Programmers Manual

KD REPORTS

KDAB

www.kdab.com

Table of Contents

List of Figures

Chapter 1. Introduction

What is KD Reports

KD Reports is a fully featured Qt tool that allows users to create printable reports for a variety of applications.

Users can generate reports by two methods. They can use KD Reports to generate code, with its easy-to-use C++ API, or they can produce reports from structured sources like XML or SQL databases. KD Reports formats data into useful styles complete with watermarks, headers and footers. Once generated, reports can be previewed, printed, or saved as PDF or HTML files. For those using KDAB's KD Chart, KD Reports easily incoporates graphics generated with KD Chart into their final documents.

KD Reports is intended for C++ programmers who use Qt in their applications. This reference assumes an understanding of the C++ programming language and a working knowledge of Qt.

Installation

Licensed customers may install KD Reports from a binary package, with a graphic user interface, or directly from the source code. Experienced developers may choose the latter option, as it allows programmers to integrate the KD Reports source code directly into their own builds.

Feature Summary

KD Reports reports are simple C++ objects integrated into your project's application source code. KD Reports provides programmers with tools to implement “reporting modules” inside their own applications.

KD Reports uses XML templates. Programmers use the templates to design the look and structure of the reports and to create “placeholders” for report content. This is helpful when integrating data from non-technical staff. At runtime, the application fills in the placeholders with user supplied data and generates the report.

Reports can contain tables generated from Qt's abstract item models. With the QtSql module, applications retrieve report table data from SQL databases. Because reports are generated within application logic, programmers can apply higher-level calculations within the same programming environment as in the rest of the application.

Charts can be added to reports with KDAB's KD Chart package. KD Chart provides developers with a wealth of contrasting chart types (as well as great performance, since it is also written in C++). Charts are regular report elements, and can be formatted the same way as any other element.

Any image format supported by Qt can be added to reports. Images can be added as elements of the report or as watermarks.

Headers and footers that repeat on every page can be filled with any type of elements, such as text and images.

For structuring the reports, groups of elements can be arranged horizontally or vertically. Page breaks are performed automatically. Printing on endless paper is supported.

What's New in KDReports?

New Features in Version 1.3

- **Spreadsheet mode**, for faster, more powerful printing with large tables. A table can now be broken across multiple pages horizontally or vertically. Tables can be forced to fit into N*M pages, using a smaller font if necessary. (See Report::setReportMode for the difference between the two modes.)
- New XML tag **<hline/>** to simplify the use of horizontal lines in XML.
- Support for custom color, configurable thickness and margin.
- **XmlElementHandler** has a new virtual method, `hLineElement`, called upon when parsing an *hline*. Note: this new virtual is only enabled if KDReports is compiled with the define KDREPORTS_ALLOW_BINARY_INCOMPATIBILITY.
- New XML tag **<ifdef id="...">** to skip sections of the XML template when a text value is empty. Note: this is only evaluated at loading time. Changing the text value later on will not have an effect on “skipping sections.”
- Support for configuring the charts: new constructor `ChartElement(KDChart::Chart*)` for charts created in code, and `setChart(KDChart::Chart*)` for setting the chart from the *XmlElementHandler*.
- Improved charts rendering and printing by drawing charts at full resolution (without the use of an intermediate pixmap).
- New method `setDefaultFont` for all table elements.
- Mail merge support (see examples/MailMergeXML).

The Structure of this Manual

This manual contains the following main parts:

- Introduction - you are reading this section.
- Programmer's Manual - for first time users.
- Programmer's Reference - a complete reference for KD Reports functionality.
- Appendix - contains sample reports and licensing information

Chapter 2. Programmers Manual

This Chapter introduces the KD Reports API. In the following examples, we will use KD Reports to illustrate how the API works. Let's start with a simple "Hello World" application.

Generating a Report: Hello World

This first example shows how to generate a basic report.

```
...

int main( int argc, char** argv ) {
  QApplication app( argc, argv );

  // Create a report
  KDReports::Report report;

  // Add a text element for the title
  KDReports::TextElement titleElement( QObject::tr( "Hello World!" ) );
  titleElement.setPointSize( 18 );
  report.addElement( titleElement, Qt::AlignHCenter );

  // add 20 mm of vertical space:
  report.addVerticalSpacing( 20 );

  // add some more text
  KDReports::TextElement textElement(
  QObject::tr( "This is a report generated with KD Reports" ) );
  report.addElement( textElement, Qt::AlignLeft );

  // show a print preview
  KDReports::PreviewDialog preview( &report );
  return preview.exec();
}
```

A “report” is a set of output pages. The first line of code creates an object of the class KDReports::Report. Next, we give the report a title element containing the classic text: “Hello World!” Line four creates a second text element, left aligned, containing the text “This is a report generated with KD Reports.” KD Reports allows formatting functions. The third line of code creates a vertical space of 20mm between the two text elements. This is all you need to create a basic report. It can now be printed, saved to a PDF, saved as an HTML file, or displayed in a preview window. In this particular example, the last two lines of code generate a preview dialog displaying our report.

