for the rest.

This is the KD Gantt Programmer's Manual. It will get you started with creating your charts and provides lots of pointers to advanced features. Besides this manual, there are two other documents:

- Depending on your KD Gantt version, you will find different `INSTALL` files in the `doc` directory that explain how to install KD Gantt on your platform or how to build it from sources.
- KD Gantt comes with an extensive Reference Manual that is generated directly from the source code itself.

You should refer to these in conjunction with this Programmer's Manual.

► What You Should Know

You should be familiar with writing Qt applications, as well as have a working C++ knowledge. When you are in doubt about how a Qt class mentioned in this Programmer's Guide works, please check the Qt reference documentation or a good book about Qt.

► The Structure of This Manual

This manual starts with an introduction to Gantt charts in Chapter 2, *Anatomy of a Gantt Chart* where you will learn how a Gantt chart is constructed and what kind of parts it consists of. We will then have a closer look on how to create a first basic Gantt chart in KD Gantt, in Chapter 3, *Your First Own Gantt Chart*.

Following the introduction, we will dive deeper into the different parts of KD Gantt, separated into the three different chapters Chapter 4, *Working With Items*, Chapter 5, *Working With Task Links* and Chapter 6, *Working With the View*. These chapters explain how you can modify the different parts of KD Gantt and give some pointers on what you should think about.

As final chapter, we take a closer look on how you can use KD Gantt in Calendar mode; what this is, and what it can be used for. You will find this information in Chapter 7, *Calendar Mode*.

What's next

In the next chapter you will learn what parts a Gantt chart consists of...

Chapter 2. Anatomy of a Gantt Chart

In this chapter we will have a look at some different charts, displaying and explaining the different parts that a Gantt chart consists of.

► Definition of a Gantt Chart

A Gantt chart is a horizontal bar chart used to schedule and track different tasks in, for example, a software project.

It is constructed with a horizontal axis showing the timeline, that can be divided into several smaller parts, such as hours, days, weeks, etc. A Gantt chart also has a vertical axis containing the different tasks included in the project. These tasks can be subitems to other tasks or simply stand-alone tasks.

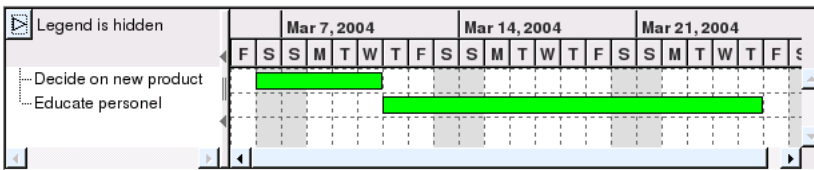
► Gantt Chart Examples

The following two Gantt charts show a few of the usage possibilities of KD Gantt. A more detailed description of the parts shown in these examples, can be found in the following chapters.

The complete source codes for these two examples will be shown in the next chapter, where we describe more closely how they work.

A Basic Gantt Chart

Figure 2.1. A Basic Gantt Chart



This is a screenshot from an application running KD Gantt. As you can see, the main widget is separated into two views—one view containing the tasks, and the other view containing the actual schedule with the timeline.

The left view in the screenshot above contains, as stated, all the scheduled tasks. All tasks have a name, type, priority and, depending on the type, a start date, a middle date and an end date. Some tasks, such as milestones, only have one date, because these are

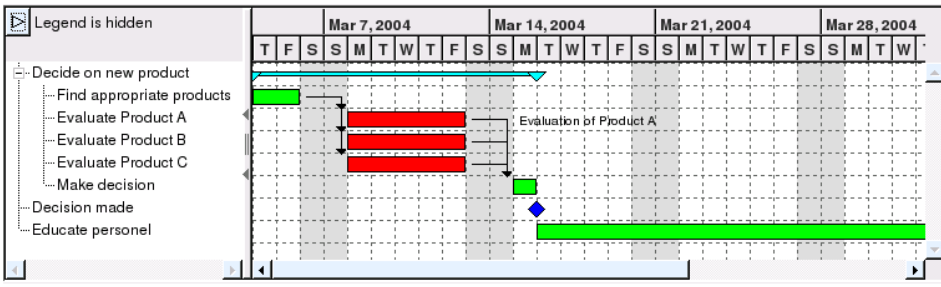
not actionable tasks, but rather a note on certain events, such as a delivery.

In the right view, we have the actual schedule—when the tasks are supposed to be worked on. In this simple chart, we only have two tasks which are performed sequentially.

A Gantt chart is often better the more verbose it is. In the following section, we will extend the simple chart we have seen so far to show more details about the performed tasks.

An Extended Gantt Chart

Figure 2.2. An Extended Gantt Chart



In the above screenshot, you can see examples of subtasks as well as task links. We have also changed the type of the first task to a `KDGanttViewSummaryItem`. Different item types are described further in Chapter 4, *Working With Items*.

The task links used in the Gantt chart above show dependencies between the tasks. You cannot evaluate the different products until you know which products to evaluate, and that is found out during the "Find appropriate products" task. You cannot decide on which product to use either until all alternatives are evaluated. The different usage alternatives for task links are shown in more detail in Chapter 5, *Working With Task Links* where we will even show a few code examples on how you use them.

We have added an event item as well, showing when the actual decision event should take place. This is an object of type `KDGanttViewItemEventItem`, with the start time set to the end time of the "Make decision"-task.

The reason for having a special color on three of the items is basically just for displaying that functionality. This feature can come in very handy when you might want to direct the viewer's focus to a certain item.

What's Next

In the next chapter we will have a look at how to create your first Gantt chart...

Chapter 3. Your First Own Gantt Chart

In this chapter we will have a look at how to create a first basic Gantt chart, first outlining the general procedure and later showing a code example.

► Procedure

The creation of a Gantt chart is a very easy and straightforward procedure:

1. Create an object of type `KDGanttView`. This is the actual Gantt view that you later will add different items to. When you want to modify the look of your Gantt chart and its items, you should modify this object, instead of modifying each item individually. This will result in a more uniform look of the Gantt view.
2. Create different objects of `KDGanttViewItem`. There are three different kinds to choose among: `KDGanttViewItemTaskItem`, `KDGanttViewItemSummaryItem`, and `KDGanttViewItemEventItem`. These three types are described in more detail in Chapter 4, *Working With Items*. You will need to modify the start and end times of these items, to have them displayed in the Gantt view. How to do this will be explained in the following examples.

► Examples

To make the discussion more practical, we will now show you a few code examples. The output of these examples is shown in Figure 2.1 and Figure 2.2.

A Basic Gantt Chart

The following code is also available in a complete example in `step01a.cpp`.

```
1
      /* -*- Mode: C++ -*-
      KDGantt
      */
5
      /*****
      ** Copyright (C) 2001-2003 Klar#lvdalens Datakonsult AB. All rights reserved.
      **
      ** This file is part of the KDGantt library.
10
      ** This file may be distributed and/or modified under the terms of the
      ** GNU General Public License version 2 as published by the Free Software
      ** Foundation and appearing in the file LICENSE.GPL included in the
      ** packaging of this file.
15
      ** Licensees holding valid commercial KDGantt licenses may use this file in
      ** accordance with the KDGantt Commercial License Agreement provided with
```

```

** the Software.
**
20 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
**
** See http://www.klaralvdalens-datakonsult.se/?page=products for
** information about KDGantt Commercial License Agreements.
25 **
** Contact info@klaralvdalens-datakonsult.se if any conditions of this
** licensing are not clear to you.
**
*****/
30
#include <qapplication.h>
#include <qmainwindow.h>
#include <qdatetime.h>

35 #include "KDGanttView.h"
#include "KDGanttViewItem.h"

int main( int argc, char*argv[] )
{
40   QApplication a( argc, argv );
   QMainWindow* mw = new QMainWindow();
   KDGanttView* gv = new KDGanttView( mw );
   mw->setCentralWidget( gv );
   gv->setScale( KDGanttView::Day ); ❶

45   // Specify when the project starts
   QDateTime projectStart( QDate( 2004, 03, 06 ) );

   // The first task in the project
50   KDGanttViewItem* decide =
       new KDGanttViewItem( gv, "Decide on new product" );❷
   decide->setStartTime( projectStart );
   // This task runs for 10 days
   decide->setEndTime( decide->startTime().addDays( 5 ) );

55   KDGanttViewItem* educate =
       new KDGanttViewItem( gv, decide, "Educate personel" );❸
   // This task starts when the decide-task finishes
   educate->setStartTime( decide->endTime() );
60   educate->setEndTime( educate->startTime().addDays( 15 ) );

   gv->show();
   gv->setListViewWidth( 300 );
   mw->setCaption( "KDGantt - Basic Gantt Chart" );
65   mw->resize( 400, 125 );
   mw->show();
   a.setMainWidget( mw );
   return a.exec();
70 }

```

- ❶ When displaying a Gantt chart with very long tasks, it might be a good idea to use a different scale than the default. The various scales available are Minute, Hour, Day, Week, Month and Auto. This modifies the lower scale in the Gantt chart, the upper scale is calculated automatically.
- ❷ Here we specify to which `KDGanttView` the item shall be added to, as well as the text to be displayed in the list view. A third argument, the name of the item, is also supported. This name can be used to identify this item later on.
- ❸ This is an alternative constructor used, which takes a `KDGanttViewItem` object as well as the previously mentioned arguments.

This makes sure that the created `KDGanttViewItem` is added below the passed `KDGanttViewItem`, and not in the top of the list view, which is default.

In this particular case, it would mean that the `educate` item will be put below the `decide` item.

