



Programmers Manual

KD Reports



The contents of this manual and the associated KD Reports 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 Reports and the KD Reports 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 is KD Reports	1
Installation	1
Feature Summary	1
The Structure of this Manual	2
2. Programmers manual	
A minimalistic report: Hello World	3
A Letter from KDAB	4
Tables with formatting and page-break: PriceList	6
Using SQL Data in Reports: A Database Example	7
3. XML templates	
Hello world as XML	9
The PriceList Example as XML	10
4. References	
A brief look at the API	11
Examples	11
API Documentation	12
5. Appendix	
Obtaining KD Reports	13
Support	13
How to contact KDAB	13
License	13



List of Figures

4.1. Pricelist example and its elements explained 11

Chapter 1. Introduction

► What is KD Reports

KD Reports, put simply, is a Qt tool that lets you easily create printable reports by providing all of the necessary features for a variety of applications.

Reports can be created programmatically, using an easy to use C++ API, or they 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.

KD Reports targets C++ programmers who use Qt in their applications. The following reference assumes familiarity with the C++ programming language and the basic concepts of Qt.

► Installation

For licensed customers, KD Reports comes as a source code package that is built during installation. Experienced developers are able to integrate this source into their own builds. For binary installations, KD Reports also provides a graphical installer.

► Feature Summary

Software engineers using KD Reports will find everything necessary to implement the reporting modules of their application. Reports are simply C++ objects that are put together in the application source code.

Using XML templates, the creation of the report content can easily be assisted by non-technical staff. The look and structure of the report can be created as XML, with placeholders that are filled in by the application at runtime.

Reports can contain tables that are generated from Qt's abstract item models. By way of the QSql module, it is possible to retrieve report table data from SQL databases. Because reports are generated within application logic, applying higher-level calculations is done within the same programming environment as in the rest of the application.

Charts can be added to reports by making use of KDAB's KD Chart package. It provides a wealth of different 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.

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

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

To structure the reports, groups of element can be arranged horizontally or in vertically. Page breaks are performed automatically, and printing on endless paper is supported.

The Structure of this Manual

This manual contains the following main parts:

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

Chapter 2. Programmers manual

In the following chapter, the KD Reports API will be introduced. The examples delivered with KD Reports will be used as the illustrations. Let's start with the simplest one, HelloWorld.

► A minimalistic report: Hello World

The first example shows what the skeleton of a report looks like. Of course, it does not do much more than that:

```
...
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();
}
```

First, we need an object of class `KDReports::Report`. A report consists of a set of output pages. Here, we start filling the report with a title element, containing the famous piece of text, "Hello World!" A second text element, left aligned, is shown below it. For better looks, a vertical space of 20mm is added between the two text elements. This is all that is needed to create a first report—it can now be printed, saved to PDF, or shown in a preview window. The latter is exactly what the last two lines of code do - show a preview dialog to the user that displays the generated report.

In the next section, a slightly more complex report is generated—a letter.

A Letter from KDAB

This time, a fully featured letter will be generated. It consists of headers and footers, multiple, differently aligned text elements, a table generated from an Qt Model/View Framework model, and 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, which is the class with which KD Reports expects to fill a table.

```
...
// Add From: and To: information:
KDReports::TextElement fromElement;
fromElement << "From:\n"
    << "Klarälvdalens Datakonsult AB\n"
    << "Rysktorp\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 app?\n"
    "Look no further!" ) );
titleElement.setPointSize( 14 );
report.addElement( titleElement, Qt::AlignCenter );
...
```

Now, the "From:" and "To:" text elements are added. Line breaks can be added to text elements as needed, and the text can 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 with centered alignment to the report.

```

...
// 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 a nice package "
"providing this kind of functionality to complement Qt was looked "
"for by many. 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 );
...

```

As the next step, the letter body is added. It consists of two blocks of text, which are added with justified alignment. In between, the table is added 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 would not 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.net | +46-563-540090";
footerText.setPointSize( 8 );
report.footer().addElement( footerText, Qt::AlignCenter );
...

```

The footer shows the first usage of `KDReports::HtmlElement`. `HtmlElement` can contain

text in HTML markup, here it is used to create a horizontal ruler. Then one more text element, a little smaller this time, and centered. This finishes the basic structure of the letter.

In the following example, garnishing the report with different types of tables will be demonstrated.

► Tables with formatting and page-break: PriceList

Tables are a common element of reports. The price list example demonstrates 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`, and therefore `KDReports::AutoTableElement` can use it directly. The table is set to a width of 100%. This matches nicely with justified text paragraphs 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(
    KDRReports::TextElement( "4 days" ) );
tableElement.cell( 3, 0 ).addElement(
    KDRReports::TextElement( "Printer Cartridges" ) );
tableElement.cell( 3, 1 ).addElement(
    KDRReports::TextElement( "3 days" ) );

report.addElement( tableElement );
...

