



---

report generator system

# NCReport Definitive Guide (Beta)

Norbert Szabo

Version 3.4, Mar 2025 - NCReport v2.30

# Table of Contents

Basics .....	3
1. About NCReport in general .....	4
1.1. A short history .....	4
1.2. What is NCReport? .....	4
1.3. Features .....	4
1.4. Why NCReport? .....	5
1.5. How NCReport does work .....	5
1.6. Working schema .....	5
1.7. Report template file .....	6
2. Installing NCReport .....	7
2.1. Requirements .....	7
2.2. NCReport has been tested with: .....	7
2.3. Install Binary package on Linux .....	7
2.4. Install (commercial) source package under Linux .....	7
2.5. Install binary package on Windows .....	8
2.6. Install (commercial) source package under Windows .....	8
2.7. Contents of the installation directory .....	9
2.8. Acknowledgements .....	9
3. Getting started .....	10
3.1. Creating a basic report .....	10
3.2. Beginning with a new report .....	10
3.3. Testing report in Designer .....	21
3.4. Variables and Groups .....	22
3.5. Integrating NCReport into a Qt application .....	27
4. Report Items .....	31
4.1. Static Text Item .....	31
4.2. HTML (Rich) Text .....	31
4.3. Data Field .....	31
4.4. Line .....	32
4.5. Rectangle .....	32
4.6. Ellipse .....	32
4.7. Image .....	32
4.8. Barcode .....	32
4.9. Table View / Model .....	32
4.10. Cross Table .....	32
4.11. Custom Graphics Content .....	33
5. Parameters .....	34
5.1. Parameter Syntax .....	34

5.2. Testing Parameters . . . . .	34
5.3. API code: Passing parameters to the report . . . . .	35
Designer . . . . .	36
6. Getting Started with NCRReport Designer . . . . .	37
6.1. Launching Designer . . . . .	37
6.2. The User Interface . . . . .	37
6.3. NCRReport Designer Main Window . . . . .	38
6.4. Geometry editor . . . . .	38
6.5. Data Source Tree . . . . .	39
6.6. Field Expression Builder . . . . .	40
6.7. Designing a report . . . . .	41
6.8. Connecting to database from Designer . . . . .	42
6.9. Beginning a new report . . . . .	43
6.10. Report sections . . . . .	44
6.11. Setting up page and report options . . . . .	46
6.12. Adding data sources . . . . .	47
6.13. Assigning data source to the Detail . . . . .	49
6.14. Adding report items . . . . .	50
6.15. Adding total variable field . . . . .	60
6.16. Other items . . . . .	61
6.17. Adjustment and formatting . . . . .	61
6.18. Report is ready . . . . .	61
6.19. Adding Variables for Totals . . . . .	62
6.20. Running the report . . . . .	63
Advanced Features . . . . .	66
7. Data/Script Expressions . . . . .	67
7.1. Using references in expressions . . . . .	67
7.2. References in templates . . . . .	68
7.3. Reference examples . . . . .	68
7.4. Testing Field Expression . . . . .	68
7.5. Field expression . . . . .	68
7.6. Result of field expression . . . . .	69
7.7. Print When Expressions . . . . .	69
7.8. Testing Print when expression . . . . .	70
7.9. Print only when expression is true condition . . . . .	70
7.10. Templates in Fields and Texts . . . . .	70
7.11. Script expressions in special locations . . . . .	71
7.12. Data Source Functions . . . . .	71
7.13. Data Source related (meta) functions . . . . .	71
7.14. Data Source Column related (Value) functions . . . . .	73
8. Script Editor . . . . .	75

8.1. Script ID .....	75
8.2. Script Definition .....	75
8.3. Available Buttons .....	75
9. Data Formatting .....	77
9.1. Text formats .....	77
9.2. Numeric formats .....	77
9.3. Date Formats .....	78
10. Zones .....	81
10.1. Zone ID in property dialog .....	81
10.2. Zones in Design mode .....	81
11. Dynamic data driven size and position .....	83
11.1. Dynamic position and size settings .....	83
12. Dynamic data driven shape style .....	84
12.1. Dynamic style settings .....	84
13. Page Breaks .....	85
13.1. Detail page break condition .....	85
13.2. Group page break condition .....	85
13.3. Report item page break .....	85
13.4. Report header page break .....	85
14. Text Document printout mode .....	86
14.1. Steps of usage .....	86
14.2. Text Document printout report example .....	86
15. Data Relation system .....	87
15.1. Defining a parent data source .....	88
15.2. Defining child data sources .....	88
15.3. Setting up the detail section .....	89
15.4. Designing the report .....	89
15.5. Sub-query report example in Designer .....	89
15.6. Result of a sub-query report example .....	90
15.7. Changes in 2.13 version .....	91
16. Double pass mode .....	93
16.1. Setting double pass mode .....	93
16.2. Example using of <b>pagecount</b> variable .....	93
17. Internationalization .....	94
17.1. Adding languages .....	94
17.2. Adding translations of Fields or Labels .....	94
17.3. Setting up the current language .....	95
17.4. Setting up the language .....	95
17.5. Setting up the current language from command line .....	95
18. Sub-Report iteration .....	96
18.1. Sub-Report data source .....	96

18.2. Reference to master data source .....	96
19. Table View Rendering .....	97
19.1. Adding TableView item .....	97
19.2. Table View Dialog .....	97
19.3. Setting the object references .....	98
19.4. Example .....	99
19.5. QTableView widget .....	99
19.6. QTableView table in print preview .....	99
19.7. Printing Item Model Based Table without QTableView .....	100
19.8. Custom Cell Content .....	100
19.9. Handle progress signal of table rendering .....	102
20. Cross-Tab Tables .....	103
20.1. Table Structure .....	103
20.2. Using Cross-Table in Designer .....	105
21. Conditional Formatting .....	108
21.1. Dynamic Style Tag Symbols .....	108
21.2. Editing Style Code in Designer .....	109
21.3. Default Style .....	109
22. General TEXT output .....	110
22.1. Text template manager tags .....	110
22.2. Text template tags .....	110
22.3. Examples .....	111
23. Batch Report Mode .....	112
24. Special Detail Sections .....	113
24.1. Sub (Detail) Sections .....	113
24.2. Adding a Sub-Section .....	113
24.3. Example Sub-Sections .....	113
24.4. Odd / Even Pages .....	114
24.5. Repeated detail by constant or dynamic value .....	115
Command Line Tool .....	116
25. Command line client .....	117
25.1. To run command line executable .....	117
25.2. Command line options .....	117
Using NCRReport API .....	119
26. Using NCRReport API .....	120
26.1. Project file settings .....	120
26.2. Initialize NCRReport class .....	120
26.3. Include directives .....	120
26.4. Creating NCRReport class .....	121
26.5. Connecting to SQL database .....	121
26.6. Setting the Report's source .....	122

26.7. Adding parameters .....	122
26.8. Running the Report .....	122
26.9. Running the Report by One Step .....	122
26.10. Running the Report in customized mode.....	123
26.11. Initializing Report's Output.....	123
26.12. Running the Report.....	124
26.13. Previewing Report.....	124
26.14. Deleting Report object.....	125
26.15. Using other data sources .....	125
26.16. Custom data sources .....	128
26.17. Custom items in NCReport.....	132
26.18. Batch report mode.....	134
Specification .....	135
27. Specification .....	136
27.1. Data sources .....	136
27.2. Report sections .....	139
27.3. Report Parameters.....	144
27.4. Variables.....	144
27.5. System Variables.....	144
27.6. Expressions .....	146
27.7. References in expressions .....	146
27.8. Using script expression in field:.....	146
27.9. Report items .....	147
27.10. Fields.....	148
27.11. HTML Text.....	151
27.12. Line .....	152
27.13. Rectangle .....	153
27.14. Image .....	154
27.15. Barcode.....	155
27.16. Graph or custom item.....	157

## Dedication

To all NCReport users,

This work is dedicated to you, the tireless explorers of the digital realm. Your curiosity drives innovation, your perseverance overcomes countless challenges, and your creativity brings life to technology in ways unimaginable. Whether you are troubleshooting, coding, designing, or simply navigating the complexities of the virtual world, your efforts shape the future of our digital landscape.

Thank you for your resilience, your passion, and your relentless pursuit of excellence. This is for you.

With friendly greetings,

Norbert Szabo

This book is designed to be the clear, concise, normal reference to the NCReport reporting software. This we can use as the official documentation for NCReport. We hope to answer, definitively, all the questions you might have about all the elements, features and entities in NCReport. It is essentially a comprehensive user documentation, definitive guide of the NCReport Reporting System. It also contains installation instructions, tutorials and information about the contents of the distribution. In particular, we cover the following subjects:

- The general nature of NCReport. We quickly get you up to speed on how the pieces fit together.
- How to create NCReport reports. Where should you start and what should you do?
- Understanding all of the report elements. Each element is extensively documented, including the intended semantics and the purpose of all its attributes. An example of proper usage is given for every element.
- How to run NCReport reports. After you've created one, what do you do with it?
- How to integrate NCReport library into a Qt application.

### Getting this Documentation

If you want to hold this book in your hand and flip through its pages, unfortunately it is not yet possible unless you print it for yourself. You can also get this book in electronic form, as PDF, from our web site: <https://www.ncreportsoftware.com/download>

### Getting Examples from This Documentation

All of the examples are included on our web site. You can get the most up-to-date information about this documentation from our web site: <https://www.ncreportsoftware.com/download>

### Request for Comments

Please help us improve future editions of this book by reporting any errors, inaccuracies, bugs, misleading or confusing statements, and plain old typos that you find. An online errata list is maintained at <https://tracker.ncreport.org> Email your bug reports and comments to us at [support@ncreportsoftware.com](mailto:support@ncreportsoftware.com)



In general a report generator software tool, often referred to as a report generator or reporting tool, is a computer program or software application designed to automate the process of creating, designing, and producing various types of reports. These reports can encompass a wide range of data and information, such as financial statements, business intelligence reports, marketing analytics, inventory reports, and more.

Key features and functionalities of report generator software tools typically include:

- **Data Extraction:** They can extract data from various sources, such as databases, spreadsheets, and external APIs.
- **Report Design:** Report generators often provide tools for designing the layout and formatting of reports, including the ability to add tables, charts, graphs, and other visual elements.
- **Template Creation:** Users can create report templates or use pre-designed templates to maintain consistency in the look and feel of reports.
- **Data Manipulation:** These tools allow users to manipulate and transform data before it is included in a report, which can involve calculations, filtering, sorting, and grouping.
- **Automation:** Report generators automate the process of report generation, saving time and reducing the potential for human error.
- **Export Options:** They support various output formats, including PDF, Excel, HTML, CSV, and more, so that users can choose the format that best suits their needs.
- **Integration:** Many report generators can integrate with other software applications and databases, enabling seamless data extraction and reporting.
- **Scheduling:** Some report generators allow for the scheduling of report generation at specific intervals, which is particularly useful for recurring reports.
- **Collaboration:** Report generators may facilitate collaboration among team members by allowing them to work on reports simultaneously and share them with others.

Report generator software is commonly used in business environments to streamline the reporting process and provide decision-makers with timely and accurate information. It can be particularly valuable in industries such as finance, healthcare, marketing, and manufacturing where regular reporting and data analysis are essential for operations and decision-making.

# Basics

This part covers the basic steps of the NCReport usage in practice.

# Chapter 1. About NCReport in general

## 1.1. A short history

NCReport's history is more than 10 years old. The project has been started in 2002 as a joint project of a Qt3 application and later the tool has become a unique GPL project. The reason why the system was started to plan the urgent needs of data printing as a very missing function in Qt/C++ programming environment. In 2007 the full project has been rewritten into a new commercial project by following the well formatted fully object oriented design concept. This version was named 2.0 version.

## 1.2. What is NCReport?

NCReport is a report generator, report writer tool, report engine with GUI designer primarily for Qt applications, though it is by no means limited to Qt environment. The software tool enables applications to print data driven reports, tables, lists, rich text documents or even any paginated graphical contents from one or more data sources. The system consists of at least two parts: Report engine and designer GUI application. The report engine is also available as command line executable. The report engine can be used and integrated into any Qt applications independently. NCReport has already been used and integrated by a growing community of commercial users and professionals.

## 1.3. Features

NCReport provides the following features and functions

- XML report definition (report template)
- Metric based, band oriented system
- Wide range of data source types: SQL Database, Text, `QAbstractItemModel`, XML, any user defined data source class
- Output formats: Preview, Printer, Postscript, PDF, SVG, Image, HTML, Text
- Fast private preview window system
- Internal or external SQL database connections
- Items: Label (simple text), Field, HTML/Rich text, Line, Rectangle, Ellipse, Image, Barcode, Custom content item
- Page header/footer
- Report header/footer
- Unlimited level of grouping with group headers and footers
- Variables for totals and aggregate functions, system variables
- Static and dynamic images
- Static and dynamic HTML contents

- Barcode rendering with at least 50 types of available barcodes thanks to the Zint barcode library.
- Template mode, expression and script evaluations in fields
- Parameters from application side
- Zones
- HTML Text Document printout mode
- Pure Qt4/Qt5/Qt6 compatible code (Qt4.5 - Qt6.5)
- Conditional Field or Label formatting

## 1.4. Why NCReport?

Modern software applications often use various data sources and SQL databases. In most cases, they must have the ability to print or represent data in several output formats, making report generation a crucial feature. For data-center applications, the ability to generate reports is almost always required. If you want to enable your application to generate reports efficiently, NCReport is an excellent choice.