An Extended Gantt Chart

The following example is a more verbose version of the Gantt chart above. It can also be found in `step01b.cpp`.

```
1      /* -*- Mode: C++ -*-
      KDGantt
*/
5
** Copyright (C) 2001-2003 Klar#lvdalens Datakonsult AB. All rights reserved.
**
** This file is part of the KDGantt library.
10 **
** This file may be distributed and/or modified under the terms of the
** GNU General Public License version 2 as published by the Free Software
** Foundation and appearing in the file LICENSE.GPL included in the
** packaging of this file.
15 **
** Licensees holding valid commercial KDGantt licenses may use this file in
** accordance with the KDGantt Commercial License Agreement provided with
** the Software.
20 ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
**
** See http://www.klaralvdalens-datakonsult.se/?page=products for
** information about KDGantt Commercial License Agreements.
25 **
** Contact info@klaralvdalens-datakonsult.se if any conditions of this
** licensing are not clear to you.
**
**
30 #include <qapplication.h>
#include <qmainwindow.h>
#include <qdatetime.h>
#include <qptrlist.h>
35 #include "KDGanttView.h"
#include "KDGanttViewSummaryItem.h"
#include "KDGanttViewTaskItem.h"
#include "KDGanttViewEventItem.h"
40 #include "KDGanttViewTaskLink.h"

int main( int argc, char* argv[] )
{
  QApplication a( argc, argv );
  QMainWindow* mw = new QMainWindow();
  KDGanttView* gv = new KDGanttView( mw );
  mw->setCentralWidget( gv );
  gv->setScale( KDGanttView::Day );
50 // Specify when the project starts
   QDateTime projectStart( QDate( 2004, 03, 04 ) );

   KDGanttViewSummaryItem* decide =
```

```

        new KDGanttViewSummaryItem( gv, "Decide on new product" );
55  decide->setStartTime( projectStart );
    // This tasks runs for 10 days
    decide->setEndTime( decide->startTime().addDays( 12 ) );

    KDGanttViewTaskItem* find =
60    new KDGanttViewTaskItem( decide, "Find appropriate products" );❶
    find->setStartTime( projectStart );
    find->setEndTime( find->startTime().addDays( 2 ) );

    // Evaluate products
65    KDGanttViewTaskItem* evaluateA =
        new KDGanttViewTaskItem( decide, find, "Evaluate Product A" );
    evaluateA->setStartTime( find->endTime().addDays( 2 ) );
    evaluateA->setEndTime( evaluateA->startTime().addDays( 5 ) );
    evaluateA->setText( "Evaluation of Product A" );❷
70    evaluateA->setColors( Qt:: red, Qt::black, Qt::white );❸

    KDGanttViewTaskItem* evaluateB =
        new KDGanttViewTaskItem( decide, evaluateA, "Evaluate Product B" );
75    evaluateB->setStartTime( find->endTime().addDays( 2 ) );
    evaluateB->setEndTime( evaluateB->startTime().addDays( 5 ) );
    evaluateB->setColors( Qt:: red, Qt::black, Qt::white );

    KDGanttViewTaskItem* evaluateC =
        new KDGanttViewTaskItem( decide, evaluateB, "Evaluate Product C" );
80    evaluateC->setStartTime( find->endTime().addDays( 2 ) );
    evaluateC->setEndTime( evaluateC->startTime().addDays( 5 ) );
    evaluateC->setColors( Qt:: red, Qt::black, Qt::white );

    KDGanttViewTaskItem* makeDecision =
85    new KDGanttViewTaskItem( decide, evaluateC, "Make decision" );
    makeDecision->setStartTime( evaluateC->endTime().addDays( 2 ) );
    makeDecision->setEndTime( makeDecision->startTime().addDays( 1 ) );

    // Have the item expanded from the start
90    decide->setOpen( true );

    QPtrList<KDGanttViewItem> findList;
    QPtrList<KDGanttViewItem> evaluateList;
    QPtrList<KDGanttViewItem> decisionList;
95    findList.append( find );
    evaluateList.append( evaluateA );
    evaluateList.append( evaluateB );
    evaluateList.append( evaluateC );
    decisionList.append( makeDecision );
100

    // Create the task links
    KDGanttViewTaskLink* findToEvaluate =
        new KDGanttViewTaskLink( findList, evaluateList );❹
105    KDGanttViewTaskLink* evaluateToDecision =
        new KDGanttViewTaskLink( evaluateList, decisionList );

    // Event where the decision should've been made
    KDGanttViewItem* decisionMade =
        new KDGanttViewItem( gv, decide, "Decision made" );
110    decisionMade->setStartTime( makeDecision->endTime() );

    KDGanttViewTaskItem* educate =
        new KDGanttViewTaskItem( gv, decisionMade, "Educate personel" );
    // This task starts when the decide-task finishes
115    educate->setStartTime( decide->endTime() );
    educate->setEndTime( educate->startTime().addDays( 30 ) );

    gv->show();
    gv->setListViewWidth( 300 );
120    mw->setCaption( "KDGantt - Extended Gantt Chart" );
    mw->resize( 600, 300 );
    mw->show();
    a.setMainWidget( mw );

```

```
        return a.exec();
125 }
```

- ❶ Instead of specifying the `KDGanttView` to add the item to, we specify in this constructor which `KDGanttViewItem` should be the parent of this item. Just as in item 2 in the previous example, this constructor can take a third argument, the internal name of the item.
- ❷ The text specified here is normally displayed after the task. If the item is a `KDGanttViewItemTaskItem`, and `displaySubItemsAsGroup()` returns `true`, the text is displayed inside the item, and will be truncated automatically in case it does not fit inside.
- ❸ What we do here is to change the color of this particular item. As you can see in the screenshot, the item was purely red, without any black or white in it. The reason for this is that the first argument affects the color of an item only when the item is a task or an event. More on this in the next chapter.
- ❹ Here we create a task link between the tasks in `findList` and `evaluateList`. For more on task links, see Chapter 5, *Working With Task Links*.

What's Next

In the next chapter, we will have a closer look at the different items that can be used in a Gantt chart, and also at how to modify these items in various ways...

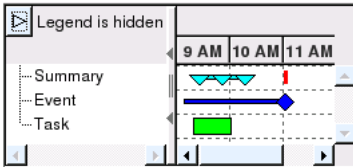
Chapter 4. Working With Items

In this chapter, we will have a closer look at the different kind of items that can be used in a chart, as well as on some ways to modify these items.

► Introduction to Items

There are three different items available in KD Gantt: Task, Event and Summary items, represented by the three classes `KDGanttViewTaskItem`, `KDGanttViewEventItem`, and `KDGanttViewSummaryItem`. You can see a typical example of item usage in Figure 4.1.

Figure 4.1. The Different Items



There are several different constructors you can use when you create the different item objects. The ability to specify the text that will appear in the list view, as well as the internal name of item item is available in all constructors, but the first arguments differ. Either, you pass the `KDGanttView` that the item should be added to to the constructor, or you specify a `KDGanttViewItem` that will be the parent of the new item. Besides these two alternatives, you can also specify after which other `KDGanttViewItem` the new item should appear. This is used in the extended Gantt chart in the previous chapter in order to get the items in the correct order.

When adding several items at once, which could e.g. be the case when a file is loaded, redrawing the Gantt view after every addition might become very slow. To avoid this, you can use `KDGanttView::setUpdateEnabled(false)`, which will cause the update of the Gantt view to be suspended until you set the value to `true` again. You can find a more detailed explanation of this, as well as other updating modes, in Chapter 6, *Working With the View*.

Tasks

The Task item has two purposes. It is used both as a regular schedule item, as well as a Calendar object. Switching between these two modes, is done by passing `true` to `KDGanttViewItem::setDisplaySubitemsAsGroup()`, if you want the item to be a Calendar object, or `false` if you want the item to be a regular schedule item. Read more about Calendar mode in Chapter 7, *Calendar Mode* as well as in the section Dis-

play Mode further down in this chapter.

For a regular task item, you can specify a start and end time, which will be drawn accordingly in the Gantt view. If the time is not specified, or if the start time equals the end time, a vertical line is drawn in the Gantt view to show that there is no time interval set.

If the item is a Calendar object, its children will affect the start/end time of the item and will not need to be set.

Task items have no shapes. The item itself is a rectangle stretching from the start time to the end time. Due to this, trying to change the shapes of a task item will have no effect at all. The lack of shapes also affects changing colors of the item. A call to `KDGanttViewItem::setColors()` sets the color of the item to the color in the first argument.

Events

Events are used to schedule certain events in, for example, a software project. You can specify a lead time and a start time for this item. If lead time is set, there will be a lead time line drawn from that time to the start time, where the actual event shape is drawn. If the lead time is greater than the start time, or is not set, no lead time line will be drawn.

It is possible to change both the shape and the color of an event item. This is done by the regular calls to `KDGanttViewItem::setColors()` and `KDGanttViewItem::setShapes()`, but in both cases only the first argument is used. This is because an event item does not have a start, middle, and end shape, but only a start shape instead.

Summaries

Summary items can be used for scheduling different tasks just as regular task items, but they can also be used to summarize several tasks into one, and having the total time calculated from its children's start/end times. This is done by calling `KDGanttViewItem::setDisplaySubitemsAsGroup()`, and give it `true` as the first and only argument. When you later add task items as children to this summary item, the start and end time of this item will be calculated automatically.

When using the Summary item as a regular item, you can specify start, middle and end time. If it is used to summarize other items, you can only set the middle and end time. The start time is set, based on the start time of the first child. You can also set the actual end time for a summary. This is drawn as a vertical line in the Gantt view.

Summary is the only item type where you are able to use all three arguments in `KDGanttViewItem::setColors()` and `KDGanttViewItem::setShapes()`. For a closer explanation of these two methods read the section Shapes and Colors further ahead in this manual.

► Customizing Items

There are several ways of customizing the items in KD Gantt. In this section, we will try to demonstrate the majority of them, and also give some pointers to what to remember when using them, as well as some code examples.

Shapes

An item in the Gantt view can have different shapes. For an item of type Summary, it is possible to change the start, middle and end shapes which by default is a triangle pointing down. An item of type Event only has one shape, treated as the start shape, which by default is a diamond. The method call looks the same no matter which item type you are changing, but for events, only the first argument is used.



Note

Only items of type Summary and Event can have their shapes changed.

There are two different ways of changing the shape of an item. The recommended way is to use `KDGanttView::setShapes()`, and give it the type to change, followed by the start, middle, and end shapes. This results in a uniform Gantt view, where all items of a certain type look the same.

The other way is to use `KDGanttViewItem::setShapes()`, which lets you modify the shape of a certain item object. Just specify which shapes to use in the start, middle and end. This way, you can have items of the same type look differently, which might be useful if you want to separate one item from the rest.

Have a look at the following code example to get an idea on how to use these methods. This example code is an excerpt from `step02a.cpp`. The result is shown in Figure 4.2.

```
// 3 summaries and 1 event created

gv->setShapes( KDGanttViewItem::Summary, ❶
              KDGanttViewItem::TriangleDown,
              KDGanttViewItem::Circle,
              KDGanttViewItem::TriangleUp );

gv->setShapes( KDGanttViewItem::Event, ❷
              KDGanttViewItem::Circle,
              KDGanttViewItem::Square,
              KDGanttViewItem::Square );

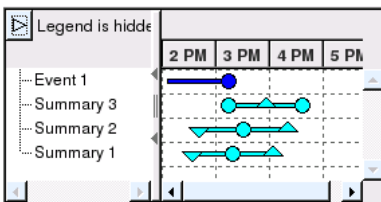
summary3->setShapes( KDGanttViewItem::Circle, ❸
                    KDGanttViewItem::TriangleUp,
                    KDGanttViewItem::Circle );
```

❶ The call to `KDGanttView::setShapes()` affects all items of a certain type, even

those created before the call. This means that Summary 1, Summary 2 and Summary 3 has their shapes changed, even though they are created earlier.