In the next section, we'll generate a slightly more complex report—a letter.

A Letter from KDAB

In the following example, we will show you how to generate a fully featured letter with KD Reports. We will format headers and footers, align multiple bodies of text, create indented textual elements, include a table generated from an Qt Model/View Framework model, and place a picture.

```
...
```

```
// Create a report
KDReports::Report report;

// create a table:
QStandardItemModel model;
model.setHorizontalHeaderItem( 0, new QStandardItem( QObject::tr( "Product" ) ) );
...
model.setItem( 0, 0, new QStandardItem( QObject::tr( "KD Reports 1.0" ) ) );
...
```

First, the report object and a QStandardItemModel are initialized. QStandardItemModel implements QAbstractItemModel: the class that KD Reports uses to fill a table.

```
...
// Add From: and To: information:
KDReports::TextElement fromElement;
fromElement << "From:\n"
            << "Klarälvdalens Datakonsult AB\n"
            << "Hagfors\n"
            << "Sweden\n";
report.addElement( fromElement );
report.addVerticalSpacing( 10 );
KDReports::TextElement toElement;
toElement << "To:\n"
          << "All Qt Users Out There In The World\n";
report.addElement( toElement );
report.addVerticalSpacing( 10 );

// Add a text element for the title
KDReports::TextElement titleElement( QObject::tr(
        "Ever wanted to create a printed report from within your Qt application?\n"
        "Look no further!" ) );
titleElement.setPointSize( 14 );
report.addElement( titleElement, Qt::AlignCenter );
...
```

Next, the “From:” and “To:” text elements are added. Line breaks can be added to text elements, as needed. Text can also be streamed into the text element object. The “title element” is a text element as well, but it gets a slightly larger font and is added to the report with a centered alignment.

```
...

// Add another text element, demonstrating "<<" operator
KDReports::TextElement bodyElement;
bodyElement.setPointSize( 10 );
bodyElement << QObject::tr( "Dear KDAB Customers,\n" );
bodyElement <<
 "we are happy to introduce the newest member of KDAB's line of industry "
 "leading software products: KD Reports. KD Reports is the Qt tool to "
 "easily create printable reports. It provides all necessary features for "
 "a variety of applications:\n"
 "Reports can be created programmatically, using an easy to use C++ API, "
 "or can be data-driven, creating reports from XML or SQL data sources "
 "complete with watermarks, headers and footers. Reports can be previewed "
 "manually, sent directly to a printer, or saved as PDF files. "
 "Additionally, using KDAB's KD Chart package together with KD Reports allows "
 "reports to be garnished with the myriad of chart types supported by KD Chart."
report.addElement( bodyElement, Qt::AlignJustify );

// release date table:
KDReports::AutoTableElement tableElement( &model );
tableElement.setBorder( 1 );
report.addElement( tableElement, Qt::AlignCenter );
report.addVerticalSpacing( 6 );

KDReports::TextElement body2Element;
body2Element.setPointSize( 14 );
body2Element <<
```

```
 "Reporting is a rather general feature, and it seems many were looking for a nice
 "package providing this kind of functionality to complement Qt."
 "We at KDAB hope to make the life of our customers more enjoyable with it. "
 "Let us know if we were successful!\n";
report.addElement( body2Element, Qt::AlignJustify );
report.addVerticalSpacing( 30 );
KDReports::TextElement signatureElement;
signatureElement << QObject::tr( "Cheers,\n" ) << QObject::tr(
 "Klarälvdalens Datakonsult AB, Platform-independent software solutions" );

report.addElement( signatureElement );

...
```

Next, the body of the letter is added. It consists of two blocks of text. Both are added with justified alignment. Between the two paragraphs, we add a table that shows the KD Reports release dates. Adding a table is as simple as instantiating a table object and plugging the data model into it. Finally, the signature is added (it wouldn't be a letter without it.) Of course, the footer should contain contact info. Let's add that:

```
...

// add footer with contact information:
KDReports::HtmlElement rulerElement;
rulerElement << "<hr />";
report.footer().addElement( rulerElement );
KDReports::TextElement footerText;
footerText << "www.kdab.com | email: info@kdab.com | +46-563-540090";
footerText.setPointSize( 8 );
report.footer().addElement( footerText, Qt::AlignCenter );

...
```

The footer shows our first usage of KDReports::HtmlElement. HtmlElement contains text in HTML markup. Here, we use it to create a horizontal ruler. The following lines include our contact information; a little smaller this time, and centered. This completes the basic structure of our letter.

In the following example, we will demonstrate how to include different types of tables in our reports.