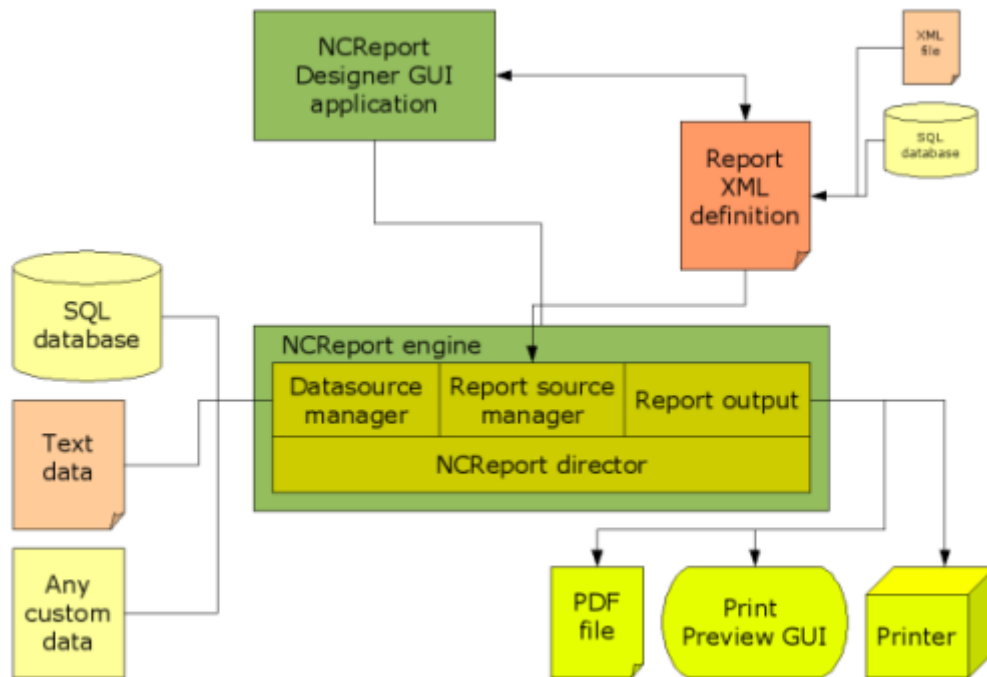
The NCReport project represents thousands of development hours and is continuously maintained. By integrating this reporting tool into your application, you will save a significant amount of development time and eliminate the need to develop any printing functions for your application. This is especially true for software written in C++ using the Qt multi-platform toolkit/library. Additionally, NCReport is a fully portable, native C plus plus multi-platform solution.

## 1.5. How NCReport does work

What does NCReport do exactly? In few words NCReport generates ready to print documents from raw data by a template. As a first step an XML report definition as a template must be created. This is a scenario for the report engine that describes what content must exactly render and how should it look like, where the data come from and so on. This report definition can come from local or remote file or from SQL database depending on what report source was defined. Report source manager is a part of the report engine that handles and loads report definition from it's origin. The report designer application as a separated GUI application designed for creating report XML definitions. When running a report first the report engine parses report definition and opens the specified data source(s). If SQL data source is defined a valid SQL database connection must be alive (in case of non built-in database connection is defined) After the data source(s) successfully opened SQL query is run by the report director. The report engine begins to process data row by row by specified data source assigned to the first detail section. While report is processing, the report director manages the rendering of different sections and the items inside. The result is rendered to the specified output such as: printer, print preview, postscript, PDF, SVG, Image, HTML, Text

## 1.6. Working schema

The following diagram illustrates how the report generator works in general.



## 1.7. Report template file

NCRReport uses Extensible Markup Language (XML) format for report definition. This is a universal standard file format, which simplifies also the human reading and processing the report definition templates.

# Chapter 2. Installing NCReport

## 2.1. Requirements

- Linux or any Unix like operation systems or <sup>TM</sup>Microsoft Windows or MacOS 10.5 or above.
- At least 512Mb of memory and a 1GHz CPU.
- 90Mb of free disk space

NCReport is officially supported on Windows XP/Vista/10/11, on Linux  $\geq 2.6$  and on MacOSX  $\geq 10.4$ . It is also possible to use it on other platforms that are supported by Qt but with limited support or without getting support from us.

## 2.2. NCReport has been tested with:

- Qt4.5-Qt6.4 under Windows XP/7/10/11
- Qt4.5-Qt6.4 under Linux (Ubuntu 10.04, 12.04, 16.04, 18.04, 20.04, 22.04)
- Qt4.7-Qt6.4 MacOS 10.6 - MacOS 12

## 2.3. Install Binary package on Linux

Make sure that the appropriate Qt version binaries are already installed on your Linux system. The required version is specified in the downloaded package. Unpack the NCReport Linux distribution to any directory you want: (i.e ncreport)

```
$ cd ncreport
tar xzvf ncreport2.x.x.tar.gz
$ cd ncreport/bin
```

NCReport binary files are intended to be used directly from the `ncreport-2.x.x/bin` directory. That is, you can start NCReport binaries by simply executing: To start the report designer:

```
$ ./NCReportDesigner
```

To start the command line report engine:

```
$ ./ncreport
```

After all you may want to add `ncreport_2.x.x/bin/` to your `$PATH`.

## 2.4. Install (commercial) source package under Linux

Make sure that GCC/G c compiler and the appropriate version of Qt development environment is

already installed on your Linux system. In addition, you need to be compiled/installed appropriate Qt's database drivers. Example reports mostly use QMYSQL and QSQLITE database drivers.

Unpack the NCReport Linux source package inside any directory you want:

```
$ cd directory
$ tar xzvf ncreport2.x.x.tar.gz
$ cd NCReport2.x.x
$ qmake
$ make
```

To start NCReport binary files just do the same as it's written in previous section.

## 2.5. Install binary package on Windows

It is strongly recommended to download and install one of the auto install `setup.exe` files. (NCReport\_2.x.x\_Windows.exe, NCReport\_2.x.x\_Windows\_MinGW.exe)

Just simply run the setup executable file and follow the setup wizard instructions. To start NCReport Designer use the Start menu

## 2.6. Install (commercial) source package under Windows

For license holders only. Make sure that a <sup>TM</sup>Windows C development environment is already installed on your Windows system. If you use Open Source version of Qt, the `GNU MinGW` compiler is contained in the `Qt SDK`. Current example shows the compiling procedure using `Microsoft Visual C` compiler

Make sure that the appropriate version of Qt development environment is already installed on your Windows system. In addition, you need to be compiled/installed appropriate Qt's database drivers. Example reports are mostly use QMYSQL and QSQLITE database drivers.

Simply unpack the downloaded `ncreport2.x.x.zip` or `.tar.gz` or `.7z` source package. Use a tool like Winzip or 7-Zip Windows has built-in support for `.zip` archives. To unzip the NCReport distribution inside any directory:

```
mkdir ncreport
cd ncreport
unzip ncreport_2.x.x_src.zip
qmake
nmake
```

## 2.7. Contents of the installation directory

- `/bin` Contains the NCReport executable files
- `/doc` Contains the User Guide and API documentation in html format
- `/sql` Contains the sql script files are required for some of example reports
- `/reports` Contains the sample reports for demonstrating NCReport features
- `/lib` Contains the binary library files (Unix/Linux only)
- `/testdata` Contains test files for demonstration purposes. `defaulttestdata.xml` file is used by Designer application for storing test parametersdata. If want to use it, please copy this file to `/bin` directory before starting NCReportDesigner.
- `/i18n` Contains internationalization files.
- `/images` Contains image files for a `sql_productlist_with_dynimages_demo.xml` test report
- `/src` Contains the source codes of NCReport system. The binary package contains only the source of demo and sample applications. The full source code is available for commercial license holders only.

## 2.8. Acknowledgements

On Windows, the NCReport installer is built using Inno Setup by <https://www.jrsoftware.org>, Jordan Russell's software. We highly recommend this excellent and free-to-use tool.



# Chapter 3. Getting started

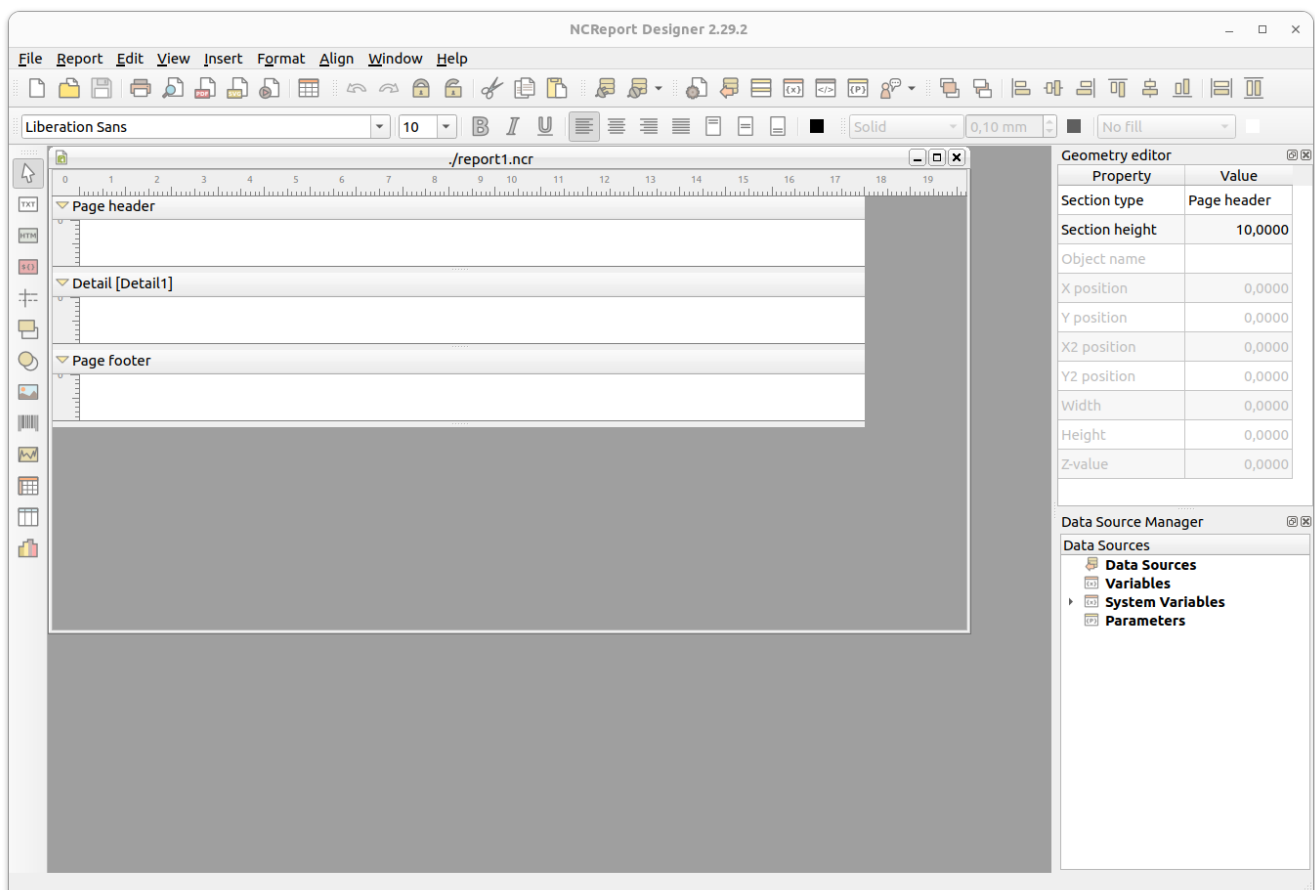
This chapter is intended to provide a quick introduction to NCRReport system. If you're already familiar with using the tool, you only need to skim this chapter. To work with NCRReport, you need to understand a few basic concepts of structured editing in general, and NCRReport, in particular. That's covered here. You also need some concrete experience with the way a NCRReport report definition is structured. That's covered in the next chapter.

## 3.1. Creating a basic report

At the very beginning we go through the first basic steps of creating a simple report. In our example we build a plain product price list report grouped by product category.

## 3.2. Beginning with a new report

Open the report designer GUI application and let's begin a new report by clicking [ **New** ] toolbar button or use **File** > **New** menu.



*A new empty report in the Designer*

### 3.2.1. Setting up page options

Page options of the current report can be specified in **Report and Page settings** dialog. Open the **Report** > **Report and page settings** In the report page settings dialog you can specify the following options:

## Report name

Type the name of the report. It's just an informative option, it's not used by report generator.

## Report type

There are two type of reports available. **Report** represents a normal report, **Text document** is a limited report mode. In this mode the report can contain HTML text items only. The generated report will be a paginated html document.

## File encoding

The encoding of the XML file. When user opens or saves the report definition file, this will be the default encoding. In most cases UTF-8 fulfils the requirements, but for special international characters you can choose any specified encoding.

## Default font

The font name and size are basically used for the text labels and fields in the whole report. Unique object settings may overwrite this option.

## Page size

The size of the page. The size names are listed in the combobox and their names are the standard size names. Currently the standard page sizes are supported.

## Background color

The background color of the report. This option currently is unused.