- ② As stated in the previous paragraph, this call affects all items of a certain type. But since the type specified is event, the third and fourth argument will not be used. The shape of the event item will therefore be a circle, and nothing else.
- ③ This call only changes the shapes of Summary 3, which might be needed sometimes, to separate this particular item from other items of the same type. Keep in mind that calling `KDGanttView::setShapes()` after `KDGanttViewItem::setShapes()` will change the shapes of the item again, if that item type is modified.

Figure 4.2. Customizing shapes



Colors

Different types of items have different default colors. The initial default color is set to blue for event items, green for tasks and cyan for summaries.

As for shapes, there are two different ways of setting the default color of an item. The recommended way is to use `KDGanttView::setDefaultColor()`, give it the type to change and the color to use. This way, all items of the same type get the same colors, which leads to a more uniform Gantt view.

The other way is to use `KDGanttViewItem::setDefaultColor()`, and pass it the color to use. This way, you can get different colors on all the items, independent of which type they are.

You can also change the colors of the start, middle and end shapes in two different ways. `KDGanttView::setColors()` should be used to get a uniform look of the Gantt view. It requires the type to change, as well as the start, middle, and end colors. You can also change a single item by using `KDGanttViewItem::setColors()` and pass the start, middle, and end colors to use to it.

This code example comes from `step02b.cpp`, and the result is shown in Figure 4.3.

```
// summary1 created
// task1 created

// Change the default color of type Summary to yellow
gv->setDefaultColor( KDGanttViewItem::Summary, Qt::yellow );❶
```

```

// This Summary will be yellow
KDGanttViewSummaryItem* summary2 =
    new KDGanttViewSummaryItem( gv, "Summary 2" );

// Change the color only of summary1
summary1->setColors( Qt::black, Qt::red, Qt::gray );❷

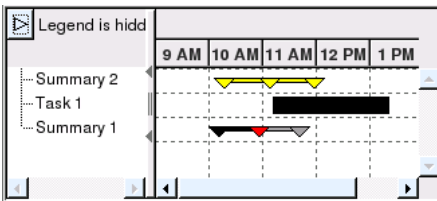
// Change the color of all tasks
gv->setColors( KDGanttViewItem::Task, Qt::black, Qt::red, Qt::gray );❸

```

- ❶ Using `KDGanttView::setDefaultColor()` will not change the colors of already created items. In the example above, Summary 1 will still be in cyan after the change of the default color for summaries. Only those items created after the default color change are affected.
- ❷ The call to `KDGanttViewItem::setColors()` changes the colors of the shapes of the item. The first color (black) stretches from the first shape up to the middle shape. The second color (red) only affects the middle shape, and the third color (gray) stretches from the middle shape to the end of the item.
- ❸ Here, we use `KDGanttView::setColors()` to change the color of all items of a certain type, in this case all tasks.

Since task items do not have shapes, only the first color argument will be used; thus, the task will be black in the Gantt view. Items of the type Event have the same behavior in this context.

Figure 4.3. Customizing Colors



Highlight Colors

When an item is clicked, it is highlighted. The default color the item is displayed in when highlighted is red, but that can be changed in two different ways.



Note

It is not possible to configure highlighting in the Gantt chart by default. You, the programmer, need to implement this yourself. One way to do this is displayed as a code example further down in this section.

To get a uniform Gantt view, you should use `KDGant-`

`tView::setDefaultHighlightColor()`, and pass it which type to change and the color to use. You can also change the highlight color of a single item using `KDGanttViewItem::setDefaultHighlightColor()`, and pass it which color to use.

It is also possible to change the highlight colors of the start, middle and end shapes. This is done using `KDGanttView::setHighlightColors()` to get a uniform Gantt view, but it is also possible to use `KDGanttViewItem::setHighlightColors()`, which lets you modify a single item.

For a closer explanation on when and how to use the methods used to modify a single item, please see the Reference Manual.

As mentioned earlier, you, the programmer, need to implement the highlighting by yourself. This example shows an easy way of doing this.

First, we have the header file, `step02c_slot.h`.

```
#include "KDGanttViewItem.h"
#include "KDGanttView.h"

class GanttSlot:public QObject
{
    Q_OBJECT
public:

public slots:
    void KDGanttViewItemClicked(KDGanttViewItem*);
};
```

Nothing remarkable done here. Next follows the matching implementation file: `step02c_slot.cpp`.

```
#include "step02c_slot.h"

extern KDGanttView* gv ;

void GanttSlot::KDGanttViewItemClicked(KDGanttViewItem* i)
{
    static KDGanttViewItem* lastClickedItem = 0;

    if ( lastClickedItem->listView() )
        if ( lastClickedItem && lastClickedItem != i )
            lastClickedItem->setHighlight( false );
    if ( i )
        i->setHighlight(!i->highlight());
    lastClickedItem = i;
}
```

What this does is basically de-highlighting any item that was highlighted, and then toggling highlighting on the clicked item.

Now, to get this highlighting to work with our program, we need to connect the correct signals to this slot. This, along with changing the highlight colors, is done in the following example, which can also be found as a runnable example in `step02c.cpp`. The result is shown in Figure 4.4.

```

// Change the default highlight color of Tasks to yellow
gv->setDefaultHighlightColor( KDGanttViewItem::Task, Qt::yellow );

KDGanttViewSummaryItem* summary1 =
    new KDGanttViewSummaryItem( gv, "Summary 1" );

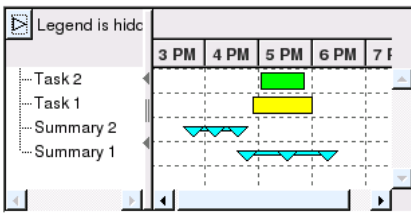
// Change the shape highlight colors of a single
// summary item to black, green, red
summary1->setHighlightColors( Qt::black, Qt::green, Qt::red );❶

// Connect click on Gantt view
GanttSlot* ganttSlot = new GanttSlot();
QObject::connect( gv, SIGNAL( lvCurrentChanged( KDGanttViewItem* ) ),
                 ganttSlot, SLOT( KDGanttViewItemClicked( KDGanttViewItem* ) ) );
QObject::connect( gv, SIGNAL( gvCurrentChanged( KDGanttViewItem* ) ),
                 ganttSlot, SLOT( KDGanttViewItemClicked( KDGanttViewItem* ) ) );

```

- ❶ `KDGanttViewItem::setHighlightColors` has the same behavior as `KDGanttViewItem::setColors` when it comes to argument handling. If the item is a summary, all three arguments will be used, but if the item is an event or a task only the first argument will be used.

Figure 4.4. Customizing Highlight Colors



Start and End Time

When a new item is created, its start and end time are set automatically to the current time.

To change the start and end time of an item, you use `KDGanttViewItem::setStartTime()` and `KDGanttViewItem::setEndTime()`. There are a few differences depending on what type of item you are working with.

1. When the item is an Event item, only `KDGanttViewItem::setStartTime()` is available. There is also an additional method called `KDGanttViewItemEventItem::setLeadTime()`.

The Event item is drawn at the start time, and if the lead time is specified and less than the start time, a line is drawn from the lead time to the start time.

- When the item is a Summary, you have two additional methods, besides the two mentioned above: `KDGanttViewSummaryItem::setMiddleTime()` and `KDGanttViewSummaryItem::setActualEndTime()`.

The item is drawn between the start time and end time, with an additional shape located at the middle time. A red line is also drawn in the Gantt chart, to show the actual end time of the summary. This line is not drawn unless the actual end time is specified.

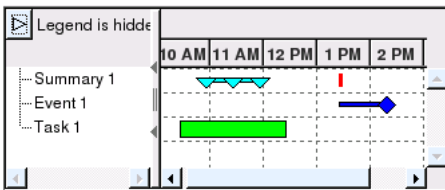
The complete source code for this example is in `step02d.cpp`.

```
KDGanttViewTaskItem* task1 =
    new KDGanttViewTaskItem( gv, "Task 1" );
task1->setStartTime( QDateTime::currentDateTime() );
task1->setEndTime( QDateTime::currentDateTime().addSecs( 7200 ) );

KDGanttViewEventItem* event1 =
    new KDGanttViewEventItem( gv, "Event 1" );
event1->setLeadTime( QDateTime::currentDateTime().addSecs( 10800 ) );
event1->setStartTime( QDateTime::currentDateTime().addSecs( 14000 ) );

KDGanttViewSummaryItem* summary1 =
    new KDGanttViewSummaryItem( gv, "Summary 1" );
summary1->setStartTime( QDateTime::currentDateTime().addSecs( 1800 ) );
summary1->setMiddleTime( QDateTime::currentDateTime().addSecs( 3600 ) );
summary1->setEndTime( QDateTime::currentDateTime().addSecs( 5400 ) );
summary1->setActualEndTime( QDateTime::currentDateTime().addSecs( 10800 ) );
```

Figure 4.5. Customizing Start and End Times



Priority

Every item in the Gantt chart has a priority between 1 and 199. On creation time, every item has a default priority of 150. To change the priority of an item, you should use `KDGanttViewItem::setPriority()`.

Depending on the priority, there are a few behavior differences in KD Gantt:

- If the priority of an item is below 100, that item is drawn below the grid in the Gantt view.

2. When an item is set to display its subitems as a group, the subitems with the highest priorities are painted on top of the others. This is shown in the screenshot below. For more information about showing subitems as a group, including Calendar mode, please see Chapter 7, *Calendar Mode*, as well as the next section.