Tables with Formatting and a Page-break: PriceList

Tables are a common element in reports. This price list example shows how tables can be generated from code, from model data, or loaded from CSV files (using a helper class provided with KD Reports).

```
...
TableModel table1;
table1.setDataHasVerticalHeaders( false );
table1.loadFromCSV( ":/table1" );
KDReports::AutoTableElement autoTableElement1( &table1 );
autoTableElement1.setWidth( 100, KDReports::Percent );
report.addElement( autoTableElement1 );
...
```

This section fills the table1 object with data loaded from the CSV formatted resource ":/table1". TableModel implements QAbstractItemModel, so the KDRe-

ports::AutoTableElement can use it directly. The table is set to a width of 100%. This matches nicely with justified text paragraphs located above or below the table.

```
...
KDReports::TableElement tableElement;
tableElement.setHeaderRowCount( 2 );
tableElement.setPadding( 3 );
QColor headerColor( "#DADADA" );
// Merged header in row 0
KDReports::Cell& topHeader = tableElement.cell( 0, 0 );
topHeader.setColumnSpan( 2 );
topHeader.setBackground( headerColor );
topHeader.addElement( KDReports::TextElement( "TableElement example" ),
                                              Qt::AlignHCenter );

// Normal header in row 1
KDReports::Cell& headerCell1 = tableElement.cell( 1, 0 );
headerCell1.setBackground( headerColor );
QPixmap systemPixmap( ":/system.png" );
headerCell1.addElement( KDReports::ImageElement( systemPixmap ) );
headerCell1.addInlineElement( KDReports::TextElement( " Item" ) );
KDReports::Cell& headerCell2 = tableElement.cell( 1, 1 );
headerCell2.setBackground( headerColor );
KDReports::TextElement expected( "Expected" );
expected.setItalic( true );
expected.setBackground( QColor( "#999999" ) );
headerCell2.addElement( expected );
headerCell2.addInlineElement( KDReports::TextElement( " shipping time" ) );

// Data in rows 2 and 3
tableElement.cell( 2, 0 ).addElement(
  KDReports::TextElement( "Network Peripherals" ) );
tableElement.cell( 2, 1 ).addElement(
  KDReports::TextElement( "4 days" ) );
tableElement.cell( 3, 0 ).addElement(
  KDReports::TextElement( "Printer Cartridges" ) );
tableElement.cell( 3, 1 ).addElement(
  KDReports::TextElement( "3 days" ) );

report.addElement( tableElement );
...
```

Here, a table is created directly from KDReports::Cell objects. When using this technique, it is not necessary to provide a data model. It also allows you to add more detailed formatting information for each individual cell. You can also add other elements, like pictures, to each cell. Model driven tables provide such information by returning it from QAbstractItemModel::data(..).

Using SQL Data in Reports: A Database Example

Of course, reports are not static and often present information retrieved from other data sources. SQL databases are probably the most common of these. All databases supported by Qt's QtSql module, including Oracle, MySQL, and many others, can be used in reports. This example shows how to use the content of a sample in-memory SQLite database.

```
...
// open a DB connection to an in-memory database
QSqlDatabase db = QSqlDatabase::addDatabase( "QSQLITE" );
db.setDatabaseName( ":memory:" );
if( !db.open() ) {
  QMessageBox::critical( 0, QObject::tr( "Cannot open database" ),
    QObject::tr( "Cannot create connection to the requested database. "
```

```
                "Your Qt is probably lacking the QSQLITE driver. "
                "Please check your Qt installation." ), QMessageBox::Cancel );
  return false;
}

// fill the DB with some test data
QSqlQuery query;
query.exec( "create table airlines (id int primary key, "
            "name varchar(20), homecountry varchar(2))" );
query.exec( "insert into airlines values(1, 'Lufthansa', 'DE')" );
query.exec( "insert into airlines values(2, 'SAS', 'SE')" );
query.exec( "insert into airlines values(3, 'United', 'US')" );
query.exec( "insert into airlines values(4, 'KLM', 'NL')" );
query.exec( "insert into airlines values(5, 'Aeroflot', 'RU')" );
...
```

This piece of code creates the in-memory sqlite database. It will be used to initialize a QSqlTableModel. Then, the model will be used to form a table in the report.

```
...
// Create a QSqlTableModel, connect to the previously created database, fill
// the db with some data.
QSqlTableModel tableModel( 0, db );
tableModel.setTable( "airlines" );
tableModel.select();
tableModel.removeColumn( 0 );
tableModel.setHeaderData( 0, Qt::Horizontal, QObject::tr( "ID" ) );
tableModel.setHeaderData( 1, Qt::Horizontal, QObject::tr( "Name" ) );
tableModel.setHeaderData( 2, Qt::Horizontal, QObject::tr( "Home country" ) );
report.addElement( KDReports::AutoTableElement( &tableModel ) );
...
```

After the model object is created and connected to the database, a KDReports::AutoTableElement creates a table from the query results. Column 0 is removed, so that the record ids are not shown.

Chapter 3. XML Templates

XML report templates generally offer shorter turnaround time for modifications than C++ source code. They are easier for non-programmers to understand and to edit, as well.