## Header and footer settings

The check boxes can be used to enable or disable page header/footer and report header/footer. To alter the height of these sections you may use spin boxes corresponding to their check boxes. You can also change these height properties by mouse dragging or by geometry editor

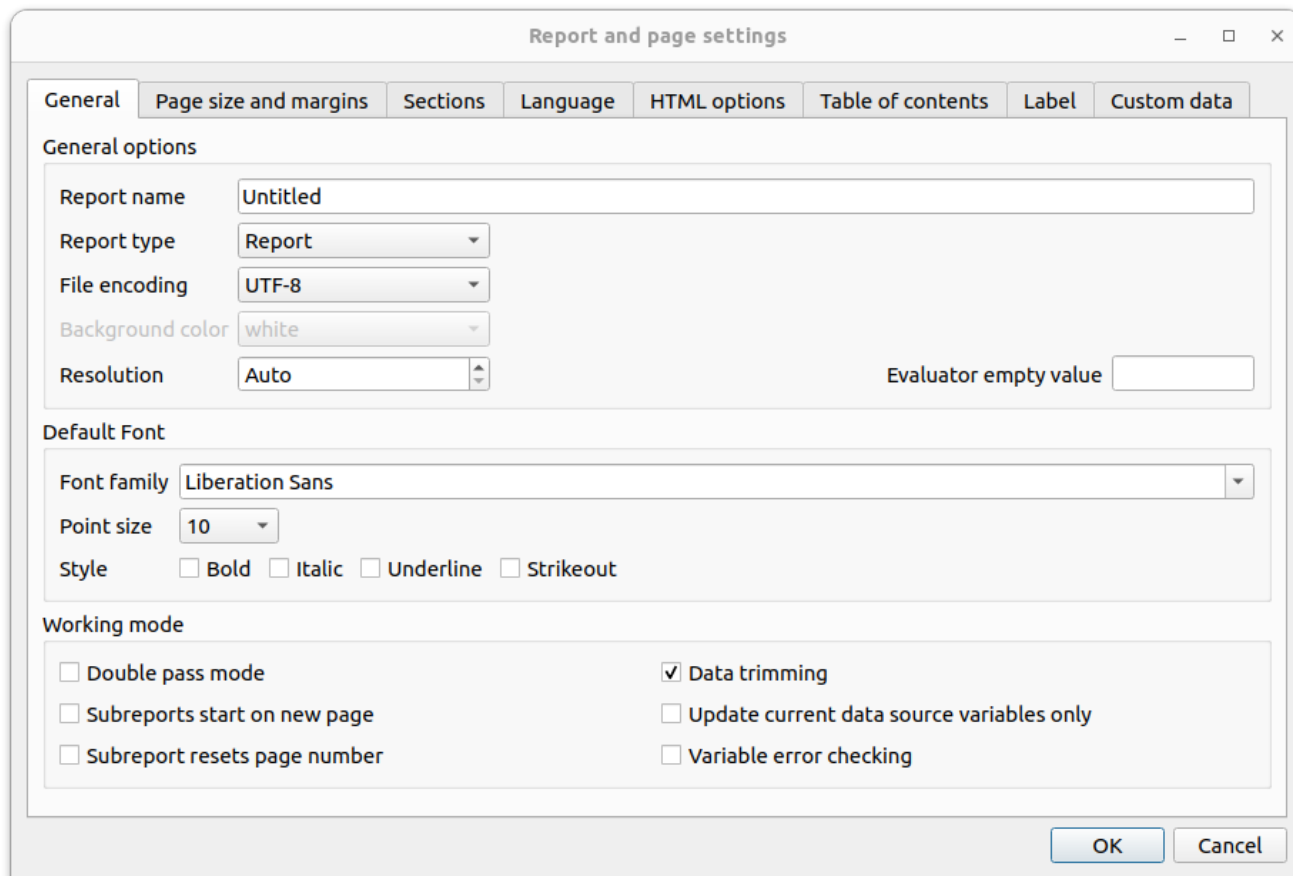
## Margins

margin properties represent the top, bottom, left and right margins of the page in millimeters. To alter the margin values just use the spin boxes.

## Orientation

This radio button option represents the orientation of the page, Portrait or Landscape orientation can be selected.

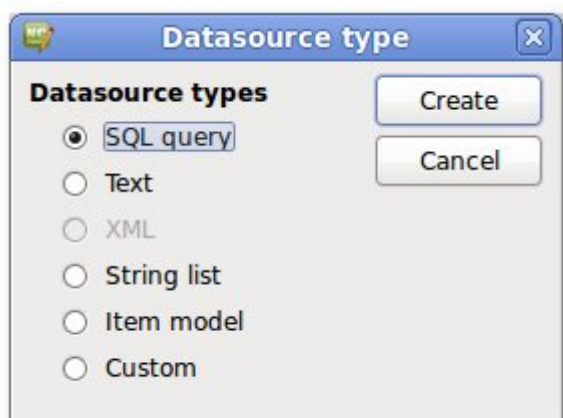
Specify the page's properties by this example and click **[ OK ]** button for saving data source settings. We add the report's name only, other default properties we don't change.



*Page settings dialog*

### 3.2.2. Adding a data source

First, you see an empty new report that contains a page header, a detail and a page footer sections by default. Before starting to add report items we define the data source that represents a definition where the data will come from. In our example the data source is a **Text**. To specify a data source in your report open the Report menu and select **Report > Data sources...** menu item. Then appears a dialog on you can add and or remove data sources. To add a new data source click the **[Add]** button in dialog and then select the **QStringList** data source type from the list of available data source types. **[Create]** button



In the data source dialog the following properties we specify:

## Data source ID

This ID is important for assigning data source to a detail section.

## Data source type

The type of the data source you've already chosen before.

## Location type

Location type is a property that describes where the data can be found. In this report we will use static **Text** which is a statically pasted text. The text will be saved into the report.

Specify the data source properties by this example and click btn::[OK] to save the data source settings.

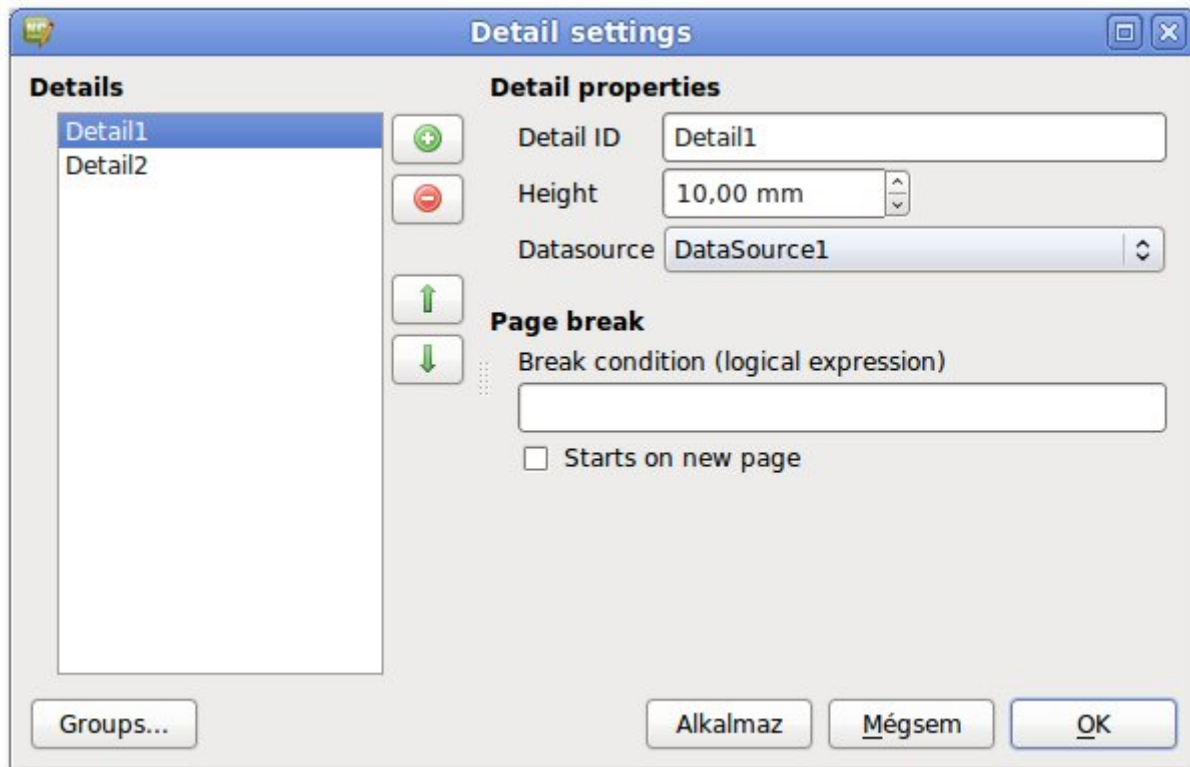
The screenshot shows a 'Dialog' window with the following components:

- Data sources:** A list box containing 'DataSource1', 'DataSource2' (selected), 'DataSource3', 'DataSource4', and 'DataSource5'. Below it are 'Add' and 'Remove' buttons.
- Data source properties:**
  - Data source type:** A dropdown menu set to 'SQL query'.
  - Opening/running role:** A dropdown menu set to 'Beginning of the report'.
  - Data source ID:** A text box containing 'DataSource2'.
  - Location type:** A dropdown menu set to 'Static'.
  - ☐ Empty datasource warning
- SQL Query / SQL Connection / Test result:** A tabbed interface with 'SQL Query' selected.
- Identification:**
  - Connection ID:** A text box containing 'db1'.
  - ☐ Forward only mode
  - Parent datasource ID:** An empty text box.
  - Test query:** A button with a blue arrow icon.
- Query:** A text area containing the SQL query:

```
SELECT id,name,address FROM customers
WHERE region = $P{region}
AND valid
```
- Buttons:** 'Mégsem' (Cancel) and 'OK' buttons at the bottom right.

### 3.2.3. Assigning data source to the detail section

To assign the data source we defined before, open the **Report > Report** menu and select **Report > Details and grouping...** menu item, then appears a dialog on you may manage the detail sections of the report. A default detail ID is Detail1, you may change it to whatever you want. Select the previously defined data source from **Report > data source** combo box.



Click [ **OK** ] button to apply detail settings.

### 3.2.4. Preparing some test data

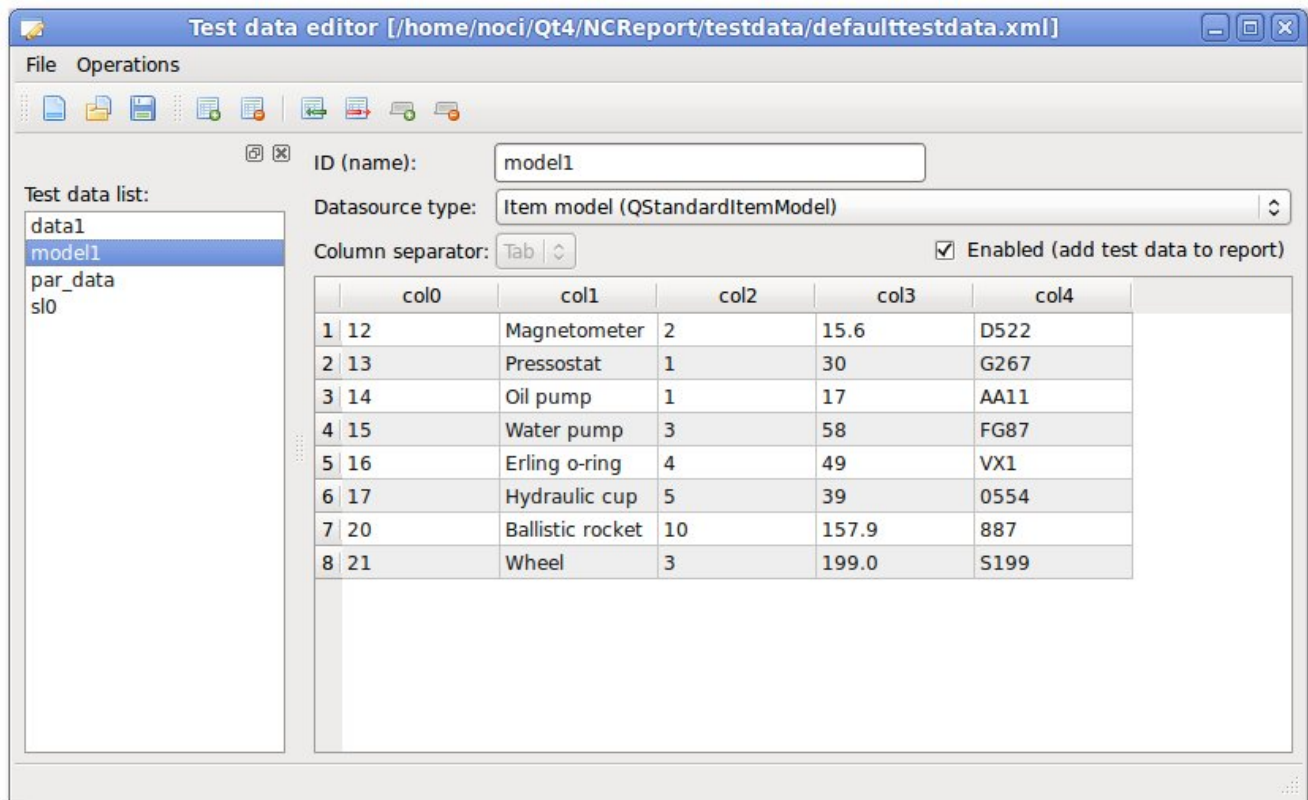
To try or test our report from Designer application we need some test data. This step is not necessary if you test the report from your application. Test data editor in designer makes easier the designing, debugging and testing of reports. In our example we create a simple product list included the following columns:

- **category** as 0. column
- **product** name as 1. column
- **product** code as 2. column
- **active** as 3. column
- **weight** as 4. column
- **price** as 5. column

A	Magnetometer	D54/78	yes	0.778	15.6
A	Pressostat	M542	no	2.547	30
B	Oil pump	CT-784	yes	1.510	17
B	Water pump	RF-800	yes	3.981	58
B	Erling o-ring	577874	yes	2.887	49
C	Hydraulic cup	HC55	no	0.435	39
C	Ballistic rocket	BV01	yes	1.260	157.9
C	Wheel	Q185/70	yes	25.554	199.0

To open the test data editor form open the **Report > Test data editor...** menu item, then appears a

dialog on you can edit and setup test data. There are three types of data source are available to use for testing. We need test data as **QStringList**, so we have to check the **Store as QStringList** check box and specify the ID.