The following code example comes from `step02e.cpp`, and the result is shown in Figure 4.6.

```
KDGanttViewItem* task1 =
    new KDGanttViewItem( gv, "Task 1 - Priority 30" );
task1->setStartTime( QDateTime::currentDateTime() );
task1->setEndTime( QDateTime::currentDateTime().addSecs( 7200 ) );
task1->setPriority( 30 );❶

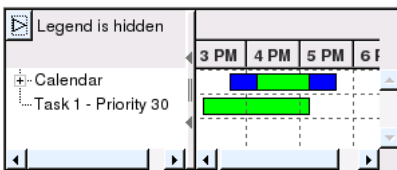
KDGanttViewItem* calendar =
    new KDGanttViewItem( gv, "Calendar" );
calendar->setDisplaySubitemsAsGroup( true );❷

KDGanttViewItem* task2 =
    new KDGanttViewItem( calendar, "Task 2 - Priority 100" );❸
task2->setStartTime( QDateTime::currentDateTime().addSecs( 1800 ) );
task2->setEndTime( QDateTime::currentDateTime().addSecs( 9000 ) );
// The color change is here to distinguish between task2 and task3
task2->setColors( Qt::blue, Qt::blue, Qt::blue );
task2->setPriority( 100 );

KDGanttViewItem* task3 =
    new KDGanttViewItem( calendar, "Task 3 - Priority 125" );❸
task3->setStartTime( QDateTime::currentDateTime().addSecs( 3600 ) );
task3->setEndTime( QDateTime::currentDateTime().addSecs( 7200 ) );
task3->setPriority( 125 );
```

- ❶ Here, we set the priority of task to a value less than 100, which means that the task will be drawn below the grid in the Gantt chart.
- ❷ We are adding both `task2` and `task3` as children to `calendar`, and call `KDGanttViewItem::setDisplaySubitemsAsGroup(true)`, which results in both of the tasks being drawn on the timeline of `calendar`. And since the priority of `task3` is higher than the priority of `task2`, `task3` is drawn on top of `task2`, which can be seen in the screenshot below.

Figure 4.6. Customizing Priority



Display Mode

There are two different display modes—the "normal" mode, and the calendar mode, which is described in more detail in Chapter 7, *Calendar Mode*.

In normal mode, the item is displayed as usual. It is drawn according to its start and end times. This is true no matter whether the item has children or not, and no matter whether the item is opened or closed.

In calendar mode, an item is displayed in the regular way if it does not have any children. If the item has children, those will be drawn on the timeline of the item, hiding the item itself. We have seen this in the previous section, where `task2` and `task3` were drawn on the `calendar`'s timeline.

To change display mode, use `KDGanttView::setCalendarMode()`, and give it either `true` (Calendar mode) or `false` (normal mode). The behavior of the items is changed with `KDGanttViewItem::setDisplaySubitemsAsGroup()`, where `true` means that it should behave like a calendar item, and `false` means that it should behave like a normal item.

```
// Comment the following line to be able to view the calendar item's children
gv->setCalendarMode( true );

KDGanttViewSummaryItem* calendar =
    new KDGanttViewSummaryItem( gv, "Calendar" );
calendar->setDisplaySubitemsAsGroup( true );

KDGanttViewTaskItem* task1 =
    new KDGanttViewTaskItem( calendar, "Task 1" );
task1->setStartTime( QDateTime::currentDateTime().addSecs( 1800 ) );
task1->setEndTime( QDateTime::currentDateTime().addSecs( 3600 ) );
task1->setDisplaySubitemsAsGroup( false );

KDGanttViewTaskItem* task2 =
    new KDGanttViewTaskItem( calendar, "Task 2" );
task2->setStartTime( QDateTime::currentDateTime().addSecs( 5400 ) );
task2->setEndTime( QDateTime::currentDateTime().addSecs( 9000 ) );
task1->setDisplaySubitemsAsGroup( false );
```

Text and icons

There are several locations in the Gantt chart where you can have text displayed for an item. These locations include: the list view, in the actual chart, as tooltip text and as "What's This" text.

You have also possibilities to change the font and color of the text displayed, although these changes only affects the text displayed inside the Gantt chart. This is done by calling `KDGanttView::setTextColor()` and `KDGanttView::setFont()` to get a uniform Gantt view, but it's also possible to change the font and text color of just one item, using `KDGanttViewItem::setTextColor()` and `KDGanttViewItem::setFont()`. When changing the font of an item, only the actual font is changed, and not the size of the text.

The text displayed in the Gantt view, and set with `KDGanttViewItem::setText()`

has two different behaviors. If the display mode is Calendar mode, the text for each item will be displayed inside the item, and will be truncated if needed. If the display mode is normal mode, the text will be displayed after the item, as can be seen in the screenshot below.



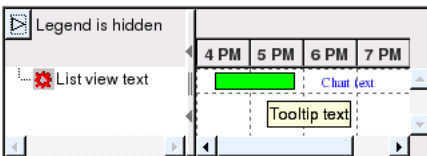
Note

The list view text, tooltip text and "What's This" text is the same as long as you do not change any of them, namely the text you specify in the constructor.

You can add an icon to a task in KD Gantt. That icon is then displayed next to the text in the list view. The icon is set by using `KDGanttViewItem::setPixmap()`; either pass only the pixmap to use, or both the column to put the icon in and the pixmap to use.

```
// ... task1 created ...  
task1->setPixmap( QPixmap( "Machine.bmp" ) );  
  
task1->setFont( QFont( "Serif" , 14 ) );  
task1->setTextColor( Qt::blue );  
task1->setText( "Chart text" );  
task1->setListViewText( "List view text" );  
task1->setTooltipText( "Tooltip text" );  
task1->setWhatsThisText( "Whats This text" );
```

Figure 4.7. Customizing Texts and Icons

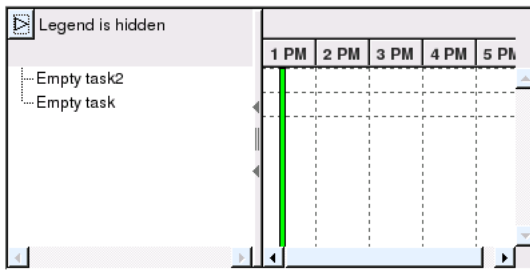


Empty Tasks

If you want to, you can have all your empty tasks in the Gantt view displayed as a vertical line. To achieve this, simply call `KDGanttView::setDisplayEmptyTasksAsLine()`, and pass true to enable this functionality or false to disable it, which is the default at startup.

When this functionality is enabled and there is a task in the Gantt chart which is empty (no times set), a vertical line will be drawn in the Gantt view. This line is drawn on the current time.

Figure 4.8. An Empty Task As Line



▶ **What's next**

In the next chapter, we will have a look at how you can link tasks together to show dependencies in your Gantt chart...

Chapter 5. Working With Task Links

Task links are most often used to show dependencies between tasks. We saw an example of this in the introductory chapters, in Figure 2.2. Using links makes it easier for the reader to see quickly which tasks need to be completed before the next task can be started.

In this chapter, we will try to explain this a bit further, by means of a few code examples on how you can use task links, and also cover the customization possibilities available in KD Gantt.

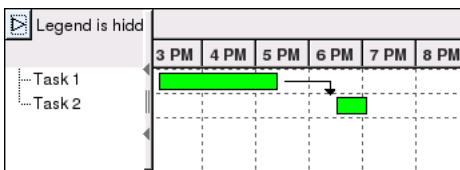
► Introduction to Task Links

A task link is an object of type `KDGanttViewTaskLink`, containing references to which the tasks that are linked together. A `KDGanttViewTaskLink` can link several tasks to several other tasks, several tasks to one task, or just one task to several other tasks.

You can also group several task links together, to manipulate them uniformly. This is done with the class `KDGanttViewTaskLinkGroup`.

The examples further down show links between objects of the type `KDGanttViewTaskItem`, but it works just the same with objects of the types `KDGanttViewItem` and `KDGanttViewSummaryItem`.

Figure 5.1. A Simple Task Link



► Examples

To show a practical example of the different usages of `KDGanttViewTaskLink` and `KDGanttViewTaskLinkGroup`, here are a few code examples.

A Simple Task Link

Linking two tasks together is very easy and straight-forward. The only thing you need to

do, is to tell the `KDGanttViewTaskLink` constructor which tasks to link. A complete version of this example can be found in `step03a.cpp`. The result of this code example is shown in Figure 5.1.

```
// A simple task link
KDGanttViewItem* task2 =
    new KDGanttViewItem( gv, "Task 2" );
task2->setStartTime( QDateTime::currentDateTime().addSecs( 12000 ) );
task2->setEndTime( QDateTime::currentDateTime().addSecs( 14000 ) );

KDGanttViewItem* task1 =
    new KDGanttViewItem( gv, "Task 1" );
task1->setStartTime( QDateTime::currentDateTime().addSecs( 0 ) );
task1->setEndTime( QDateTime::currentDateTime().addSecs( 8000 ) );

KDGanttViewTaskLink* taskLink = new KDGanttViewTaskLink( task1, task2 );❶
```

- ❶ Here, we use one of the available constructors for `KDGanttViewTaskLink`, which in this case takes two tasks and links them together. You do not need to do anything else to have the link shown in the Gantt view.

Several Tasks Dependent On One Task

This example shows how to link several tasks together with one task. This is done by using a `QPtrList` filled with `KDGanttViewItem` objects. As usual, there is a complete source code example available in `step03b.cpp`, and the result is displayed in Figure 5.2.

```
// Several tasks dependent on one task
KDGanttViewItem* task3 =
    new KDGanttViewItem( gv, "Task 3" );
task3->setStartTime( QDateTime::currentDateTime().addSecs( 5000 ) );
task3->setEndTime( QDateTime::currentDateTime().addSecs( 10000 ) );

KDGanttViewItem* task2 =
    new KDGanttViewItem( gv, "Task 2" );
task2->setStartTime( QDateTime::currentDateTime().addSecs( 5500 ) );
task2->setEndTime( QDateTime::currentDateTime().addSecs( 14000 ) );

KDGanttViewItem* task1 =
    new KDGanttViewItem( gv, "Task 1" );
task1->setEndTime( QDateTime::currentDateTime().addSecs( 3600 ) );

QPtrList<KDGanttViewItem> from;❶
QPtrList<KDGanttViewItem> to;❶

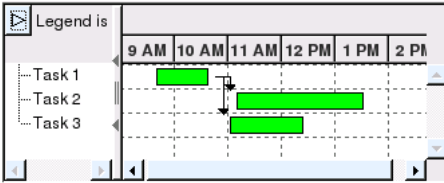
from.append( task1 );
to.append( task2 );
to.append( task3 );

KDGanttViewTaskLink* taskLink = new KDGanttViewTaskLink( from, to );❶
```

- ❶ When you need to link more than two tasks together, you will need to use `QPtrList`, as mentioned above. By adding different amounts of items to the two lists,

you can get different combinations of links, such as one-to-one, one-to-many (as in the code example), many-to-one or many-to-many.