Hello World as XML

The details of the XML syntax are described in the reference section of this manual. Here, we will discuss only the details of loading the report from an XML file:

```
...
int main( int argc, char** argv ) {
  QApplication app( argc, argv );

  // Create a report
  KDReports::Report report;

  QFile reportFile( "HelloWorld.xml" );
  if( !reportFile.open( QIODevice::ReadOnly ) ) {
    QMessageBox::warning(0, QObject::tr( "Warning" ), QObject::tr(
                           "Could not open report description file 'HelloWorld.xml'. "
                           "Please start this program from the HelloWorldXML directory." ) );
    return -1;
  }

  if( !report.loadFromXML( &reportFile ) ) {
    QMessageBox::warning( 0, QObject::tr( "Warning" ),
                          QObject::tr( "Could not parse report description file." ) );
    reportFile.close();
    return -2;
  }

  // show a print preview:
  KDReports::PreviewDialog preview( &report );
  return preview.exec();
}
...
```

This is the complete main() function of the HelloWorldXML example. A report object is generated, and the file HelloWorld.xml is loaded into the report. Done! In later examples, we will show how to bind model data to structures in the XML syntax and add other elements like images.

The PriceList Example as XML

To use model data and other resources in reports generated from XML templates, the XML elements and the data sources need to be associated with each other. To achieve this, XML element ids are mapped to text elements or data sources.

```
...

// Create a report
KDReports::Report report;

// Set the content of a text field - this shows how xml files can be
// used as templates for reports, not only as complete (generated) reports.
report.associateTextValue( "title_element", "Price list example" );
```

```
report.associateTextValue( "company_address", "Klarälvdalens Datakonsult AB\n"
                           "Rysktorp\n"
                           "SE-68392 Hagfors\n"
                           "Sweden" );
// Note: id="table1_title" is used twice in the xml, both places get the right value
report.associateTextValue( "table1_title", "Network Peripherals" );
report.associateTextValue( "table2_title", "Printer Cartridges" );

report.associateImageValue( "image_system", QPixmap( ":/system.png" ) );

...
```

In this example, "title_element" is the id of a text element in the XML sources. The content of the XML element will be replaced with the given value.

```
...

// Create two table models which will be used by one table element each.
TableModel table1;
table1.setDataHasVerticalHeaders( false );
table1.loadFromCSV( ":/table1" );
report.associateModel( QLatin1String( "table1" ), &table1 );
TableModel table2;
table2.setDataHasVerticalHeaders( false );
table2.loadFromCSV( ":/table2" );
report.associateModel( QLatin1String( "table2" ), &table2 );

...
```

Here, the data model is associated with a table tag (element) in the XML source. The respective XML snippet looks like this:

```
...
<table model="table1" width="100%"/>
...
```

The examples show that after associating the XML elements with programatically created report elements and data sources, the report XML template can now be modified in style and content independently of the program source code.

Chapter 4. References

A Brief Look at the API

KD Reports offers classes for embedding content such as images, text, HTML and charts, as well as ways for arranging them. Use the KDReports::AutoTableElement for the Model-View approach, or just arrange your elements in a KDReports::TableElement with cells and a header. Content that will allow printing, watermarks, exports and layout can be added to an instance of the report class. Use the preview dialog class to give users an on-screen version of the report.

The KD Reports XML Template Format

Introduction

XML report templates are a means to separate the layout of a report from its data acquisition and processing logic. Templates are quicker to modify and test than C++ code as well. While a working knowledge of XML is required to use them, they still should be much easier for non-programmers to understand than C++.

Element Reference

Common Attributes in Detail

Here we describe in detail some attributes that are common to many elements. These descriptions are too long to repeat for every element, so you'll only find a short description in the element documentations. We recommend reading this section before reading any element descriptions.

Common Attributes	
color	Set the foreground color. The #RRGGBB format known from HTML where each letter represents a hex digit for a color component is valid, but also the coarser #RGB and the finer #RRRGGGBBB and #RRRRGGGGBBBB formats. Additionally SVG named colors as defined by the World Wide Web Consortium (W3C) such as “royalblue”, “chartreuse” or “transparent” can be used.
background	Set the background color; the format is the

Common Attributes	
	same as color.
bgred, bggreen, bgblue	Set the RGB color components of the background separately. You must use either none or all three of them. Each component has a value range of 0-255 and values must be given in decimal.
width, height	Values can be given either in millimeters like 'width="100"' (note not 'width="100mm"') or in percentage of the page width/height like 'width="50%"'.
inline	If the inline attribute is present in an element, it is inserted into the current flow of text or other layout items. If it is not present, the element is put into an area of its own.

<report>

The top-level element of a KD Reports XML template. It is used to set up the page layout and to set some defaults for the report.