Click [ **OK** ] button to apply detail settings. The checked types with their corresponding ID will effect the designer to add the test data to report test runner as a specified by ID data before running the report.

### 3.2.5. Using Geometric Editor

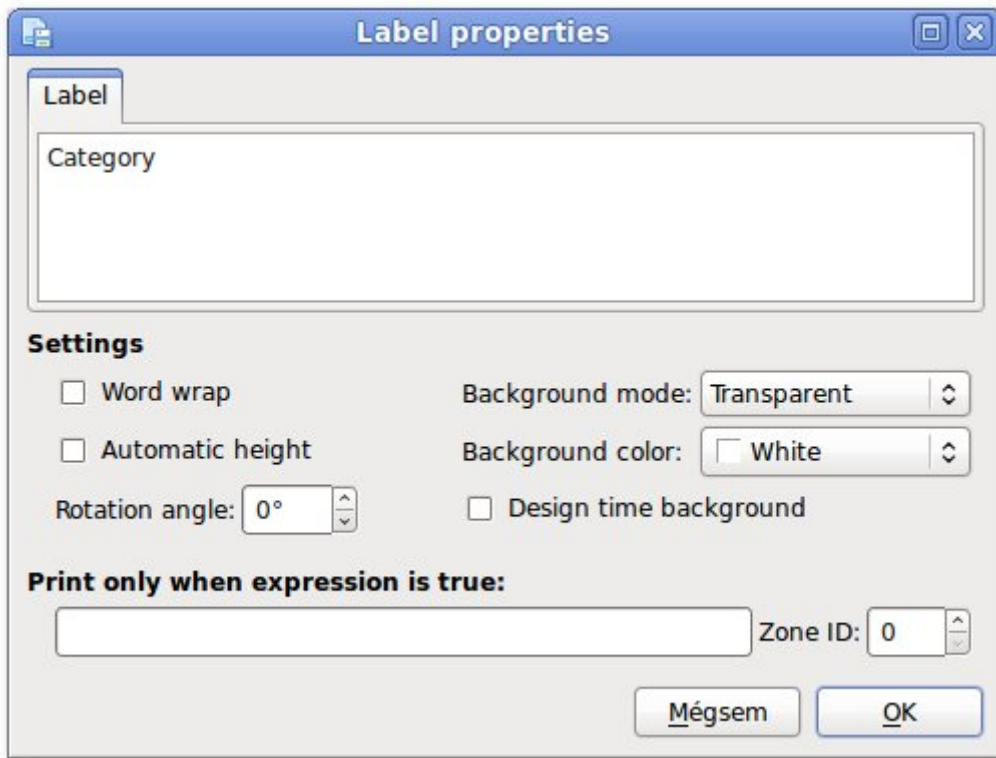
Geometry editor is a small property tool window in designer for showing or editing the position and size of objects in focus. To enable/disable Geometry editor just use **Report > View** menu and enable/disable **Report > Geometry editor** menu item. Then the tool window will appear in the right side. The current objects or sections are always activated by a mouse click. You can type the numeric size or position values into the spin boxes. Any changes made to the object's properties cause it to be updated immediately.

### 3.2.6. Designing page header section

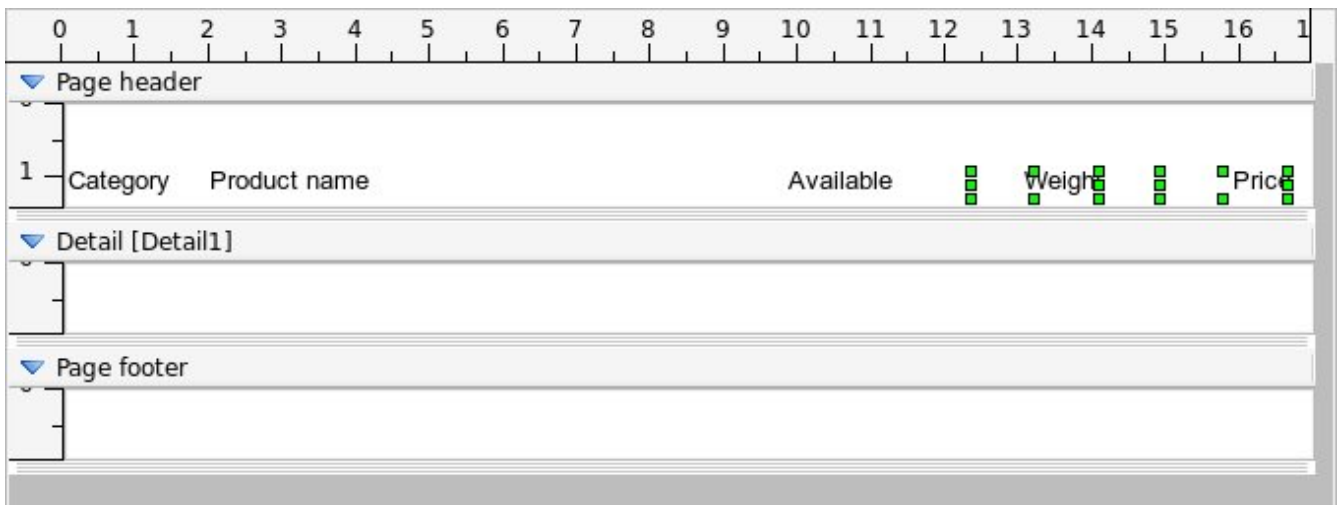
Page headers is used to contain page headings. First, we will add column titles as labels to page header section. Labels are simple texts. Label items are used to display descriptive information on a report, such as titles, headings, etc. Labels are static items, their value never change.

### 3.2.7. Adding Labels

Select the Label tool button or menu item in Tools menu. After that the cursor changes to a cross beam, then click in the page header of the report definition where you want the Label to be located. Doing so will create the Label object in that section and opens the Label settings dialog.



Add labels to page header for column titles and move them to positions by example. Then select "Weight" and "Price" (multiple selecting is available) and align them right by clicking [ **Right alignment** ] tool button.



### 3.2.8. Resize section

Increase the height of page header section by dragging the resizer bar at the bottom of the section. Another way for resizing to type **Section height** value in Geometry editor.

### 3.2.9. Drawing a line

To underline the labels, let's draw a Line by selecting the Line button in the tool bar or menu item in Tools menu. After that the cursor changes to a cross beam, then click in the section of the report definition where you want the line to be started and simply drag the line to the end position. (To move the line just drag and drop by left mouse button.)



Page header				
1	Category	Product name	Available	Weight
				Price

### 3.2.10. Designing Detail section

The core information in a report is displayed in its Detail section. This section is the most important section of the report since it contains the row by row data from the data source.

### 3.2.11. Adding Fields

Select the Field tool button or menu item in Tools menu. After that the cursor changes to a cross beam, then click in the detail section where you want the Field to be located. Doing so will create the Field object in that section and opens the Field settings dialog. The following properties must be specified:

#### Field source type

The combo box contains the possible sources from where the field can pull data.

#### Field column/expression

This property represents the name of the data column from where field's value is loaded from. To identifying data columns specify:

- the name of SQL column when using **SQL data source**
- the number of column **0,1,2...n** or **col0,col1,col2...coln**
- when using **StringList, ItemModel, StringParameter, Text data source**.

### 3.2.12. Data type

The field's base data type. The following data types are supported: **Text,Numeric,Date,Boolean**

The field's property dialog of the 1st column field:



Field properties

Field source type: Datasource Data type: Text

**Field**

DataSource1.name

**General** Numeric Date/Time Other

Description / title:

Template arg() string:

☐ Word wrapping Background mode: Transparent

☐ Automatic height Background color: User color 1

☐ Hide repeated values ☐ Design time background

Rotation angle: 0°

**Print only when expression is true:**

Zone ID: 0

Mégsem OK

Add Fields to Detail and move them to positions by example. Field column names are: **col0**, **col1**, **col2**, **col3**, **col4**, **col5** (alternative naming: **0**, **1**, **2**, **3**, **4**, **5**) Select **col4** and **col5** field item and align them right by clicking **[ Right alignment ]** tool button. After, in the field's dialog set Data type to **Numeric** and use the **Numeric tab** page to set number formatting properties.

**Field properties**

Field source type: **Datasource** Data type: **Numeric**

**Field**

DataSource1.price

**General** **Numeric** Date/Time Other

☒ Number formatting Decimal precision: **2**

☒ Use localized settings Field width: **0**

☐ Blank if value equals zero Format character: **f**

Fill character:

**Print only when expression is true:**

Zone ID: **0**

**Mégsem** **OK**

Resize the detail section to 4.5 mm height. After also a title label added to the page header section and formatted, the report should look like this:

**Page header**

**Product list example by category**

Category	Product name	Product code	Available	Weight	Price
col0	col1	col2	col3	col4	col5

**Detail [Detail1]**

**Page footer**

### 3.2.13. Designing page footer section

Page footer is usually used to display informations such as number of the page. In our example we only add two system variable fields: Application info and the current page number.

### 3.2.14. Adding System variable fields

Select the Field tool button or menu item in Tools menu. After that the cursor changes to a cross

beam, then click in the detail section where you want the Field to be located. Doing so will create the Field object in that section and opens the Field settings dialog.

### 3.2.15. Adding page number field

Specify the field's properties by this example:

The screenshot shows the 'Field properties' dialog box. At the top, 'Field source type' is 'System variable' and 'Data type' is 'Numeric'. The 'Field' tab is selected, showing the field name 'pageno'. Below are tabs for 'General', 'Numeric', 'Date/Time', and 'Other'. The 'General' tab contains the following settings: 'Description / title' is empty; 'Template arg() string' is 'Page %1'; 'Word wrapping' is unchecked; 'Automatic height' is unchecked; 'Hide repeated values' is unchecked; 'Rotation angle' is '0°'; 'Background mode' is 'Transparent'; 'Background color' is 'User color 1'; and 'Design time background' is unchecked. At the bottom, there is a section 'Print only when expression is true:' with an empty text box and a 'Zone ID' set to '0'. Buttons for 'Mégsem' and 'OK' are at the bottom right.

### 3.2.16. Adding application info field

Add again a new field to page footer and specify the field's properties by this, similar to the previous: Field source type: **System variable** Field column expression: **appinfo**

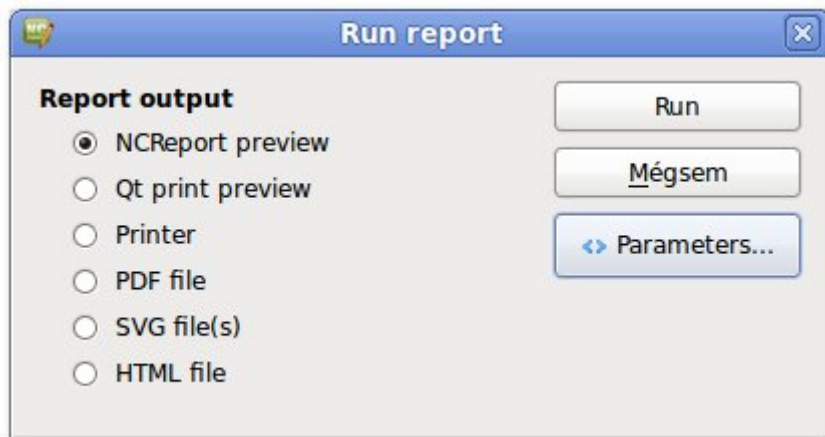
### 3.2.17. Resize section

Derease the height of page footer section by dragging the resizer bar at the bottom of the section. Another way for resizing to type **Section height** value in Geometry editor. After setting the alignments and moved fields to the right positions, the report should look like this:

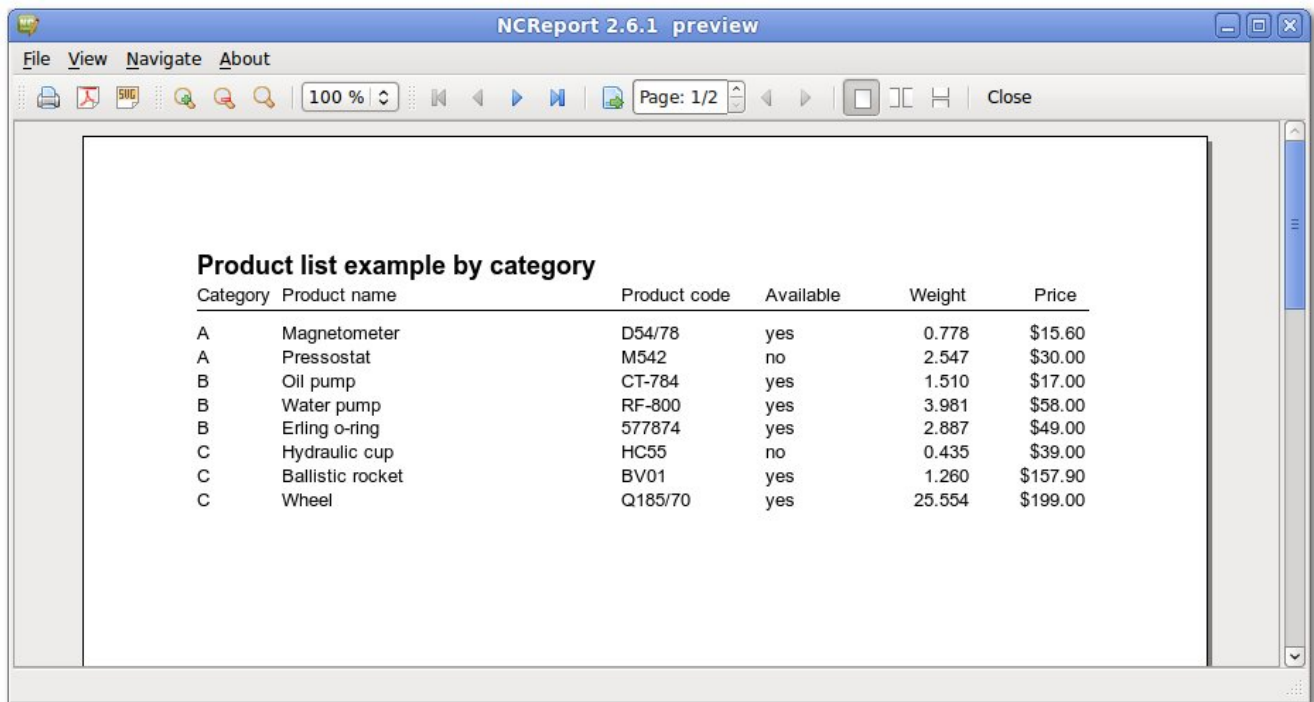
Page header					
1 <b>Product list example by category</b>					
Category	Product name	Product code	Available	Weight	Price
Detail [Detail1]					
col0	col1	col2	col3	col4	col5
Page footer					
appinfo					pageno

### 3.3. Testing report in Designer

Our sample report now is ready for testing. To run report from designer there are at least two ways: Select **Report > Report/Run report...** menu and after the report runner dialog appears you can choose the report's output. To start running report just click [ **OK** ] button.