Figure 5.2. Linking More Than Two Tasks



Grouping Task Links

To uniformly modify several task links at once, you can use `KDGanttViewTaskLinkGroup`, and add several `KDGanttViewTaskLink` objects to it. You can also specify in the constructor which group a certain `KDGanttViewTaskLink` should belong to. A complete code example for this can be found in `step03c.cpp`.

```
// ... Creating task1 and task2 ...
KDGanttViewTaskLink* taskLink1 = new KDGanttViewTaskLink( task1, task2 );

// ... Creating task3, task4 and task5 ...
QPtrList<KDGanttViewItem> from1;
QPtrList<KDGanttViewItem> to1;

from1.append( task3 );
to1.append( task4 );
to1.append( task5 );

KDGanttViewTaskLink* taskLink2 = new KDGanttViewTaskLink( from1, to1 );

// Grouping taskLink1 and taskLink2
KDGanttViewTaskLinkGroup* group = new KDGanttViewTaskLinkGroup;
group->insert( taskLink1 );❶
group->insert( taskLink2 );

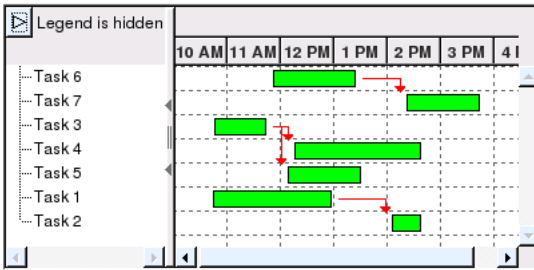
// ... Creating task6 and task7 ...

// taskLink3 now belongs to group
KDGanttViewTaskLink* taskLink3 =
    new KDGanttViewTaskLink( group, task6, task7 );❷

// Highlight the group
group->setHighlight( true );
```

- ❶ Here we show two of the three different ways of assigning a task link to a task link group. The third way is to use `KDGanttViewTaskLink::setGroup()` and pass it group the task link should belong to.

Figure 5.3. A Highlighted Task Link Group



► Customizing Task Links

When you have added several task links to your Gantt chart, you may want to customize them in some way.

Color

Changing the color of a task link is done by calling `KDGanttViewTaskLink::setColor()`, passing the color to use.

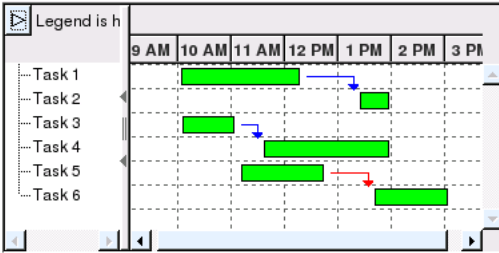
You also have the possibility to change the color of several task links at once, by adding them to a `KDGanttViewTaskLinkGroup` object, and then using `KDGanttViewTaskLinkGroup::setColor()`.

```
// ... Creating task1 and task2 ...
KDGanttViewTaskLink* taskLink1 = new KDGanttViewTaskLink( task1, task2 );
// ... Creating task3 and task4 ...
KDGanttViewTaskLink* taskLink2 = new KDGanttViewTaskLink( task3, task4 );
// ... Creating task5 and task6 ...
KDGanttViewTaskLink* taskLink3 = new KDGanttViewTaskLink( task5, task6 );

// Grouping taskLink1 and taskLink2
KDGanttViewTaskLinkGroup* group = new KDGanttViewTaskLinkGroup;
group->insert( taskLink1 );
group->insert( taskLink2 );

group->setColor( Qt::blue );
taskLink3->setColor( Qt::red );
```

Figure 5.4. Task Link Color



Highlight Color

When a user clicks on a task link, you may want to display that task link in another color than all the other task links. This is done by highlighting the task link. The highlighting color is red by default, but this can be changed by calling `KDGanttViewTaskLink::setHighlightColor()`.

You can also set the highlight color of a task link group, by using `KDGanttViewTaskLinkGroup::setHighlightColor()`.



Note

You, the programmer, need to implement the slot that is highlighting the task link, or task link group. An example of how this can be done is shown below, and is also available in `step03e.cpp`.

```
// ... task1 and task2 created ...
KDGanttViewTaskLink* taskLink1 = new KDGanttViewTaskLink( task1, task2 );

// ... task3 and task4 created ...
KDGanttViewTaskLink* taskLink2 = new KDGanttViewTaskLink( task3, task4 );

// Connect click on Gantt view
GanttSlot* ganttSlot = new GanttSlot();
QObject::connect( gv,
                  SIGNAL( taskLinkLeftClicked( KDGanttViewTaskLink * ) ),
                  ganttSlot,
                  SLOT( KDGanttViewTaskLinkLeftClicked( KDGanttViewTaskLink* ) )
                  );

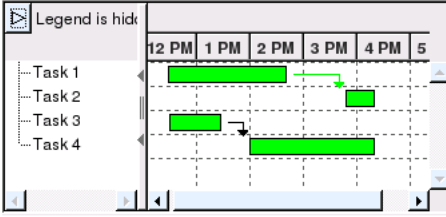
taskLink1->setHighlightColor( Qt::green );
```

❶ The slot we connect to here is implemented in `step03e_slot.cpp`.

❷ Here we change the highlight color of a task link to green. The other task link

taskLink2) still has red as its highlighting color.

Figure 5.5. Task Link Highlight Color



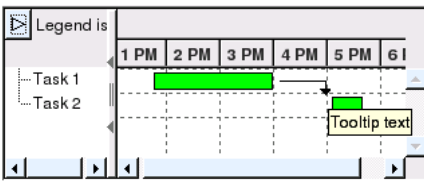
Text

There are a few places in the Gantt view where the text of a task link is displayed, including the tooltip text and the "What's This?" text. With the other customization possibilities, including visibility, it was possible to modify an entire task link group. This is not the case here. You cannot specify a certain tooltip text or "What's This?" text for a whole task link group—that would simply make no sense.

```
// ... Created task1 and task2 ...
KDGanttViewTaskLink* taskLink1 = new KDGanttViewTaskLink( task1, task2 );

taskLink1->setTooltipText( "Tooltip text" );
taskLink1->setWhatsThisText( "What's This? text" );
```

Figure 5.6. Task Link Text



Visibility

If, for some reason, you want to change the visibility of a task link, or task link group, that can be achieved quite easily by calling either `KDGanttViewTaskLink::setVisible()` or `KDGanttViewTaskLinkGroup::setVisible()`, passing either `true` or `false`, depending on what kind of behavior you want.

```
// ... created task1 and task2 ...  
KDGanttViewTaskLink* taskLink1 = new KDGanttViewTaskLink( task1, task2 );  
taskLink1->setVisible( false );
```

What's Next

In the next chapter we will have a look at what you can do with the Gantt view, and the dialogs available in KD Gantt...

Chapter 6. Working With the View

So far, we have discussed and given examples on what you can do with the tasks and task links. In this chapter, we will discuss the very foundation of the Gantt chart—the actual view.

► Legends

You have probably noticed the short phrase "Legend is hidden" in the top-left corner of the widget in virtually every screenshot in this manual. The legend is simply a small part of KD Gantt containing short information about what is shown in the Gantt chart itself.

There are several ways to modify the appearance and content of the legend. We will discuss some of them here and add the remaining ones later, in the section Customizing Colors section.

Legend Items

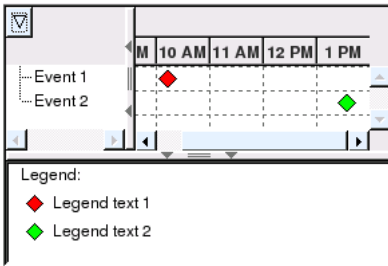
Adding an item to a legend is a very simple process. You just use `KDGanttView::addLegendItem()` and pass it the shape to add, its color, and the legend text to be displayed next to the shape.

The complete source code of this example is available in `step04a.cpp`. The result is shown in Figure 6.1.

```
// ... Created event1 and event2 ...  
event1->setColors( Qt::red, Qt::red, Qt::red );  
event2->setColors( Qt::green, Qt::green, Qt::green );  
gv->addLegendItem( KDGanttViewItem::Diamond, Qt::red, "Legend text 1" );❶  
gv->addLegendItem( KDGanttViewItem::Diamond, Qt::green, "Legend text 2" );
```

- ❶ Here, we add an item in the shape of a diamond to the legend, which is the same shape `KDGanttViewItem` uses in the Gantt view. The color is red, and the text "Legend text 1" is displayed next to the item.

Figure 6.1. Legend Items



It was easy to add items to the legend, and it is even easier to remove them. Simply use `KDGanttView::clearLegend()`, and all legend items are removed.

Widgets In The Legend Header

It is possible to add your own widgets to the legend header, which is the area of the header where the "Show legend" button is. When you add a widget to this area, the widget will be placed below the "Show legend" button. All widgets added are laid out horizontally, from left to right.

If you want to keep the whole area for your own widget, call `KDGanttView::setShowLegendButton(false)`, to hide the button.

To remove any widget you have added, call `widget->reparent(newParent, ...)` or `delete widget`.

```
QPushButton* button1 = new QPushButton( mw );
button1->setText( "Legend widget 1" );
QPushButton* button2 = new QPushButton( mw );
button2->setText( "Legend widget 2" );

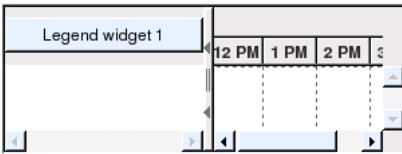
gv->setShowLegendButton( false );❶

gv->addUserdefinedLegendHeaderWidget( button1 );❷
gv->addUserdefinedLegendHeaderWidget( button2 );

delete button2;❸
```

- ❶ Here we hide the "Show Legend" button to make room for our own widgets.
 - ❷ `button1` is added to the legend header. It will take up all possible space, until another widget is added, which is the case in the next row. Of course, it is possible to add any `QWidget` you want, not just `QPushButtons`.
 - ❸ `button2` is deleted, and therefore not shown in the screenshot below.
-
-
-

Figure 6.2. Legend Header Widgets



Visibility

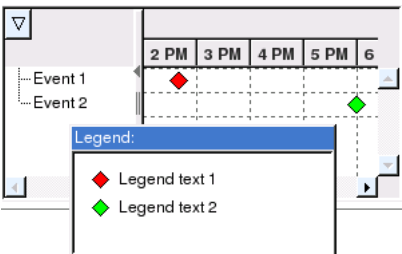
There are several different visibility options for the legend and its associated objects:

1. "Show Legend" button shown or hidden
2. Legend shown or hidden
3. Legend displayed as a dock window or integrated in the widget

We have already seen an example on how to hide the "Show Legend" button, by using `KDGanttView::setShowLegendButton()`. The actual legend can be hidden in a similar way, using `KDGanttView::setShowLegend()`, and pass it either `true` or `false` depending on whether you want to have it visible or hidden.