Attributes	
font	Set the default font face for the document.
pointsize	Set the default font size for the document in typographic points.
header-body-spacing	Set the spacing between header and body in millimeters.
footer-body-spacing	Set the spacing between footer and body in millimeters.
margin-top	Set the page's top margin in millimeters.
margin-top	Set the page's top margin in millimeters.
margin-left	Set the page's left margin in millimeters.
margin-bottom	Set the page's bottom margin in millimeters.
margin-right	Set the page's right margin in millimeters.

<text>

A section of text.

Attributes	
id	This is used to refer to the text in code and replace it with a programmatically obtained value. If no such replacement takes place, the element contents will be used.
background	Set the background color.
bgred, bggreen, bgblue	Set the RGB color components of the background separately. You cannot set one or two components only; all three must be present.
font	Set the font face.
pointsize	Set the font size in typographic points.
color	Set the text color.
bold	Make the text bold.
italic	Make the text italic.
strikeout	Strike out the text using an horizontal line.
underline	Underline the text.
inline	Add as inline element.
alignment	Set the horizontal alignment; possible values are "left", "right" and "hcenter". This is only applicable if the inline attribute is not present.
paragraph-background	Set the background of the whole paragraph rectangle. Possible values are like color or background. This is only applicable if the inline attribute is not present.
model	The model key for the model that supplies the data - see the API documentation of KDReport::associateModel(). If you are not a programmer, we recommend that you pick a unique and descriptive name for the data source to be used. If no model is associated with this key, the element contents will be used as fallback.
row	To be used together with the model attribute. Sets the row in the model from which to extract the text.
column	To be used together with the model attribute. Sets the column in the model from which to extract the text.

<html>

A section of HTML-formatted rich text.

Attributes	
id	This is used to replace the HTML code with a programmatically generated value. If no such replacement takes place, the element contents will be used.
background	Set the background color.
bgred, bggreen, bgblue	Set the RGB color components of the background separately. You cannot set one or two components only; all three must be present.
inline	Add as inline element.
alignment	Set the horizontal alignment; possible values are "left", "right" and "hcenter". This is only applicable if the inline attribute is not present.
model	The model key for the model that supplies the data - see the API documentation of KDReport::associateModel(). If you are not a programmer, we recommend that you pick a unique and descriptive name for the data source to be used. If no model is associated with this key, the element contents will be used as fallback.
row	To be used together with the model attribute. Sets the row in the model from which to extract the text.
column	To be used together with the model attribute. Sets the column in the model from which to extract the text.

<hline />

Insert a horizontal line. This element has no content.

Attributes	
color	The color of the line. This attribute is optional, the default color is gray.
thickness	The thickness of the line, in pixels. This attribute is optional, the default thickness is

Attributes	
	2 pixels.
margin	The empty space (in mm) above and below the line that slightly separates it from the surrounding text. This attribute is optional, the default value is 6mm above and below the line.

<vspace />

Insert a vertical blank space. This element has no content.

Attributes	
size	The vertical blank size in millimeters. This attribute is mandatory.

<page-break />

Insert a page break. This element has no attributes and no content.

<image />

Insert an image into the report. This element has no content.

Attributes	
file	A path to an image file.
id	This is used to replace the image with a programmatically obtained image. If no such replacement takes place, the image from the file attribute will be used.
width	Set the width in millimeters or percentages of the page width. Mutually exclusive with height and fitToPage.
height	Set the height in millimeters or percentages of the page height. Mutually exclusive with width and fitToPage.
fitToPage	Make the image fit one page. Mutually exclusive with width and height.
inline	Add as inline element.
alignment	Set the horizontal alignment; possible values are "left", "right" and "hcenter". This is

Attributes	
	only applicable if the inline attribute is not present.

<header>

Define a page header for one or several pages. A header is a content area in its own right and may contain almost any other element.

Attributes	
location	Specify which pages the header should be used on. Possible values are "first", "last", "odd", "even", "all"; a comma-separated list of values is allowed as well.
font	Set the default font face of the header.
pointsize	Set the default point size of the header font.

<footer>

Define a page footer for one or several pages. A footer is a content area in its own right and may contain almost any other element.

Attributes	
location	Specify which pages the footer should be used on. Possible values are "first", "last", "odd", "even", "all"; a comma-separated list of values is allowed as well.
font	Set the default font face of the footer.
pointsize	Set the default point size of the footer font.

<variable>

Variables can be used in headers and footers to insert data from a small list of pre-defined choices, e.g. page number or current date. The *type* attribute is mandatory and specifies which data to insert. The <variable> element has no content.

Attributes	
type	Possible values:

Attributes	
	• pagenumber - insert the current page number • pagecount - insert the page count of the report • textdate - insert the current date as in the example. Names will be abbreviated and localized; the order of elements, however, will not be localized. Example: "Thu Aug 18 2005" • isodate - insert the current date according to ISO 8601. Example: "2005-08-18" • localedate - insert the current date formatted according to the locale in use • texttime - insert the current time-of-day like in the example. Example: "13:42:59" • isotime - insert the current time-of-day according to ISO 8601. Example: "13:42:59" (same as texttime) • localetime - insert the current time-of-day according to the locale in use

<table>

A table.