For fast preview just select **Report > Report/Run report to preview...** menu and then the Designer will run report to print preview immediately. In this state the preview of our example report appears like this:



## 3.4. Variables and Groups

The following section describes how to use some advanced feature of NCReport. We will define a group and after we will add summary variables to our example report.

### 3.4.1. Adding a variable for summary

Variables are special numeric items used for providing counts and totals. Each of them have name, function type, data type, and have an assigned data source column the variable based on. To add a variable open the **Report > Report** menu and select **Report > Variables...** menu item. Then appears a dialog on you can manage variables.

The following options are available for variables:

#### Variable ID

The name/ID of the variable

#### Variable expression

The data source column name the variable is based on

#### Function type

The function type of the variable. Supported function types: **Sum**, **Count**

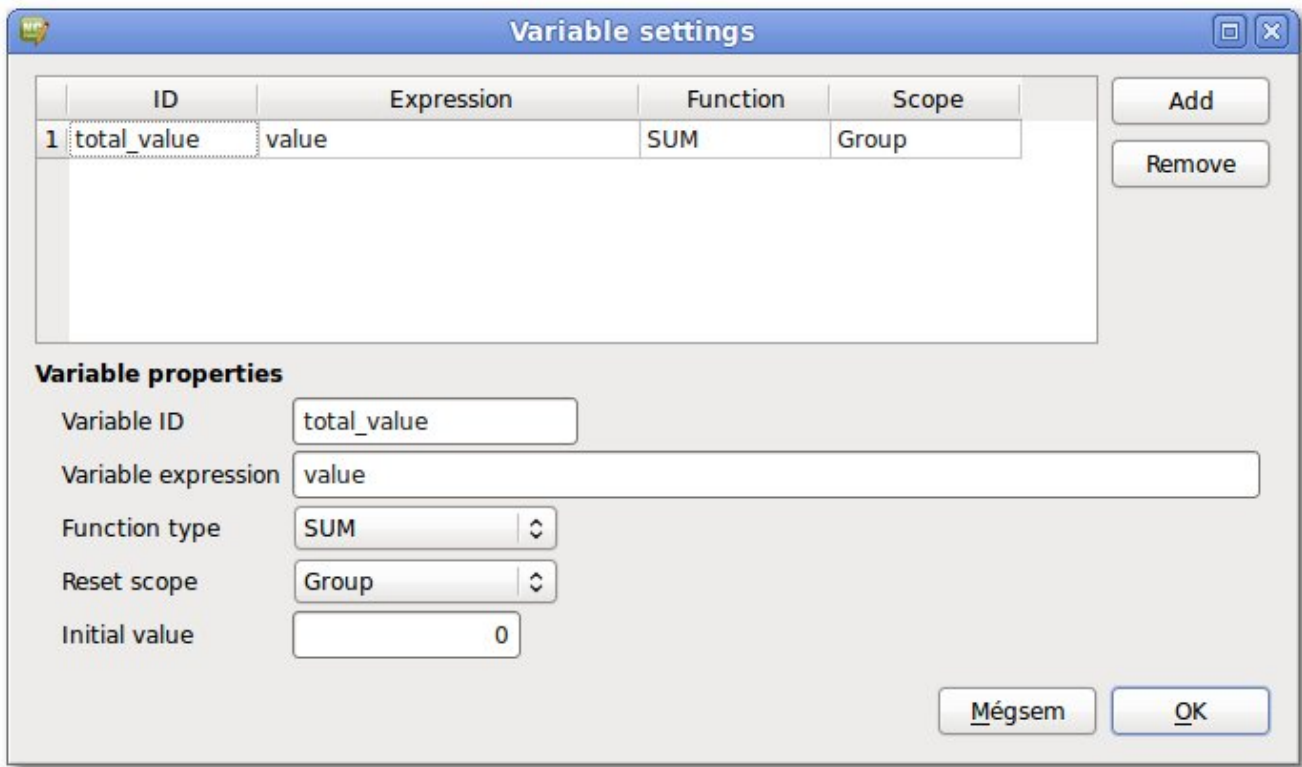
#### Reset scope

Specifies the scope after report engine resets the variable. Group level resets also must be set by group settings dialog.

## Initial value

Initial value of the variable

Let's create a **var0** which will summarize col4 column. (weight) It provides variable to summarize col4 values in 'Group' **Reset scope**. Specify the field's properties by this example:



The image shows a 'Variable settings' dialog box. It contains a table with columns: ID, Expression, Function, and Scope. The first row has ID '1', Expression 'total\_value', Function 'value', and Scope 'SUM'. Below the table are 'Add' and 'Remove' buttons. Under the 'Variable properties' section, there are input fields for 'Variable ID' (total\_value), 'Variable expression' (value), 'Function type' (SUM), 'Reset scope' (Group), and 'Initial value' (0). At the bottom right are 'Mégsem' and 'OK' buttons.

ID	Expression	Function	Scope
1	total_value	value	SUM

**Variable properties**

Variable ID: total\_value

Variable expression: value

Function type: SUM

Reset scope: Group

Initial value: 0

To apply settings click [ OK ] button on Variable dialog.

### 3.4.2. Defining a group

Reports often require summary data by band. In our example we will add weight summary by product category to report. First, open the **Report > Report** menu and select **Report > Details and grouping...** menu item, then appears a dialog on you may manage the detail sections and groups of the detail. Select "Detail1" detail and click the [ **Data grouping...** ] button, then the Group settings dialog will appear. The following properties are available for a group:

#### Group ID

The name/ID of the group for identification purposes

#### Group expression

The name of the data source column the group is based on. Group expression can be data source column or any data expression or constant. Let's set a constant group expression: **%CONST** Constant groups have only one group header and footer.

#### Header and Footer

To enable or disable group header and footer, check on or off the specified check box. To set initial

height of these sections you can use spin boxes near the check boxes.

### Reset variables

This list contains the 'Group' scope variables. You can specify which variable the report generator has to reset when a group level run out.

We want the grouping to be based on **col0** column (product category column). Specify the field's properties by this example:

The screenshot shows a dialog box titled "Data grouping [Detail1]". On the left, under the "Groups" heading, there is a list box containing "Group0" and "Group1". To the right of this list are two buttons: a green plus sign and a red minus sign. Below these are two more buttons: a green up arrow and a green down arrow. The "Group properties" section contains a "Group ID" text box with "Group0" and a "Group expression" text box with "%CONST". The "Header and Footer" section has four items: "Group header" (checked) with a height of "1,87 mm", "Group footer" (checked) with a height of "9,35 mm", "Reprint header on new page" (unchecked), and "Starts on new page" (unchecked). The "Reset variables" section has a list box containing "total\_value" which is checked. At the bottom right are two buttons: "Mégsem" and "OK".

To apply settings click **[ OK ]** button on Group dialog and then click **[ OK ]** button on Detail dialog. After doing so group header and footer of the detail will appear.



Page header						
Product list example by category						
1	Category	Product name	Product code	Available	Weight	Price
Group header [Detail1.Group0]						
1						
Detail [Detail1]						
	col0	col1	col2	col3	col4	col5
Group footer [Detail1.Group0]						
1						
Page footer						
	appinfo				pageno	

### 3.4.3. Adding summary field to group footer

To add a Field based on **var0** variable just add again a new field to group footer and specify the field's properties by example:

Field properties

Field source type: Variable

Data type: Numeric

Field

total value

General

Numeric

Date/Time

Other

Description / title:

Template arg() string:

Word wrapping

Automatic height

Hide repeated values

Background mode: Transparent

Background color: User color 1

Design time background

Rotation angle:

0°

Print only when expression is true:

Zone ID: 0

Mégsem

OK



After adding variable field and some labels and a line to group header and footer our report should look like this:

**Page header**

1 **Product list example by category**

Category	Product name	Product code	Available	Weight	Price
----------	--------------	--------------	-----------	--------	-------

**Group header [Detail1.Group0]**

Product category: col0

**Detail [Detail1]**

col0	col1	col2	col3	col4	col5
------	------	------	------	------	------

**Group footer [Detail1.Group0]**

Weight total: var

**Page footer**

appinfo pageno

### 3.4.4. Final testing the report

Now we are ready! For preview testing just select again **Report > Report/Run report to preview...** menu and then the Designer will run our report to print preview. In this state the preview of our example report appears like this:

**Product list example by category**

Category	Product name	Product code	Available	Weight	Price
Product category: A					
A	Magnetometer	D54/78	yes	0.778	\$15.60
A	Pressostat	M542	no	2.547	\$30.00
Weight total:				3.325	
Product category: B					
B	Oil pump	CT-784	yes	1.510	\$17.00
B	Water pump	RF-800	yes	3.981	\$58.00
B	Erling o-ring	577874	yes	2.887	\$49.00
Weight total:				8.378	
Product category: C					
C	Hydraulic cup	HC55	no	0.435	\$39.00
C	Ballistic rocket	BV01	yes	1.260	\$157.90
C	Wheel	Q185/70	yes	25.554	\$199.00
Weight total:				27.249	

Congratulations! We have created a simple one level group report. In the next step we will describe how to run this report from your application.

## 3.5. Integrating NCReport into a Qt application

### 3.5.1. Adding NCReport library to an application

In order to using NCReport from your application, first you have to integrate NCReport into your application project using the NCReport API classes. There are at least two ways to do this:

#### Statically adding source codes to your project

- Adding the whole sources statically to your project and build it together with your application. In this case you don't need NCReport shared libraries. Doing so open your **.pro** project file and add the full source package to the project as **testapp/testapp.pro** does.

#### Linking report engine as shared library

- Using NCReport engine as shared library. For using NCReport library like other libraries in your project you need to specify them in your project file. The following project example shows the necessary settings:

```
QT = xml sql gui core
TEMPLATE = app
CONFIG += warn_on \
qt \
thread \
release

TARGET = MyApplication
INCLUDEPATH = ../ncreport/includes

HEADERS += ...
SOURCES += ...

win32 {
LIBS += ../lib/ncreport2.lib
}
unix {
LIBS += -lncreport -L../lib -L/usr/local/bin
target.path = /usr/local/bin
}
```

For more information see the Qt documentation in qmake manual at chapter Declaring Other Libraries.

### 3.5.2. Initializing NCReport class

This step shows you how to initialize NCReport class. Includes. First we have to add includes. To include the definitions of the module's classes, use the following includes:

```
#include "ncreport.h"
#include "ncreportoutput.h"
#include "ncreportpreviewoutput.h"
#include "ncreportpreviewwindow.h"
```

Creating NCReport class. We create the report class just like as another QObject based class:

```
NCReport *report = new NCReport();
```

If NCReport object has been created earlier and passed as a parameter, you should initialize the report by calling **reset()** method:

```
report->reset();
//or
report->reset(true);
```

**NCReport::reset()** function will delete all object references, and makes report engine able to run a report again. If parameter is set TRUE, also report parameters, added data sources such as QStringList, custom items will be deleted.

### 3.5.3. Setting the Report's source

Report source means the way of NCReport handles XML report definitions. Report definitions may opened from a file - in most cases it is suitable, but also it can be loaded from an SQL database's table. In our example we apply File as report source:

```
report->setReportFile( fileName );
```

This code is equivalent with this code:

```
report->setReportSource( NCReportSource::File );
report->reportSource()->setFileName( fileName );
```

Adding parameters To add a parameter to NCReport use addParameter method. The parameter ID is a string, the value is a QVariant object.

```
report->addParameter( "id", QVariant("value") );
```

This code is equivalent with this code:

```
report->addParameter( "paramID", "Parameter value" );
```

### 3.5.4. Running the Report

Now we are ready to run the Report to different outputs. Doing so just use one of `runReportTo...` functions. Running report to printer

```
report->runReportToPrinter();
```

Running report to PDF

```
QString fileName("mypdffile.pdf");
report->runReportToPDF( fileName );
```

Running report to Print Preview

```
report->runReportToPreview();
```

If you run report to preview, result will be stored in an `NCReportPreviewOutput` object. Report engine does not run the preview form automatically. After the report engine successfully done we need to initialize an `NCReportPreviewWindow` object for previewing. Before doing so we check if a report error occurred.

```
if ( !report->hasError() ) {
    NCReportPreviewWindow *pv = new NCReportPreviewWindow();
    pv->setOutput( (NCReportPreviewOutput*)report->output() );
    pv->setWindowModality( Qt::ApplicationModal );
    pv->setAttribute( Qt::WA_DeleteOnClose );
    pv->show();
} else {
    QMessageBox::warning( tr("Error");
}
```

To get the current output use `NCReport::output()` function.