As mentioned above, the legend can be displayed as a stand-alone dock window, or be integrated in the widget, which is the case in Figure 6.1. Changing the display mode of the legend is done by calling `KDGanttView::setLegendIsDockwindow()`; pass `true` if you want to have the legend in a dock window, or `false` if you would rather have it integrated in the widget.

Figure 6.3. Legends as Dock Windows



► Timeline

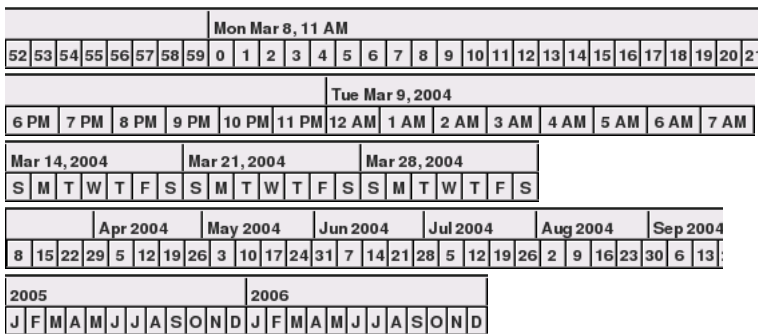
The timeline we have used so far was a pretty basic one, only showing a daily schedule on an hour-based scale. In this section, we will dig deeper into the timeline, to show in what way you can work with it.

Scale

The timeline consists of two scales, one major (upper) and one minor (lower). When you change the scale of the timeline, you change the scale of the minor scale. The major scale is recalculated automatically from the value of the minor scale.

Changing the scale is done by calling `KDGanttView::setScale()`, passing it a scale such as `KDGanttView::Minute`, `KDGanttView::Hour`, `KDGanttView::Day`, `KDGanttView::Week`, `KDGanttView::Month`, or `KDGanttView::Auto`.

Figure 6.4. Different Timeline Scales (Minute to Month)



If you want to set a limit on the minimum or maximum scale available to use, you can use `KDGanttView::setMaximumScale()` and `KDGanttView::setMinimumScale()`, and pass it the scale boundaries. Then, if the scale is set below or above the limits, it is automatically set to either the minimum or maximum value. If, for example, the minimum scale is set to `KDGanttView::Hour`, and the scale is later changed to `KDGanttView::Minute`, it will automatically be set to `KDGanttView::Hour`.

When you are using one of the predefined scales, you can change the resolution of the scale to a value of your choice, by using `KDGanttView::setMajorScaleCount()` and `KDGanttView::setMinorScaleCount()`. Passing an integer value will change the scale accordingly.

The following code shows a timeline displayed in Figure 6.5.

```
gv->setScale( KDGanttView::Minute );
gv->setMajorScaleCount( 3 );
gv->setMinorScaleCount( 15 );
```

Figure 6.5. Customized Scale Count

Mon Mar 8, 12 PM				Mon Mar 8, 3 PM															
0	15	30	45	0	15	30	45	0	15	30	45	0	15	30	45	0	15	30	45

As you can see, the minor scale now only shows every 15th minute, and the major scale only shows every 3rd hour.

When you are using `Auto` as the scale, you may still want to change the ticks. This is done by calling `KDGanttView::setAutoScaleMinorTickCount()`, passing an integer that specifies how many minor ticks you would like to have per major tick. It is not guaranteed that the exact amount of ticks specified is actually shown, due to space and other constraints. For example, if you specify to show 7 ticks, only 6 ticks might be shown.

Appearance

There are several ways in which you can modify the look of the timeline.

If you want to modify the start and end limit of the horizon, you can use `KDGanttView::setHorizonStart()` and `KDGanttView::setHorizonEnd()` and pass an object of type `QDateTime` to both of them. The horizon is the part of the Gantt view that is initialized from the start; you can reach it by dragging the horizontal scrollbar. Extending the horizon at runtime is done by scrolling to one of the edges and then using the arrow buttons of the scrollbar to scroll beyond the initial horizon.

You can easily change without scrolling which part of the timeline you would like to see. These methods can only be used when the Gantt view is visible, i.e., not in the initialization phase, because the parameters could have changed when the Gantt view is finally visible.

You can center the timeline on a specified time using `KDGanttView::centerTimeline()`, passing the date on which to center the timeline. You can also move the timeline to either the start or end of the horizon using `KDGanttView::setTimelineToStart()` and `KDGanttView::setTimelineToEnd()`.

The format of the date and time can also be changed, using `KDGanttView::setYearFormat()` and `KDGanttView::setHourFormat()`. The possible values of the different formats are `FourDigit`, `TwoDigit`, `TwoDigitApostrophe`, and `NoDate` for the year, and `Hour_24`, `Hour_12`, and `Hour_24_FourDigit` for the hour.

One last useful method is `KDGanttView::setWeekendDays()`, to which you pass integers that specify which weekdays are to be considered weekend days. The integers should be a value between 1 and 7 where 1 means Monday and 7 means Sunday; to spe-

cify a weekend from Sunday to Monday you would write `KDGanttView::setWeekendDays(7, 1)`.

Time Interval Selection

It is possible to click and drag the mouse on the timeline to select a time interval. This can then be used to zoom to the current selection, or any other action you may want.

Built into KD Gantt is the ability to zoom to the current selection by connecting the signal `KDGanttView::timeIntervalSelected()` to the slot `KDGanttView::zoomToSelection()` as in the following code example.

```
QObject::connect( gv,
                 SIGNAL( timeIntervalSelected(const QDateTime&,
                                               const QDateTime& ) ),
                 gv,
                 SLOT( zoomToSelection( const QDateTime&,
                                       const QDateTime& ) ) );
```

This will cause the following behavior when selecting a time interval.

Figure 6.6. Zooming Behaviour

Mar 2004	Apr 2004	Ma
1 8 15 22 29	5 12 19 26	3
004	Mar 21, 2004	
12 14 16 18 20 22 24 26 28 30		

▶ Menus And Dialogs

A few dialogs and menus are built into KD Gantt. These include a context menu for the timeline, a context menu for the Gantt chart itself, and a dialog for viewing/editing the properties of an item, accessible by double-clicking an item.



Note

Neither the menus nor the dialog are enabled by default, you need to enable them manually.

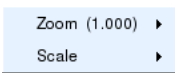
Enabling the menus and the dialog is as simple as calling one method, or connecting one slot.

The Timeline Menu

The timeline context menu is highly customizable. The method `KDGanttView::setShowHeaderPopupMenu()`, which enables or disables the context menu takes seven boolean arguments. The first specifies whether it should be visible or not, and the six further arguments specify whether a certain part of the menu should be enabled or not. These parts are: zoom, scale, time, year, grid, and print.

Calling `KDGanttView::setShowHeaderPopupMenu(true, true, true, false, false, false, false)`, will therefore result in the menu shown in Figure 6.7.

Figure 6.7. The Timeline Context Menu



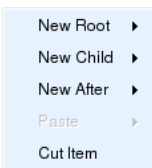
It is also possible to simply call `KDGanttView::setShowHeaderPopupMenu()` without passing any arguments. This will enable the menu with all parts except Print.

The Gantt View Menu

Integrated in KD Gantt is a context menu that is popped up when clicking into the Gantt view with the right mouse button. From this menu, it is possible to add new root items, children of an item, or after another item, as well as cutting and pasting items.

Enabling or disabling this menu is done by calling `KDGanttView::setShowTimeTablePopupMenu()`.

Figure 6.8. The Gantt View Context Menu



Selecting to add a new item as root, child of another item or after another item will bring up the Item Properties dialog, which will be explained in detail in the next section.



Note

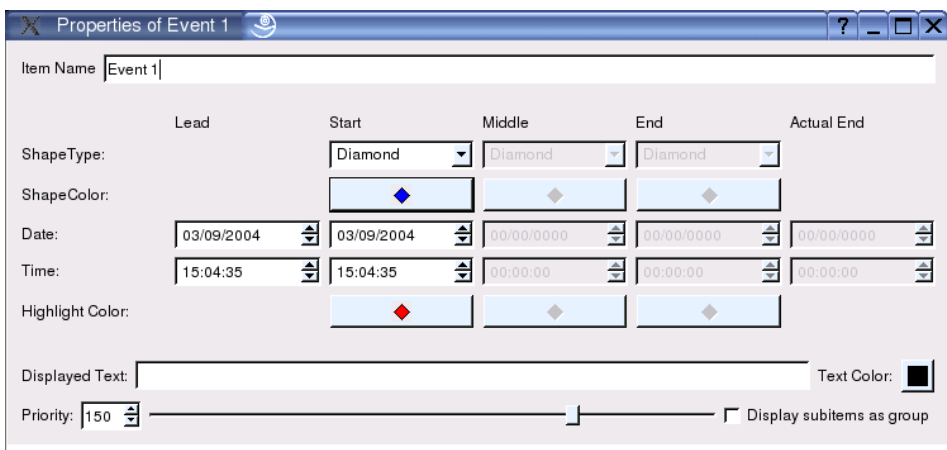
You do not need to implement anything special to get the menu actions working. This is handled internally by KD Gantt.

Item Properties Dialog

The Item Properties dialog is displayed when the user creates a new item using the previously mentioned context menu, or when an item is double-clicked.

A double-click on an item will bring up the dialog with the values for that particular item, but only if the correct slot is connected to the correct signal. The slot that needs to be connected is `KDGanttView::editItem()`. Which signal you connect to it is up to you, but for a double click, it would of course be `KDGanttView::itemDoubleClicked()`.

Figure 6.9. Item Properties Dialog



As for the menus, you do not need to implement anything else besides the slot connection to get the dialog working. All actions inside the dialog are handled by KD Gantt

▶ Drawing and Updating

In some situations such as when loading a large amount of items from a file, you do not want the Gantt view to be updated when each item is added, but rather when all items have been added.

The following code example was ran twice, once with updating turned off and once with updating turned on.

```
for(int i = 0; i < 100; i++) {  
    KDGanttViewTaskItem* task = new KDGanttViewTaskItem( gv, "Task" );  
    task->setStartTime( QDateTime::currentDateTime() );  
}
```

```
task->setEndTime( QDateTime::currentDateTime().addSecs( 3600 ) );
}
```

When the updating was turned on, the time to add all 100 items took 5133 milliseconds, while when the updating was turned off it only took 17 milliseconds. When the amount of items was increased to 150 it took 9475 ms with updating on and 25 ms with updating turned off.