Attributes	
background	Set the background color.
bgred, bggreen, bgblue	Set the RGB color components of the background separately. You cannot set one or two components only; all three must be present.
border	Set the table to show borders.
width	Set the width of the table in millimeters or percentages of the page width.
inline	Add the table as an inline element.

Attributes	
alignment	Set the horizontal alignment; possible values are "left", "right" and "hcenter". This is only applicable if the inline attribute is not present.
model	The model key for the model that supplies the data - see the API documentation of KDReport::associateModel(). If you are not a programmer, we recommend that you pick a unique and descriptive name for the data source to be used.

The following attributes are only available if the *model* attribute is set.

Attributes for Model Mode	
header-background	Set the header background color. This takes the same values as the common *color* attribute.
verticalHeaderVisible	Set to "false" to hide the header row.
horizontalHeaderVisible	Set to "false" to hide the header column.

The following attributes are only available if the *model* attribute is not set. Data should then be provided using <cell> elements inside the table element.

Attributes for Non-Model Mode	
headerRowCount	Set the number of header rows, default is zero ###.
cellpadding	Set the padding of cells in millimeters. The default is 0.5mm.

<cell>

A table cell; it is a content area in its own right and may contain almost any other element. It may only occur inside a table element.

Attributes	
row	The row number of the cell (mandatory).
column	The column number of the cell (mandatory).
rowspan	The number of rows the cell should span.
colspan	The number of columns the cell should

Attributes	
	span.
background	Set the background color.
bgred, bggreen, bgblue	Set the RGB color components of the background separately. You cannot set one or two components only; all three must be present.

<chart>

A chart generated by KDAB's KD Chart; KD Chart is required for this element to work.

Attributes	
model	Mandatory attribute. The model key for the model that supplies the data - see the API documentation of KDReport::associateModel(). If you are not a programmer, we recommend that you pick a unique and descriptive name for the data source to be used.
background	Set the background color.
bgred, bggreen, bgblue	Set the RGB color components of the background separately. You cannot set one or two components only; all three must be present.
width	Set the width in millimeters or percentages of the page width.
height	Set the height in millimeters or percentages of the page height.
inline	Add as inline element
alignment	Set the horizontal alignment; possible values are "left", "right" and "hcenter". This is only applicable if the inline attribute is not present.