When you run report to preview the report output object won't be deleted by `NCReport`. When the `NCReportPreviewWindow` object is destroyed, output is deleted automatically by its destructor.

### 3.5.5. Error handling

To catch errors you can use the following functions:

```
bool error = report->hasError();
QString errmsg = report->lastErrorMsg();
```

Deleting Report object After report running action you may delete the report object. When NCReport object is deleted all child objects are also deleted.

```
delete report;
```



Don't delete NCReport object if **NCReportPreviewWindow** object still exists. If you want to use report object again without deleting just use **NCReport::reset()** function.



## 4.4. Line

Graphical line that has a start point and an end point. The line characteristics are configurable, you can change the line type and width and the color. Lines are static objects.

## 4.5. Rectangle

Graphical rectangle that has a 4 corner points, specified by height and width. The rectangle characteristics are configurable, you can change the line type and width and the color. Rectangles can be transparent or filled. Rectangles are basically static objects, but with the **Dynamic Position** feature the height and the width can be managed dynamically using a data expression

## 4.6. Ellipse

Graphical ellipse that is specified by height and width. The ellipse characteristics are configurable, you can change the line type and width and the color. Ellipses can be transparent or filled. Ellipses are basically static objects, but with the **Dynamic Position** feature the height and the width can be managed dynamically using a data expression.

## 4.7. Image

Image file or embedded image content with PNG, JPEG, BMP, SVG supported formats. You can assign a direct link to a file or use a static, in-report stored image. The image is specified by height and width, many characteristics are configurable: Alignment, Aspect ratio, Transformation mode. There can be dynamic and static images in reports so the image element is very flexible.

## 4.8. Barcode

NCReport provides the barcode rendering with almost all symbology features thanks to the Zint back-end 3rd party library. Barcode item can render a barcode from static or dynamic source.

## 4.9. Table View / Model

Table View item is a report item destined to rendering QTableView tables with WYSIWYG print support. The function is aimed to print the tables in the same rate as the existed QTableView screen widget. The table view item should follow the formats of the original QTableView widget. The cells gets display outlook information from the table's item model. Some basic table settings such as header background, line type, line color, etc. are currently fixed. The table item also supports rendering a QAbstractItemModel without a QTableView

## 4.10. Cross Table

This is known as PivotTable or a cross-tab report, or rotated data within a table. In cross-tab tables, the data source records are represented as horizontal columns, and the cross-tab rows are printed as data source columns. Tables often include horizontal and/or vertical summarization as well.

## 4.11. Custom Graphics Content

Graph/Custom item is a special member of NCReport items. This option enables you to render a QPainter drawn contents in reports using C++/Qt code. The typical field of application is using this feature for rendering graphs or similar user defined content



# Chapter 5. Parameters

Parameters are data obtained from outside of the report generator. The application that calls the `NCReport` object passes information as parameters to the `NCReport` class using the `NCReport::addParameter(...)` method. Parameters get evaluated within SQL queries and script expressions. Field objects can also have a parameter data source type, so they can be presented as data in the report. Parameters are mostly used in SQL queries and data/script expressions.

## 5.1. Parameter Syntax

To embed a parameter into a query or an expression, use the following syntax:

```
$P{parameterID}
```

Example of using a parameter in an SQL query:

```
SELECT productId, productName FROM db.products WHERE primaryKey=$P{parameterID}
```

Example of using a parameter in a script expression:

```
"$P{firstname}" + "$P{lastname}"
```

Example of using a parameter in a data template expression:

```
Dear $P{firstname} $P{lastname},
```

## 5.2. Testing Parameters

To modify the SQL query, open the Report menu and select the Report/Data sources... menu item. Modify the SQL query of our connection as follows:

```
SELECT ProductID, ProductName,  
QuantityPerUnit, UnitPrice, QuantityPerUnit*UnitPrice as value  
FROM products  
WHERE ProductID > $P{prodID}  
ORDER BY ProductName
```

Add a Parameter with the ID/name `prodID` to `NCReport`. `NCReport Designer` has a test runner dialog with a parameter adding feature. To open the runner dialog, select the Run report... menu item from the Report menu. Click the Add button and specify the name as `prodID` and the value to 70. After running the report to Preview, we get the following result:

Page: 2/2

42	Product: Singaporean Hokkien Fried Mee	14.00	12.00	168,00
20		81.00	60.00	4 860,00
21		10.00	9.00	90,00
61	Product: Sirop d'arable	28.50	24.00	684,00
46	Product: Spegesild	12.00	10.00	120,00
35		18.00	16.00	288,00
62	Product: Tarte au sucre	49.30	41.00	2 021,30
19		9.20	8.00	73,60
29		123.79	80.00	9 903,20
14		23.00	21.00	483,00
54	Product: Tourtiare	7.45	6.00	44,70
23		9.00	8.00	72,00
7		30.00	25.00	750,00
50	Product: Valkoinen suklaa	16.25	14.00	227,50
63	Product: Vegie-spread	43.90	37.00	1 624,30
64	Product: Wimmers gute Semmelknadel	33.25	28.00	931,00
47	Product: Zaanse koeken	9.50	8.00	76,00
Total value:				98 013,66

Page: 2/2

In all reports, Parameters are always evaluated within SQL queries, scripts, and PrintWhen expressions.

## 5.3. API code: Passing parameters to the report

```
NCReport* report = new NCReport(parent);
QString companyName = tr("ABC Ltd");
report->addParameter("myid", companyName);
```

# Designer

The NCReport Designer is a standalone GUI application to create and modify report templates. This part makes possible to learn how to use the designer, how to design a report.

# Chapter 6. Getting Started with NCReport Designer

This chapter covers the fundamental steps that most users will take when creating reports with NCReport Designer. We will introduce the main features of the tool by creating a simple report that we can use with NCReport engine.

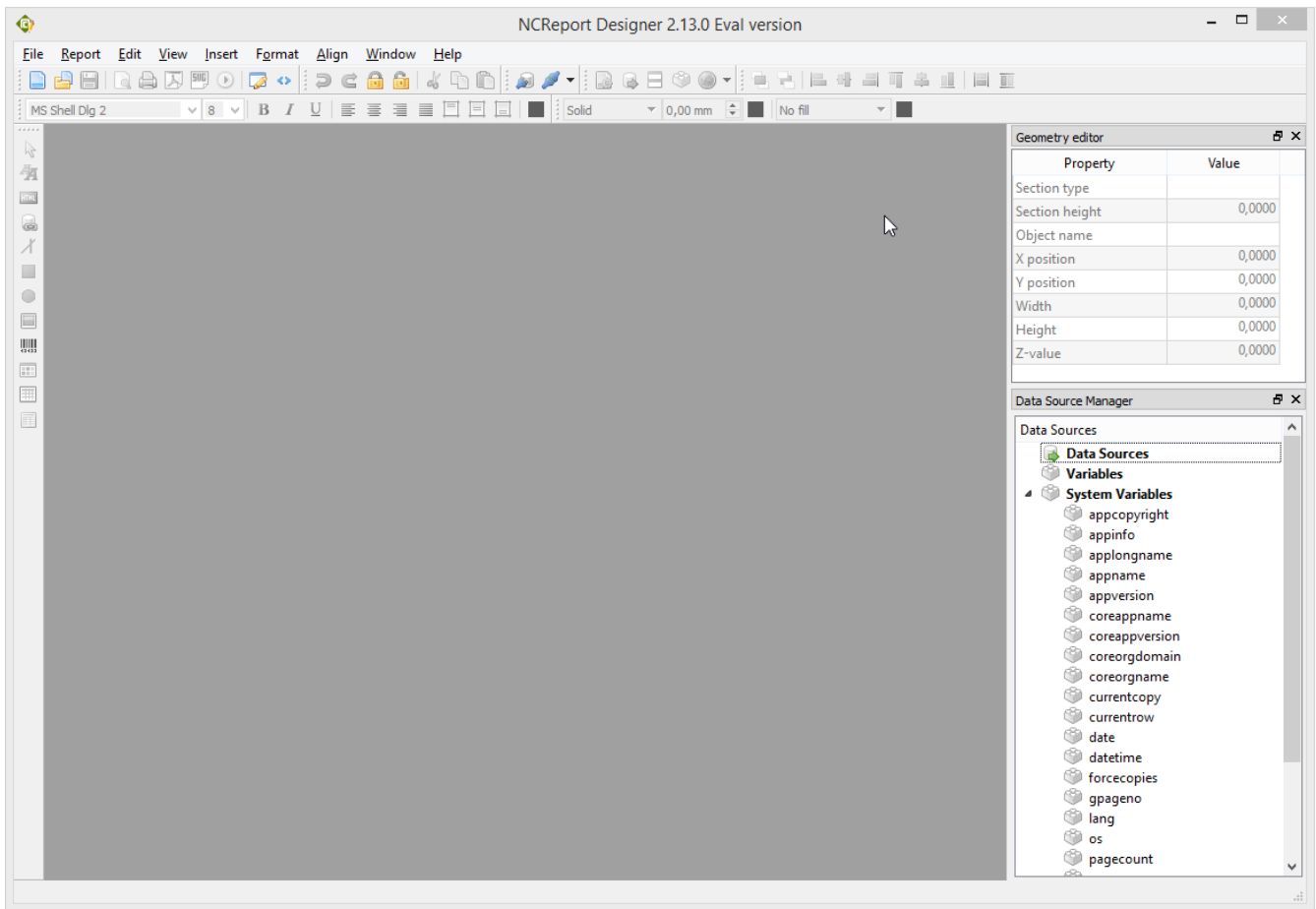
## 6.1. Launching Designer

The way that you launch NCReport Designer depends on your platform:

- On Windows, click the Start button, open the Programs submenu, open the NCReport2 submenu, and click NCReport Designer.
- On Unix or Linux, you may find a NCReport Designer icon on the desktop background or in the desktop start menu under the NCReport submenu. You can launch Designer from this icon. Alternatively, you can enter `./NCReportDesigner` in a terminal window in NCReport/bin directory
- On MacOSX, double click on NCReport Designer in the Finder.

## 6.2. The User Interface

NCReport Designer's user interface is built as any standard multi-window user interface. The main window consists of a menu bar, a tool bar, and a geometry editor for editing the position and size of objects. Geometry editor can be enabled or disabled by clicking on View/Geometry editor checkbox menu.



## 6.3. NCReport Designer Main Window

The menu bar provides all the standard actions for opening and saving report files, managing report sections, using the clipboard, and so on. The tool bar displays common actions that are used when editing a report. These are also available via the main menu. File menu provides the file operation actions, Report menu contains the report and its sections settings that belong to the current/active report. View menu displays the specified items can be enabled or disabled in MDI area. The Tool menu provides common report objects that are used to build a report. The Align menu holds the alignment actions for the specified report items can be aligned. With the Window menu you can manage the windows are opened concurrently.

Most features of NCReport Designer are accessible via the menu bar or the tool bar. Some features are also available through context menus that can be opened over the report sections. On most platforms, the right mouse button is used to open context menus.

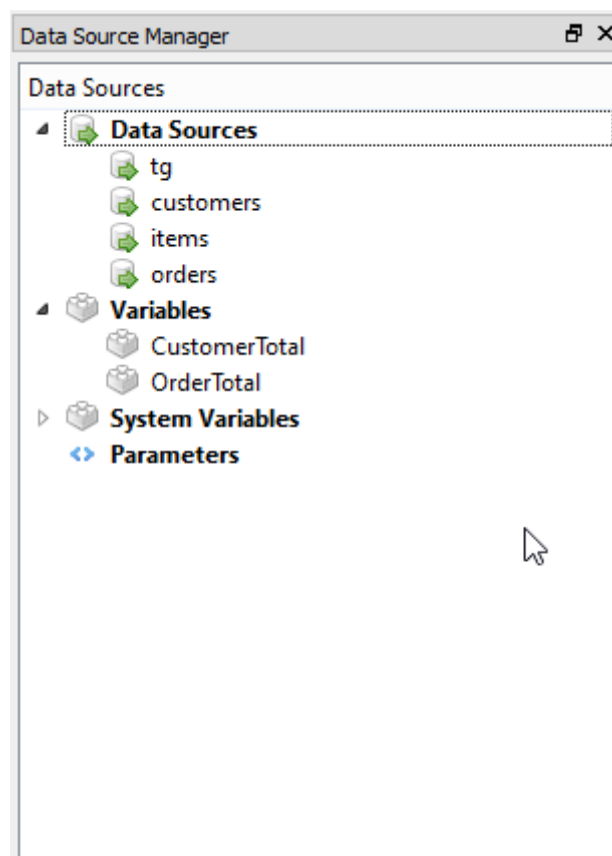
## 6.4. Geometry editor

Geometry editor is a tool window can be enabled by View/Geometry menu. This window displays the position and size informations of the current report section or object. The current objects or sections are always activated by a mouse click. You can type the numeric size or position values into the spin boxes. Any changes made to the object's properties cause it to be updated immediately.