To enable/disable updating, use `KDGanttView::setUpdateEnabled()`. Do not forget to turn the updating back on when the items have been added, otherwise you would not see any change at all.

To improve the readability of the Gantt chart, you may want to add a brush to be drawn on every second line in the Gantt view. You can find a more detailed explanation on how to use brushes in the section Customizing.

Using a brush to draw on every second line will bring up a few problems with updating. If you, for example, are using the brush `Qt::BDiagPattern`, which is the case in Figure 6.10, you see these problems very clearly.

The problem is that when the update mode is set to either `KDGanttView::No` or in some cases even `KDGanttView::Medium`, the appearance of the brush might get distorted around the edges of the Gantt view when scrolling or changing the size of the Gantt view. This happens because the whole Gantt view is not repainted on updates, only the part that needs refreshing, and therefore we might get drawing errors such as in Figure 6.11.

To avoid this problem, simply use a brush whose overall appearance is not as highly dependent on tiling, such as `Qt::HorPattern` or most of the dense patterns, or set the repaint mode to `KDGanttView::Medium` or in the worst case `KDGanttView::Always`.

Figure 6.10. Brush Example

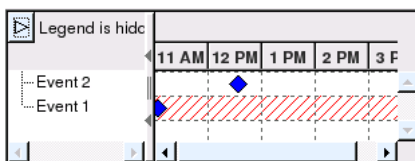
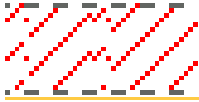


Figure 6.11. Updating Error When Using Brush



Changing the repaint mode is done by calling `KDGanttView::setRepaintMode()`. The available modes are:

<code>KDGanttView::Always</code>	Forces an extra update after every move of the scrollbar. This causes slow scrolling with updated content all the time.
<code>KDGanttView::Medium</code>	Forces an extra repaint after the release of the scrollbar. This provides fast scrolling with updated content after scrolling. Using this repaint mode is recommended; this is also the default.
<code>KDGanttView::No</code>	Causes no repainting after scrolling. This is the fastest mode, but also the one which might cause the most drawing errors.

► Customizing

In this section, we will show you in which ways you can customize the look of your Gantt view.

Size

Changing the height and width of your actual Gantt chart is done by using the appropriate methods in `QWidget`, which is not covered in this manual. Instead, we will focus on manipulating the size of the components inside the Gantt view.

Starting with the timeline, you can set the minimum width a column should have using `KDGanttView::setMinimumColumnWidth()`, passing the requested size in pixels. What is going to happen is that the column in the timeline will not be thinner than the specified value. If the size of the Gantt chart and the scale would make it necessary to go below the specified value, the chart will automatically be made less exact.

The height of an item in the Gantt view depends on whether there is any list view icon associated with a particular item. In that case, the height of the item is changed to make room for the icon. If no icon is specified, the height is set to the default value.

You can also set the maximum width of the Gantt view widget, using `KDGanttView::setGanttMaximumWidth()`, passing an integer value, no larger than 32767.

Next is the list view width, which is changed with `KDGanttView::setListViewWidth()`, which is used in basically every code example in this manual. It takes an integer value that specifies how wide in pixels the list view should be. By default, the list view takes up the entire widget space, so that no Gantt view is visible.

Colors

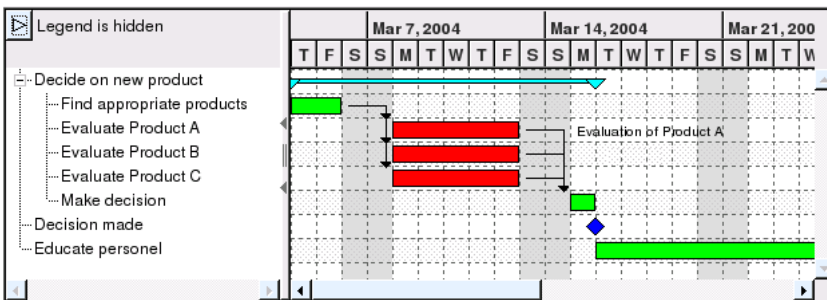
In some situations, you may want to modify the colors of the Gantt chart to better fit in with the rest of the program. In this section, we will show you in which ways you can modify the colors of your Gantt view, including a few code examples and screenshots.

Horizontal Background Lines

We have already seen an example of a background line in the Gantt view in Figure 6.10, but there are ways of modifying the background line that was not explained earlier. Setting the Gantt view to have horizontal background lines is done by calling `KDGanttView::setHorBackgroundLines()`. You can go with the default values, or you can specify your own. The default values is a background with the brush pattern `Qt::Dense6Pattern` in every other line.

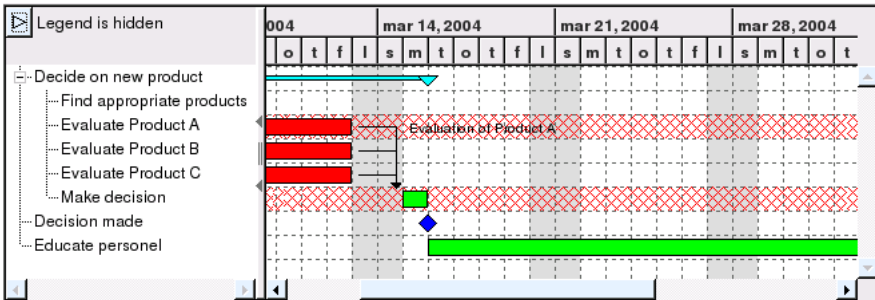
The lines in the screenshot below might be hard to see, because they are light-gray.

Figure 6.12. Default Horizontal Background Lines



To specify your own, pass how often the background line should be repeated (3 means every third line), as well as the `QBrush` to use, to `KDGanttView::setHorBackgroundLines()`. In the following screenshot, we have a background line every third line, and are using a red brush of type `Qt::DiagCrossPattern`.

Figure 6.13. Customized Horizontal Background Lines



Column Background Colors

You can set the background color of one or more columns in several different ways. We will list all of them here, with a short explanation, and then we will show a code example where we use most of them. All of the following methods are members of `KDGGanttView`.

- | | |
|---|--|
| <code>setColumnBackgroundColor()</code> | Sets the background color of the specified column. You will need to specify which column to fill, which color to use and optionally between which scales it should be visible. |
| <code>setIntervalBackgroundColor()</code> | Sets the background color between two specified dates. |
| <code>changeBackgroundInterval()</code> | You can use this method to change an already existing background interval. You will need to specify the old start and end times and the new start and end times. |
| <code>deleteBackgroundInterval()</code> | Deletes an already existing background interval from the Gantt view. You will need to specify between which dates the interval exists. |
| <code>clearBackgroundColor()</code> | Removes all background settings set with the previously mentioned methods. This does not affect the backgrounds set with <code>setWeekendBackgroundColor()</code> and <code>setWeekdayBackgroundColor()</code> . |
| <code>setWeekendBackgroundColor()</code> | Sets the background color for weekend days. |

You will need to specify which color to use.

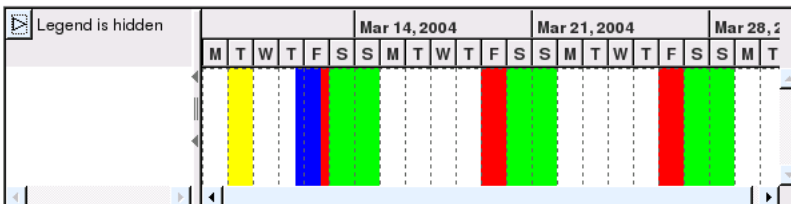
`setWeekdayBackgroundColor()` Sets the background color for a specific week day. You will need to pass which color to use, and on which day.

The complete code example is located in `step04e.cpp`, and the output is displayed in Figure 6.14

```
gv->setColumnBackgroundColor( QDateTime::currentDateTime(), ❶  
    Qt::yellow,  
    KDanttView::Minute,  
    KDanttView::Day );  
  
gv->setIntervalBackgroundColor( QDateTime::currentDateTime().addDays( 2 ), ❷  
    QDateTime::currentDateTime().addDays( 3 ),  
    Qt::blue );  
  
gv->setWeekendBackgroundColor( Qt::green ); ❸  
  
gv->setWeekdayBackgroundColor( Qt::red, 5 ); ❹
```

- ❶ This colors the column containing the current time yellow. The last two arguments specify that this background color should only be visible when the scale is between Minute and Day, i.e., when the scale is Month, this will not be shown.
- ❷ This adds a blue background stretching over a full day. Since we did not specify any limits for the scale, this background is displayed independent of the scale the Gantt view is in.
- ❸ This sets the background color for weekend days to green. From now on, all weekends in the Gantt view will be green.
- ❹ This sets the background color of all Fridays (day number 5) to red. As shown in the screenshot, this background color is below the background color set by `setIntervalBackgroundColor()`.

Figure 6.14. A Customized Gantt View Background



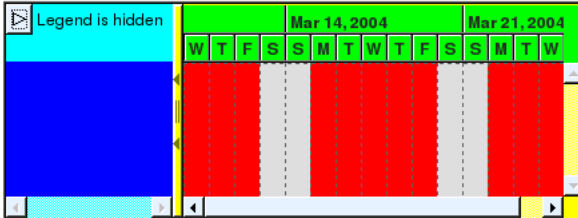
Widget Background Colors

If you want to, you can change the background color of virtually every part in KD Gantt. The available parts to change are the palette, the Gantt view, the list view, the time header and the legend header. This is done by calling the appropriate method, passing a `QColor` object. Have a look at the following code example that shows the methods to use.

```
gv->setPaletteBackgroundColor( Qt::yellow );
gv->setGvBackgroundColor( Qt::red );
gv->setLvBackgroundColor( Qt::blue );
gv->setTimeHeaderBackgroundColor( Qt::green );
gv->setLegendHeaderBackgroundColor( Qt::cyan );
```

The complete code example is located in `step04f.cpp`. The result is displayed in Figure 6.15.

Figure 6.15. Customized Widget Backgrounds



Grids

To improve the readability of the chart, a grid is displayed in the Gantt view. You can modify the appearance of this grid in a few ways.

First of all, there are two grids in KD Gantt—one major and one minor. The minor grid, or ticks, is aligned with the minor time scale, and the major ticks are aligned with the major time scale. The minor grid is what is displayed by default, while the major one is hidden. To show or hide any of the grids, use the methods `KDGanttView::setShowMajorTicks()` and `KDGanttView::setShowMinorTicks()`.