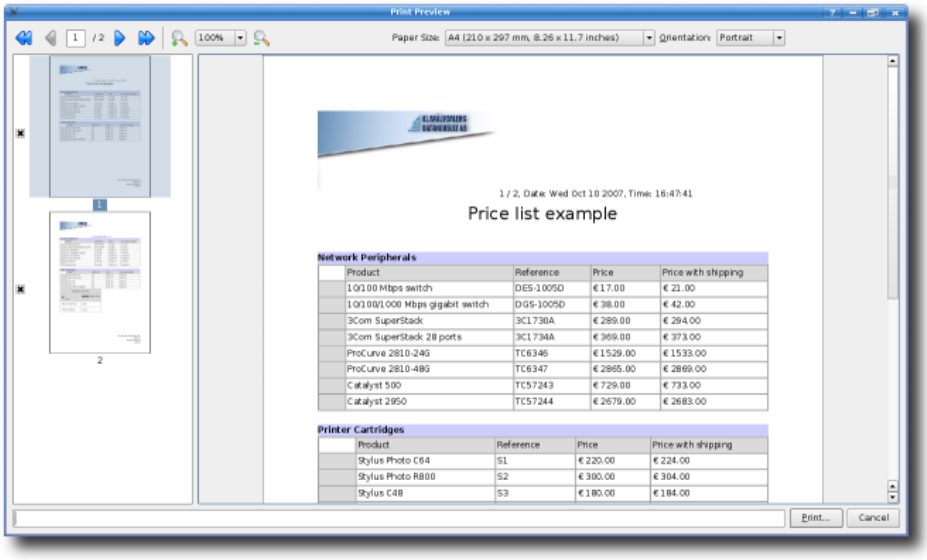
Advanced Usage

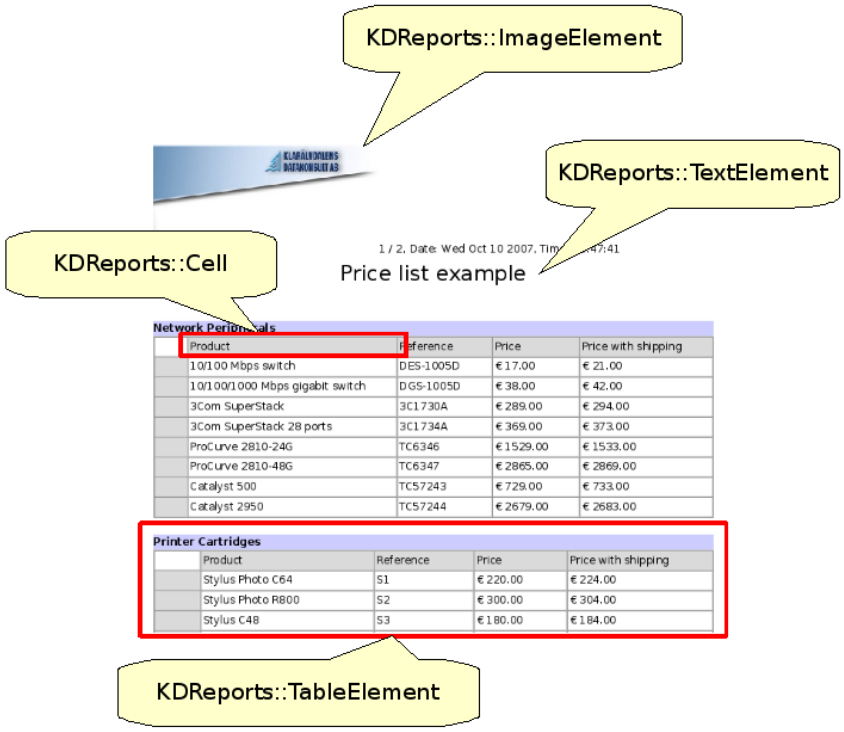
XML templates are very static by nature. In some cases, however, you might see the need for determining at runtime whether an element should be shown or hidden, for modifying an element's attributes based on some condition, or for dynamically inserting additional elements. All this can be done by installing an "XML element handler" on the report. See the KDReports::XmlElementHandler API documentation for more details.

Examples

You will find some other examples besides Hello World shipped with your KD Reports, from which you can learn how to show items with a table model that have been fetched from a database in the Database example, read from a CSV file in the PriceList example or simply created by the program, as shown in the Letters example. The PriceList and the Database examples are also available as XML, so you can explore how easy it is to create reports using XML.

Figure 4.1. Pricelist example and its elements explained





API Documentation

A complete overview of all classes and methods is available in the KD Reports API Documentation.

Chapter 5. Version History

Version 1.2

- A **ReportViewer** application, where you can open XML reports for previewing. This can be particularly useful while writing the XML file. *ReportViewer* lets you edit text values and reports to make the preview more realistic.
- Export to HTML.
- **XmlElementHandler** interface for making dynamic changes to the report while loading an XML template.
- Support for text and html elements whose text comes from a model.

Example:

<text model="table1" row="1" column="0"/>

<html model="table1" row="1" column="1"/>

- Support for merged cells in auto tables: specify the row/column span in the model's `span()` method.
- Support for configurable column widths in tables; see `AbstractTableElement::setColumnWidths`. This also leads to faster table breaking.
- Improved error handling when parsing XML files. `KDReports::Report::loadFromXML` now takes an optional `ErrorDetails` pointer, which will be filled with the error message from the parser.
- Support for changing named values with `associateTextValue`, even after the report has been constructed.
- Support for definition tab positions (most useful for aligning numbers along their decimal point).
- Support for showing the icon after the text in auto-table cells and headers:

  return `Qt::AlignRight` for the role `KDReports::AutoTableElement::DecorationAlignmentRole`.

  Also, a space is inserted between the icon and the text, in table cells.
- Support for creating a `KDReports::Report` and its elements from a non-GUI thread (requires Qt >= 4.4). Charts and pixmaps cannot be used there.

- Shortcuts for changing pages in the preview dialog: **Ctrl**+**PageUp** / **Ctrl**+**PageDown**
- Support for being compiled as a framework, on the mac.

Performance:

- Printing and previewing is much faster than in previous KDReports releases (8 times faster for a 27-pages document- this improvement being even bigger for longer documents).
- Calling `pageCount()` multiple times during report creation no longer slows down the report creation (for instance, when building a table of contents in a table model).

Bugfixes:

- **<text id="...">fallback</text>** and **<html id="...">fallback</html>** will now show “fallback” if the id is not known, as this behavior was in the documentation already.
- **<text>** **</text>** (whitespace-only text nodes) are now processed correctly.
- **Table breaking:** fixed `scaleTo(1, N)` so that it scales the font down until everything fits as expected.
- **Default font**: the font set in the **<report>** element or with `Report::setDefaultFont` wasn't applied to auto tables nor to headers and footers.
- **Auto tables**: the background color from the model now fills the entire cell.
- **Auto tables**: support models with incremental data loading (`fetchMore()`), like `QSqlTableModel`.
- **Preview dialog**: hide “Paper Size” and “Orientation” labels when `setPageSizeChangeAllowed(false)` is called.
- Preserve page breaks after autotables when calling `regenerateAutoTables()`.

Chapter 6. Appendix

Obtaining KD Reports

KD Reports is sold by KDAB under the terms of the KDAB Commercial License. For pricing and sales inquiries, please visit the KDAB website at www.kdab.com. On a case-by-case basis, KDAB may give licenses to Free Software projects, under the terms of the GPL.