Geometry editor	
Property	Value
Section type	Page header
Section height	25,389
Object name	label
X position	0,0000
Y position	0,0000
Width	89,9583
Height	9,5250
Z-value	1,0000

## 6.5. Data Source Tree

Data Source Tree (or data source manager) is a dock window widget in the main designer desktop. The widget helps to add fields to the report very easily by a simple drag and drop action. The data source tree is updated when you add or modify a data source in the report. Therefore it is recommended to start the report building with defining the data source first. If the data columns are available at design time they will appear in the widget under the appropriate data source item.



### 6.5.1. Adding a Field using the Data Source Tree

To add a field to any report section just drag the selected column and drop onto the section at the position whatever you want. The Field item will be created at the drop position. Note that mouse pointer target position is considered.

## 6.6. Field Expression Builder

When you work with Field items you get a useful helper tool for creating the correct Field expression. You find the Expression builder button in the Filed settings dialog labeled **Build Expression...** and besides the Print When logical expression editor controls. You can choose the combo boxes to select the desired expression and you can add it by simple clicking on the small add buttons. Then the expression will be inserted into the text area at the cursor position. Depending on what type of expression you insert the expression builder will apply the correct syntax. You have also the data source tree in the dialog that can be used for the same purpose if you just simply select a Data Source column, a Parameter or a Variable. Double clicking on the appropriate item will insert it from the data source tree.

### 6.6.1. Expression Builder Dialog Controls

The following description helps to understand what combo box widgets are found on the dialog and what they are good for.

#### Field source type

You can select the field source type. This is what to do first. The source type will determine what other controls will be available.

#### Data source

Selects the data source from all available data sources of the report. To add a data source click on the add item beside the widget.

#### Column

Selects the data source column. To add a data column click on the add item beside the widget.

#### Variable

You can select here all available variables including the system variables. To add a variable click on the add item beside the widget.

#### Data source function

You can select the available data source level functions here. To add a function expression click on the add item beside the widget.

#### Value function

You can select the available data source value functions here. To add a function expression click on the add item beside the widget.

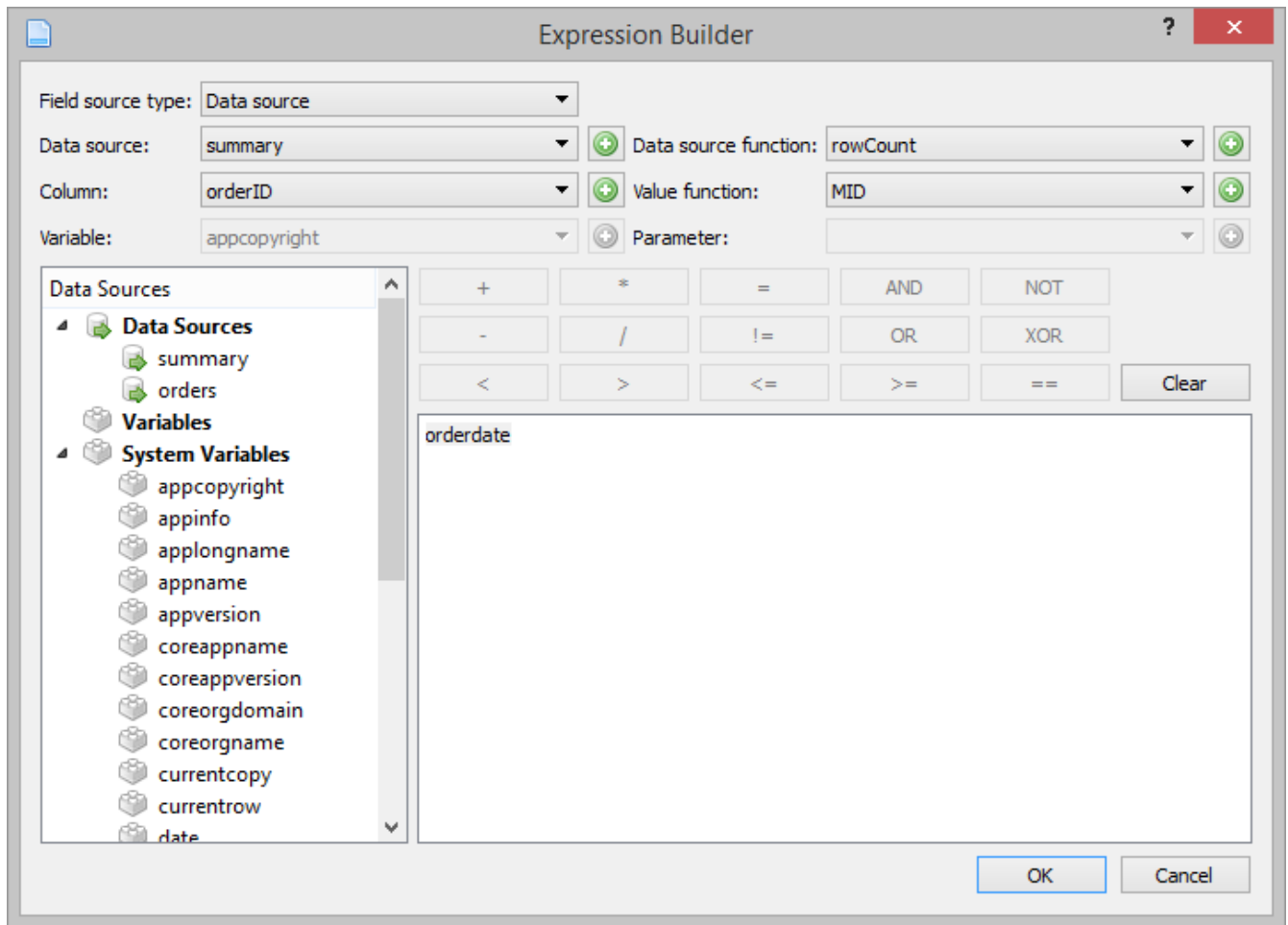
#### Parameter

You can select the design time defined parameters here. To add a parameter click on the add item beside the widget.

### 6.6.2. Logical Operation Buttons

You find also logical operation buttons on the dialog that boosts editing of a script or a logical expression. When you click on a button it will insert the named logical operation into the text area.

### 6.6.3. Expression Builder Dialog



## 6.7. Designing a report

In this chapter we will look at the main steps that users will take when creating new report with NCReport Designer. Usually, creating a new report will involve various activities:

- Deciding what kind of report structure to create.
- Deciding which kind of data sources to use.
- Defining the data sources
- Adding the report sections are needed
- Deciding which kind of items/objects to use in the different sections.
- Composing the user interface by adding report objects to the report sections.
- Connecting to SQL data source if needed\* Testing the report

Users may find that they prefer to perform these activities in a different order, However, we present each of the activities in the above order, and leave it up to the user to find the approach that suits them best. To demonstrate the processes used to create a new report, we will take a look at the steps needed to create a simple report with NCReport Designer. We use a report that engages SQL database data source to illustrate certain features of the tool.



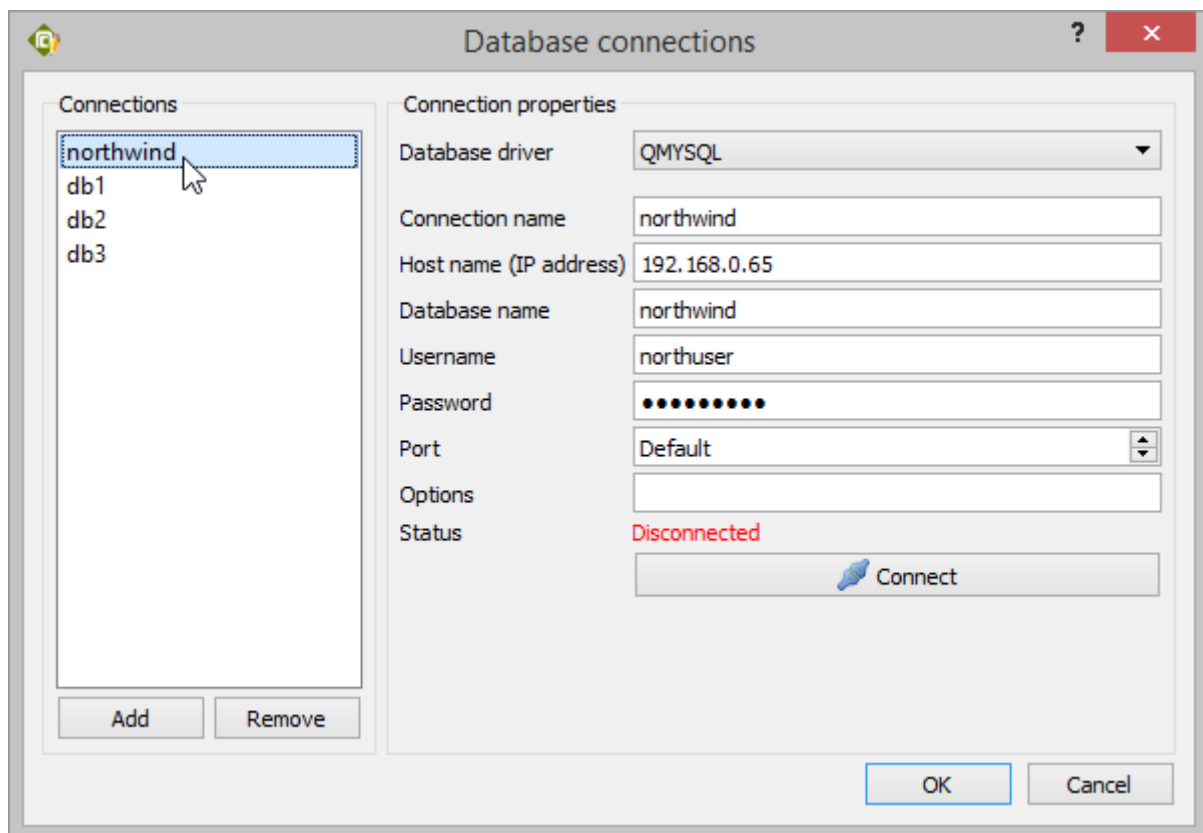
## 6.8. Connecting to database from Designer

NCReport Designer now enables you to test the report from inside the designer. Since this report requires internal MySQL database connection, first we should connect to "northwind" database. SQL database connections can be managed by the Connection manager within the designer application. Open the **Report** menu and then select **SQL connection manager...** After the Connection manager dialog will appear. By this dialog you can add one or more SQL connections.

The following options are available for connections:

- **Database driver** The appropriate SQL database driver.
- **Connection name** The name of the database connection. Qt uses this name in `addDatabase(...)` function. For identifying the corresponding connection this value have to be specified.
- **Host name** Name or IP address of host
- **Database name** Name of the database
- **Username** Connection's user name
- **Password** Connection's password
- **Port** Connection's port number. If empty, the default port is used in connection.

### 6.8.1. SQL connection dialog



- **Connect** Tries to establish the connection
- **Add** Adds a new connection and enables the options to edit
- **Remove** Removes the connection selected from the list

- **OK** Select to save your connection settings.
- **Cancel** Closes the dialog without saving any changes, returning you to the desktop.

After you specified connection parameters to the added connection use the Connect button to establish connection. If the connection is succeeded then your report is ready to run. Before running the report we rename our connection to northwind and then also our data source's connection name must be renamed to northwind. Doing so just open again the **data sources...** dialog and then rename the connection ID to northwind too



You don't need any SQL connection if you use non SQL data source in your report, for example Text, Stringlist or other data source

### 6.8.2. Connection names (ID)

Each SQL data source has its own connection name that is being used for running SQL queries. The QSqlDatabase connection should have the same name you set in the data source. In the designer the connection manager create a new connection with the same connection name you defined in the report.