Brushes

As we have shown earlier, you can use brushes in the Gantt view to make it more appealing to the eye. There are two places in KD Gantt where brushes can be used—as horizontal background lines in the Gantt view, and as the "no information" line, even that one in the Gantt view.

When specifying which brush to use, you give it a color and a pattern. The default brush for the horizontal background lines is a brush with the pattern `Qt::Dense6Pattern`; the brush for the "no information"-line has default pattern `Qt::FDiagPattern`. Both are shown in light gray color.

To change the brushes used, use the methods `KDGanttView::setNoInformationBrush()`, passing the brush to use, or `KDGanttView::setHorBackgroundLines()`, passing how often there should be a horizontal background line, as well as the brush to use. If no arguments are passed to the last method, the default brush will be used on every second line.

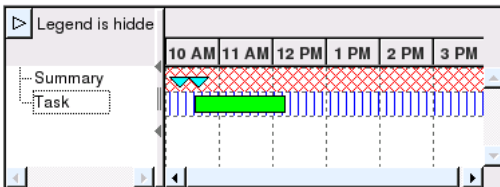
```
gv->setNoInformationBrush( QBrush( Qt::red, Qt::DiagCrossPattern ) );
gv->setHorBackgroundLines( 2, QBrush( Qt::blue, Qt::VerPattern ) );

KDGanttViewTaskItem* task =
    new KDGanttViewTaskItem( gv, "Task" );
task->setStartTime( QDateTime::currentDateTime().addSecs( 1000 ) );
task->setEndTime( QDateTime::currentDateTime().addSecs( 7200 ) );

KDGanttViewSummaryItem* summary =
    new KDGanttViewSummaryItem( gv, "Summary" );
summary->setShowNoInformation( true );
```

This code will result in the Gantt view of Figure 6.16. A complete example can be found in `step04g.cpp`.

Figure 6.16. Customized Brushes



► Drag-and-Drop

You can enable or disable drag-and-drop in KD Gantt simply by calling `KDGanttView::setDragDropEnabled()`.

When drag-and-drop is enabled, you can drag and drop items to and from the list view. For example, if you have two Gantt charts next to each other, you can drag an item from one of the Gantt charts and drop it onto the other.

When you drag an item, an icon is displayed next to the mouse pointer. This icon is the list view icon of the item, or, if the item has no list view icon defined, the start shape of the item.

If you require more control, KD Gantt lets you implement your own drag-and-drop behavior. To do this, you need to subclass `KDGanttView` and implement the methods `lvDropEvent()`, `lvDragEnterEvent()`, `lvDragMoveEvent()` and `lvStartDrag()`. For a closer explanation on what to keep in mind when doing this, have a look in the Reference Manual.

There is a complete drag-and-drop sample application in `step04h.cpp`. In this example, we have also enabled the menus, as well as zooming and the property dialog.

In the example application, you can drag items between the left and right views. When you are dragging and dropping items, they are moved by default. If you hold down `Control` while dragging and dropping, they are copied instead.

► What's Next

In the next chapter we will have a look at the Calendar display mode; what it is, and how you use it...

Chapter 7. Calendar Mode

► Introduction to Calendar mode

When the Gantt view is in Calendar mode instead of the "normal" mode, you do not see regular tasks, but rather all tasks belonging to a certain calendar object, on one line.

The two following screenshots show the same tasks in "normal" mode and Calendar mode.

Figure 7.1. "Normal" mode

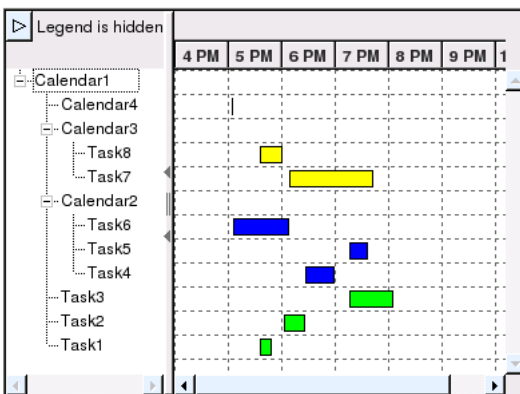
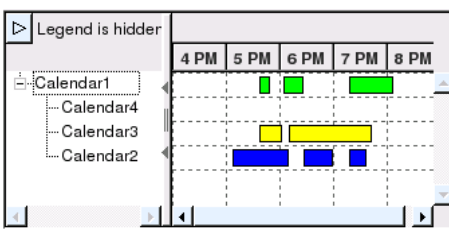


Figure 7.2. Calendar mode



As you can see, the regular tasks are not displayed when the Gantt view is in Calendar mode. They are all gathered onto the Calendar line.

► Procedure

First of all, to set the Gantt view to be in Calendar mode, you will need to call `KDGanttView::setCalendarMode(true)`. Then you use objects of type `KDGanttViewItem` to create your chart.

All `KDGanttViewItem` objects where `KDGanttViewItem::displaySubitemsAsGroup()` returns `true` are considered to be Calendar objects. Those where the return value is `false` are considered to be regular tasks, and are therefore shown on the Calendar line. This value is changed with `KDGanttViewItem::setDisplaySubitemsAsGroup()`.

Calendar objects are added to the Gantt view in the normal way, which is also true for regular tasks, with the difference that regular tasks should be added as children to Calendar objects.

► Example

To make the above explanation more practical, here is a code example. The output of this, is shown in Figure 7.2.

```
1
    /* -*- Mode: C++ -*-
   KDGantt
   */
5
   /*****
   ** Copyright (C) 2001-2003 Klar#lvdalens Datakonsult AB. All rights reserved.
   **
   ** This file is part of the KDGantt library.
10
   ** This file may be distributed and/or modified under the terms of the
   ** GNU General Public License version 2 as published by the Free Software
   ** Foundation and appearing in the file LICENSE.GPL included in the
   ** packaging of this file.
15
   ** Licensees holding valid commercial KDGantt licenses may use this file in
   ** accordance with the KDGantt Commercial License Agreement provided with
   ** the Software.
20
   ** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
   ** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
   **
   ** See http://www.klaralvdalens-datakonsult.se/?page=products for
   ** information about KDGantt Commercial License Agreements.
25
   ** Contact info@klaralvdalens-datakonsult.se if any conditions of this
   ** licensing are not clear to you.
   **
   *****/
30
   #include <qapplication.h>
   #include <qmainwindow.h>

   #include "KDGanttView.h"
35 #include "KDGanttViewItem.h"
```

```

int main( int argc, char* argv[] )
{
    QApplication a( argc, argv );
40    QMainWindow* mw = new QMainWindow();
    KDGanttView* gv = new KDGanttView( mw );
    mw->setCentralWidget( gv );
    gv->setScale( KDGanttView::Hour );
    gv->setCalendarMode( true );
45
    KDGanttViewTaskItem* calendar1 =
        new KDGanttViewTaskItem( gv, "Calendar1" );
    calendar1->setDisplaySubitemsAsGroup( true );

50    KDGanttViewTaskItem* task1 =
        new KDGanttViewTaskItem( calendar1, "Task1" );
    task1->setDisplaySubitemsAsGroup( false );
    task1->setStartTime( QDateTime::currentDateTime().addSecs( 2000 ) );
    task1->setEndTime( QDateTime::currentDateTime().addSecs( 2750 ) );
55
    KDGanttViewTaskItem* task2 =
        new KDGanttViewTaskItem( calendar1, "Task2" );
    task2->setDisplaySubitemsAsGroup( false );
    task2->setStartTime( QDateTime::currentDateTime().addSecs( 3600 ) );
60    task2->setEndTime( QDateTime::currentDateTime().addSecs( 5000 ) );

    KDGanttViewTaskItem* task3 =
        new KDGanttViewTaskItem( calendar1, "Task3" );
    task3->setDisplaySubitemsAsGroup( false );
65    task3->setStartTime( QDateTime::currentDateTime().addSecs( 8000 ) );
    task3->setEndTime( QDateTime::currentDateTime().addSecs( 11000 ) );

    gv->setDefaultColor( KDGanttViewItem::Task, Qt::blue );

70    KDGanttViewTaskItem* calendar2 =
        new KDGanttViewTaskItem( calendar1, "Calendar2" );
    calendar2->setDisplaySubitemsAsGroup( true );

    KDGanttViewTaskItem* task4 =
75    new KDGanttViewTaskItem( calendar2, "Task4" );
    task4->setDisplaySubitemsAsGroup( false );
    task4->setStartTime( QDateTime::currentDateTime().addSecs( 5000 ) );
    task4->setEndTime( QDateTime::currentDateTime().addSecs( 7000 ) );

80    KDGanttViewTaskItem* task5 =
        new KDGanttViewTaskItem( calendar2, "Task5" );
    task5->setDisplaySubitemsAsGroup( false );
    task5->setStartTime( QDateTime::currentDateTime().addSecs( 8000 ) );
    task5->setEndTime( QDateTime::currentDateTime().addSecs( 9250 ) );
85

    KDGanttViewTaskItem* task6 =
        new KDGanttViewTaskItem( calendar2, "Task6" );
    task6->setDisplaySubitemsAsGroup( false );
    task6->setStartTime( QDateTime::currentDateTime().addSecs( 200 ) );
90    task6->setEndTime( QDateTime::currentDateTime().addSecs( 4000 ) );

    gv->setDefaultColor( KDGanttViewItem::Task, Qt::yellow );

    KDGanttViewTaskItem* calendar3 =
95    new KDGanttViewTaskItem( calendar1, "Calendar3" );
    calendar3->setDisplaySubitemsAsGroup( true );

    KDGanttViewTaskItem* task7 =
        new KDGanttViewTaskItem( calendar3, "Task7" );
100    task7->setDisplaySubitemsAsGroup( false );
    task7->setStartTime( QDateTime::currentDateTime().addSecs( 4000 ) );
    task7->setEndTime( QDateTime::currentDateTime().addSecs( 9600 ) );

    KDGanttViewTaskItem* task8 =
105    new KDGanttViewTaskItem( calendar3, "Task8" );
    task8->setDisplaySubitemsAsGroup( false );

```

```
task8->setStartTime( QDateTime::currentDateTime().addSecs( 2000 ) );
task8->setEndTime( QDateTime::currentDateTime().addSecs( 3500 ) );

110   KDGanttViewItem* calendar4 =
        new KDGanttViewItem( calendar1, "Calendar4" );
calendar4->setDisplaySubitemsAsGroup( true );

gv->show();
115   gv->setListViewWidth( 150 );
mw->setCaption( "KDGantt - Calendar Mode" );
mw->resize( 600, 400 );
mw->show();
a.setMainWidget( mw );
120   return a.exec();
}
```