```

Here, a table is created directly from `KDRReports::Cell` objects. This way, it is not necessary to provide a data model. Also, it allows the addition of elements like pictures to table cells, and to specify more detailed formatting for the individual cells. Model driven tables can 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. This example shows how the content of a sample in-memory SQLite database can be used in reports. All databases supported by Qt's `QtSql` module, including Oracle, MySQL, and many others, can be used in reports.

```

...
// 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 an in-memory sqlite database. It will be used to initialize a `QSqlTableModel`. The model in turn 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, an `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

For rapid prototyping and easier maintenance, it may be easier to write report templates in some editable format independent of the program's source code. This is where XML templates come in.

▶ Hello world as XML

The details of the XML syntax are described in the reference. Here, we 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! Later examples 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" is used twice in the xml, both places get the right value
report.associateTextValue( "table1", "Network Peripherals" );
report.associateTextValue( "table2", "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 or filled 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 programmatically 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

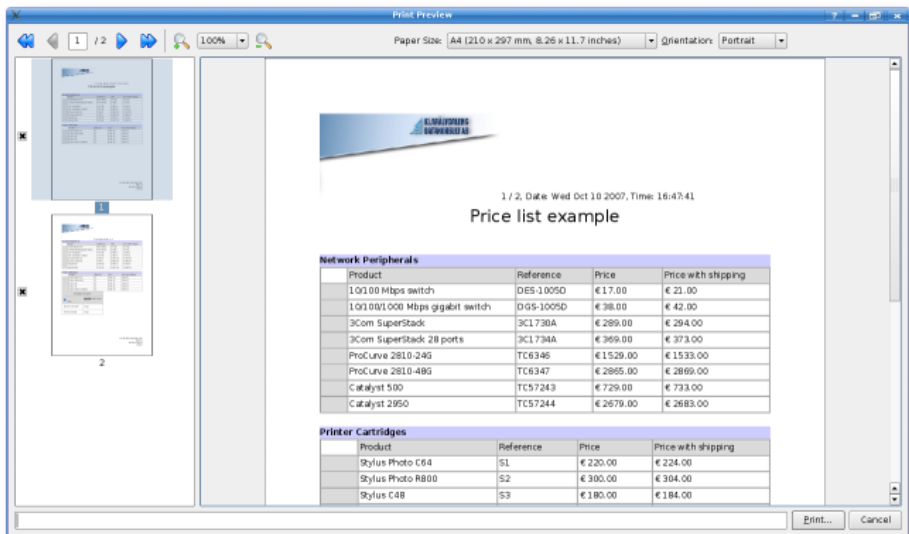
▶ A brief look at the API

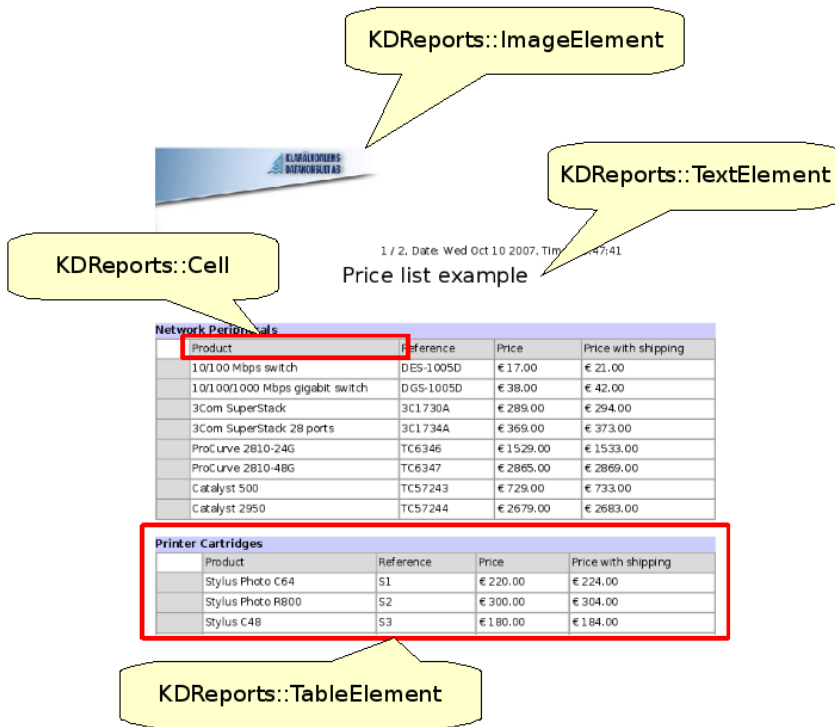
KD Reports offers classes for embedding content such as images, text, HTML and charts, as well as ways for arranging them. Use the `KDReports::AutoTableElement` for the Model-View approach, or just arrange your elements in a `KDReports::TableElement` with cells and a header. The content can be added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

▶ Examples

Besides Hello World you will find some examples shipped with your KD Reports where you can learn how to show items with a table model, either fetched from a database in the Database example, read from a CSV file in the PriceList example or just 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 off all classes and methods is available in the KD Reports API Documentation.

Chapter 5. 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.net. The KDAB support staff will provide bug reporters with information on the tracking numbers of their support request and will inform the reporter when a solution to the reported problem is available.

▶ How to contact KDAB

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

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

```
Klarävdalens Datakonsult AB 30  
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) 2002-2007  
Klarälvdalens Datakonsult AB. All rights reserved. License text used  
with kind permission of Trolltech AS. The software and accompanying  
material is Copyright (C) 2007 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.