Support

Commercial license holders are entitled to support by contacting support@kdab.com. The KDAB support staff will provide bug reporters with the tracking numbers of their support requests and will inform them when solutions to their reported problems are available.

How to Contact KDAB

Please contact KDAB via e-mail to info@kdab.com. Other options are:

- Phone: +46-563-540090
- Fax: +46-563-10625
- Our postal address is:

```
Klarävdalens Datakonsult AB
Box 30
SE-683 21 Hagfors
Sweden
```

License

```
KD Reports COMMERCIAL LICENSE AGREEMENT
FOR COMMERCIAL VERSIONS
Version 1.0

Copyright of this license text (C) 2001 Trolltech AS and (C) 2002-2009
Klarälvdalens Datakonsult AB.  All rights reserved. License text used
with kind permission of Trolltech AS. The software and accompanying
material is Copyright (C) 2009 Klarälvdalens Datakonsult AB.

This non-exclusive non-transferable License Agreement ("Agreement") is
between you ("Licensee") and Klarälvdalens Datakonsult AB (KDAB), and
pertains to the Klarälvdalens Datakonsult AB software product(s)
accompanying this Agreement, which include(s) computer software and
may include "online" or electronic documentation, associated media,
```

and printed materials, including the source code, example programs and the documentation ("Software").

COPYRIGHT AND RESTRICTIONS

1. All intellectual property rights in the Software are owned by KDAB and are protected by Swedish copyright laws, other applicable copyright laws, and international treaty provisions. KDAB retains all rights not expressly granted. No title, property rights or copyright in the Software or in any modifications to the Software shall pass to the Licensee under any circumstances. The Software is licensed, not sold.

2. By installing, copying, or otherwise using the Software, you agree to be bound by the terms of this agreement. If you do not agree to the terms of this Agreement, do not install, copy, or otherwise use the Software.

3. Upon your acceptance of the terms and conditions of this Agreement, KDAB grants you the right to use the Software in the manner provided below.

4. KDAB grants to you as an individual a personal, nonexclusive, non-transferable license to make and use copies of the Software for the sole purposes of designing, developing, testing and distributing your software product(s) ("Applications"). You may install copies of the Software on an unlimited number of computers provided that you are the only individual using the Software. If you are an entity, KDAB grants you the right to designate one, and only one, individual within your organization who shall have the sole right to use the Software in the manner provided above.

5. The license granted in this Agreement for you to create and distribute your own Applications is subject to all of the following conditions: (i) all copies of the Applications you create must bear a valid copyright notice, either your own or the copyright notice that appears on the Software; (ii) you may not remove or alter any copyright, trademark or other proprietary rights notice contained in any portion of the Software; (iii) you will indemnify and hold KDAB, its related companies and its suppliers, harmless from and against any claims or liabilities arising out of the use and/or reproduction of your Applications; (iv) your Applications must be written using a licensed, registered copy of the Software; (v) your Applications must add primary and substantial functionality to the Software; (vi) your Applications may not pass on functionality which in any way makes it possible for others to create Applications with the Software; (vii) your Applications may not compete with the Software; (viii) you may not use KDAB's or any of its suppliers' names, logos, or trademarks to market your programs, except to state that your program was written using the Software.

6. WARRANTY DISCLAIMER
THE SOFTWARE IS LICENSED TO YOU "AS IS". TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KDAB ON BEHALF OF ITSELF AND ITS SUPPLIERS, DISCLAIMS ALL WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT WITH REGARD TO THE SOFTWARE.

7. LIMITATION OF LIABILITY
IF, KDAB'S WARRANTY DISCLAIMER NOTWITHSTANDING, KDAB IS HELD LIABLE TO YOU BASED ON THE SOFTWARE, KDAB'S ENTIRE LIABILITY TO YOU AND YOUR EXCLUSIVE REMEDY SHALL BE, AT REPAIR OR REPLACEMENT OF THE SOFTWARE, PROVIDED YOU RETURN TO KDAB ALL COPIES OF THE SOFTWARE AS ORIGINALLY DELIVERED TO YOU. KDAB SHALL NOT UNDER ANY CIRCUMSTANCES BE LIABLE TO YOU BASED ON FAILURE OF THE SOFTWARE IF THE FAILURE RESULTED FROM ACCIDENT, ABUSE OR MISAPPLICATION, NOR SHALL KDAB UNDER ANY CIRCUMSTANCES BE LIABLE FOR SPECIAL DAMAGES, PUNITIVE OR EXEMPLARY DAMAGES, DAMAGES FOR LOSS OF PROFITS OR INTERRUPTION OF BUSINESS OR FOR LOSS OR CORRUPTION OF DATA.

9. This Agreement may only be modified in writing signed by you and an authorized officer of KDAB. All terms of any purchase order or other ordering document shall be superseded by this Agreement.

10. This Agreement shall be construed, interpreted and governed by the laws of Sweden, the venue to be Sunne Tingsrätt.