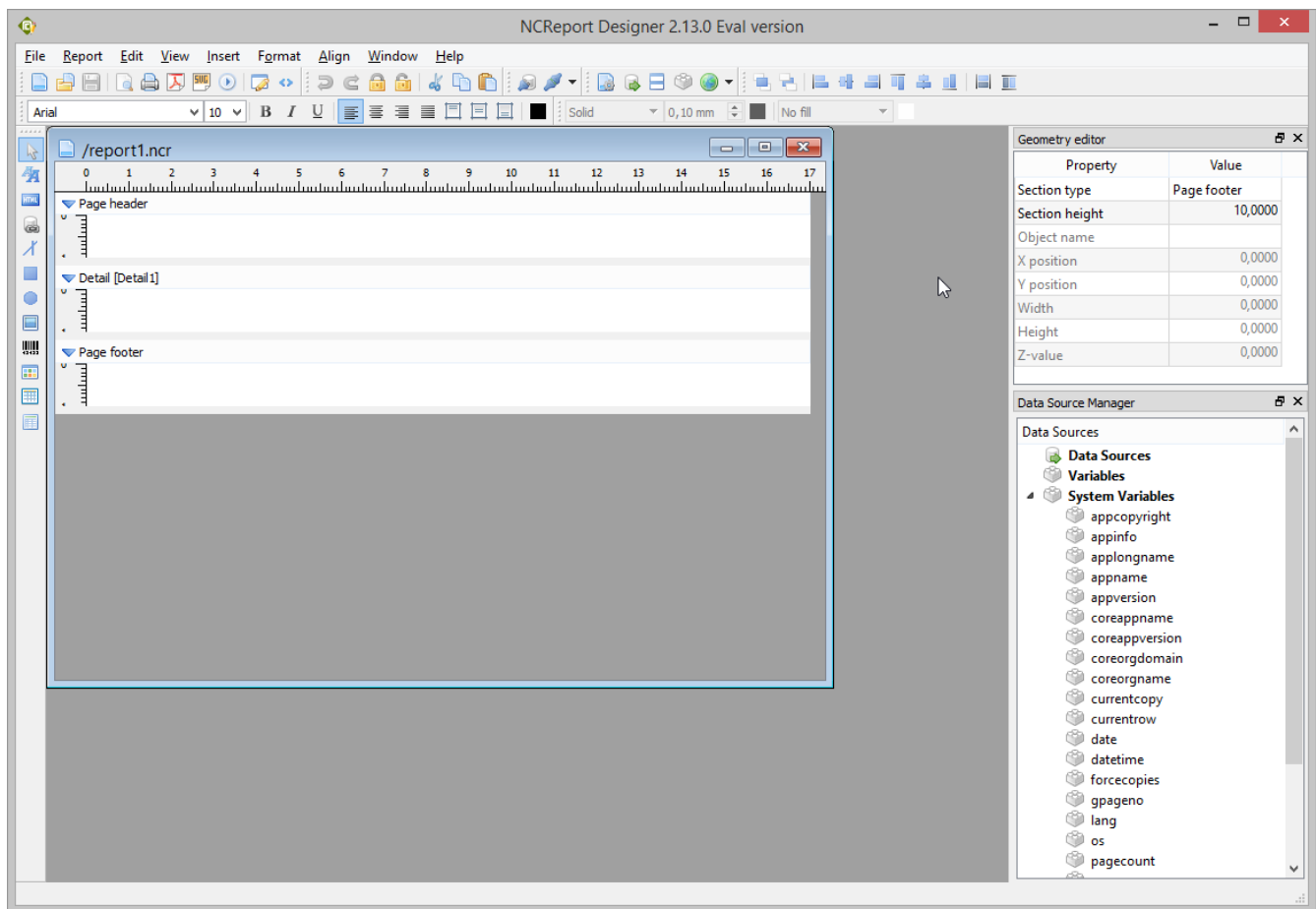
### 6.8.3. Using default connection

When the SQL data source does not have a connection name the report engine will run the SQL query under the scope of the default database connection. Create a DB connection in the designer connection manager with "default" connection name. That connects as a default connection so you'll be able to run your report from the designer with not specified (empty) connection name in the report.

## 6.9. Beginning a new report

By clicking the New menu or tool opens a new instance of a report. Select this tool button or menu to begin a new report definition. By default the new empty report contains page header, a detail and a page footer sections.

### 6.9.1. New report



## 6.10. Report sections

Report sections are the representations of the function specific areas inside the report. Reports are builded from sections. They are often a recurring areas such as detail, header or footer. The most important section is called Detail since details can contain the fields are changed row by row. Each sections can contain all kinds of report items. Item's coordinates are always relative to their parent section. One report can contain the following sections: Report header, report footer, page headers, page footers, group headers and footers and details

To change the height of a section just drag the bottom resizer bar under the section area and resize to the size you want or type the height value in millimeter at Geometry editor's spinbox if that is enabled. To activate the current section just click onto the empty area of a section

### 6.10.1. Detail

The core information in a report is displayed in its Detail section. This section is the most important section of the report since it contains the row by row data from the data source. Detail section have the following characteristics:

- Generally print in the middle of a page (between headers and footers)
- Always contain the core information for a report
- Display multiple rows of data returned by a data source
- The detail sections generally contains fields.
- Multiple independent details are allowed in one report, each detail after the other

- All of details are assigned to one specified data source

### 6.10.2. Page header

Page headers is used to contain page headings. Page headers have the following characteristics:

- Always print at the top of a page
- Always contain the first information printed on a page
- Only display one (current) row of data returned by a data source
- Only one allowed per page

In most cases you need page header in reports. To add or remove page header select Report/Page options... menu, then appears a dialog on you can set the page options of the current report. To enable or disable page header just use Page header check box.

### 6.10.3. Page footer

Page Footer are commonly used to close the pages. Page footers have the following characteristics:

- Always print at the bottom of a page
- Only display one (current) row of data returned by a data source
- Only one allowed per page

Page footer is usually used to display informations such as number of the page, report titles and so on. In most cases you need page footer in reports. To enable or disable page footer just use Page footer check box in Report/Page options... menu.

### 6.10.4. Report header

Report header is a section used to contain report headings. Report header has the following characteristics: \* Always printed after the page header \* Report header is printed only once at the beginning of the report \* Displays only one (current) row of data returned by a data source

To enable or disable report header use Report header check box in Page options dialog can be activated by opening **Report** menu and selecting **Page Options...**

### 6.10.5. Report footer

Report footer is a section commonly used to close the report. Report footer has the following characteristics:

- Always printed before the page footer at the end of the report
- Only display one (current) row of data returned by a data source
- Only one allowed per report

To enable or disable report footer use Report footer check box in Page options dialog can be activated by opening **Report** menu and selecting **Page Options...**

## 6.11. Setting up page and report options

Page options of the current report can be specified in Page options dialog. Open the **Report** menu and select **Page options....** In the report page settings dialog you can specify the following options:

- **Report name** Type the name of the report. It's just an informative option, it's not used by report generator.
- **File encoding** The encoding of the XML file. When user opens or saves the report definition file, this will be the default encoding. In most cases UTF-8 encoding suit the requirements, but for special international characters you may choose the specified encoding.
- **Page size** The size of the page. The size names are listed in the combobox and their names are the standard size names. Currently the standard page sizes are supported.
- **Default font** The font name and size are basically used for the text labels and fields in the whole report. Each object may change this option.
- **Background color** The background color of the report. This option currently is not used.
- **Header and footer settings** The check boxes can be used to enable or disable page header/footer and report header/footer. To alter the height of these sections you may use spin boxes corresponding to their check boxes. You can also change these height properties by mouse dragging or by geometry editor
- **Margins** margin properties represent the top, bottom, left and right margins of the page in millimeters. To alter the margin values just use the spin boxes.
- **Orientation** This radio button option represents the orientation of the page, Portrait or Landscape orientation can be selected

The screenshot shows the 'Report and page settings' dialog box with the 'Page settings' tab selected. The 'General options' section includes a text field for 'Report name' containing 'My test report', a dropdown for 'Report type' set to 'Report', a dropdown for 'File encoding' set to 'UTF-8', and a dropdown for 'Background color' set to 'white'. Below these are two unchecked checkboxes: 'Double pass mode' and 'Subreports start on new page'. The 'Default Font' section includes a dropdown for 'Font family' set to 'Arial', a dropdown for 'Point size' set to '10', and four unchecked checkboxes for 'Style': 'Bold', 'Italic', 'Underline', and 'Strikeout'. The dialog has 'OK' and 'Cancel' buttons at the bottom right.

The following buttons are available for apply or cancel settings:

- **OK** Select to apply your settings.
- **Cancel** Closes the screen without saving any changes, returning you to the designer desktop.

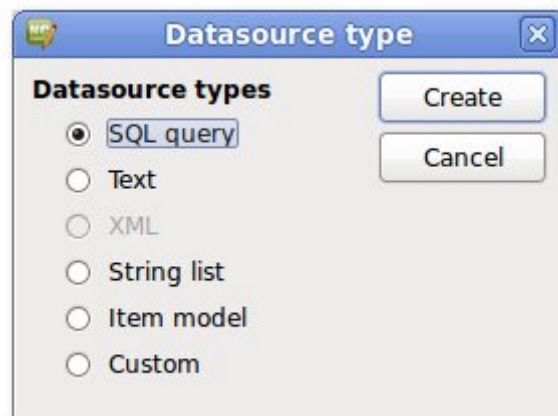
Specify the report page properties by using Page options dialog and validate the settings by clicking the *OK* button.

## 6.12. Adding data sources

At the very beginning we have to decide what data source type(s) we intend to use in the report. Since the report generator builds a printable representation of data from a data source, at least one data source must be defined in the report. Data may be fetched from an SQL query using Qt's database SQL database connection drivers or from other sources that don't require SQL connection, such as text, string list or custom defined data source. One report can contain multiple data sources and each details can be connected to one selected data source. Often a data source is not assigned to any of detail, in this case you can use these kind of unassigned data sources as a one (first) row/record source of data. See the details later.

To specify a data source to your report open the **Report** menu and select **Data sources...** menu item. Then appears a dialog on you can add and remove data sources. To add a new data source click the **Add** button and then select the data source type from the list of available data source types.

### 6.12.1. data source types dialog



In our example we choose SQL query data source type. After you click **OK** button a new SQL query data source will be added to the list in dialog panel. Then you can specify the data source options. The following properties are available for data sources:

- **data source ID**. This string property is very important for identification purposes. You can refer to the data source by this ID string.
- **data source type** The type of the data source you've chosen before. It is cannot be changed after the data source added to the list
- **Location type** Location type is a property that describes where the data or the sql query can be found, inside the report file or inside an external file. It's value may be: Static, File, HTTP, FTP, Parameter. HTTP and FTP type currently is not supported for SQL queries. For the different type

of data sources it means a bit different. For SQL query the Static location type is suitable, it means that SQL query will be saved statically into the report file. Parameter type provides that the data is added to NCReport by NCReportParameter. For example a QString Text or an SQL query can be added as parameter to NCReport depending on the data source type.

- **File name/URL** In case non Static location type is selected, here you can specify the name of the file that contains data. (URL address currently is not supported.)
- **Connection ID** This string property represents the ID of an SQL database connection. This name just the same ID that is used in QSqlDatabase::addDatabase() function for identifying database connection. When you add database connection in your application before running report, this connection name you should specify.
- **Use external connection** If you want to make available the SQL data source to use it's own database connection, you may enable this checkbox. After, the external connection panel becomes enabled and you can specify the required properties of sql connection: hostname, database, username, password, port (optional).
- **SQL query** This text area in which you can edit the sql query expression. Almost every cases it is a **SELECT...FROM** expression applying the SQL syntax of the specified database. Only one sql query is allowed for the data source. SQL expression can contain Parameters, see later.

### 6.12.2. SQL data source

The screenshot shows the 'Data source settings' dialog. On the left, a list of data sources includes 'items' and 'orders', with 'orders' highlighted. The main area is divided into 'Data source properties' and 'SQL Query' sections. In 'Data source properties', 'Data source type' is set to 'SQL query', 'Data source ID' is 'orders', 'Opening/running role' is 'Beginning of the report', 'Location type' is 'Static', and 'Allow empty data source' is checked. The 'SQL Query' section has three tabs: 'SQL Query' (selected), 'Database connection', and 'Query test'. Under 'SQL Query', the 'Identification' sub-section shows 'Connection ID' as 'northwind' and 'Forward only mode' as unchecked. The 'Query' sub-section contains a text area with the following SQL query: `SELECT * FROM orders WHERE orderdate between '2005-01-01' AND '2005-02-28' ORDER BY orderdate`. A 'Test Query' button is at the bottom right of the query area. At the very bottom of the dialog are 'OK' and 'Cancel' buttons.

In our example we set the data source ID to data source1 (the default name) and choose Static location type. We name the Connection ID Con0. After the SQL query must be specified.



This example requires a running MySQL database server with existing northwind demo database and tables. For generating sample database and tables SQL script file is attached with NCReport project

Let's use this simple query:

```
SELECT ProductID, ProductName, QuantityPerUnit, UnitPrice,  
QuantityPerUnit*UnitPrice as value  
FROM products  
WHERE ProductID>20  
ORDER BY ProductName
```

The following buttons are available for apply or cancel settings: \* **OK** Select to apply your data source settings. \* **Cancel** Closes the screen without saving any changes, returning you to the designer desktop.

Validate the data source settings by clicking the **OK** button.

## 6.13. Assigning data source to the Detail

To assign the data source we defined before, open the **Report** menu and select **Details and grouping...** menu item, then appears a dialog on you may manage the detail sections of the report. A default Detail1 named detail is already defined. You can rename it to the name you want if you change Detail ID. Select the previously defined data source from data source combo box. The combo box contains all of defined data sources. This option must be specified for working of the report.

The screenshot shows the 'Detail settings' dialog box. On the left, under 'Details', 'Detail1' is listed. The 'General' tab is selected, showing 'Detail properties' with 'Detail ID' set to 'Detail1', 'Height' set to '4,81 mm', and 'Data source' set to 'orders'. Below this is a 'Page break' section with a 'Break condition (logical expression)' field and a 'Starts on new page' checkbox. At the bottom are buttons for 'Groups...', 'OK', 'Cancel', and 'Apply'.



Here we summarize the options of Detail dialog: - **Detail ID** The name of the detail section. - **Height** Height of the detail section in millimeters. To alter the height of these sections you may use this spin box. You can also change the height by mouse dragging or by geometry editor - **data source** data source name assigned to the detail. Previously defined data sources can be selected in the combo box - **Data grouping** By clicking this button the group management dialog of the corresponding detail can be opened.

You can add more details by **Add** button or remove existing detail by **Remove** button. One detail section must be existed, so it does not construe to remove the only one detail.

The following buttons are available for apply or cancel settings:

- **OK** Select to apply your settings.
- **Cancel** Closes the screen without saving any changes, returning you to the designer desktop.

Validate the detail settings by clicking the **OK** button.

## 6.14. Adding report items

After we defined the data source and specified the report options now we can design the report by adding items to the specified sections. The **Tools** menu or the tool bar displays report items that can be used when designing a report. Let's summarize the various report items of NCReport:

### Text label

The Label represents simple text or label items. Label items are used to display descriptive information on a report definition, such as titles, headings, etc. Labels are static item, its values don't change when rendering the report.

### Field

The Field is the matter of report items. It represents the data Field objects. By data type Fields may be text, numeric and date. Field items are used for pulling dynamically generated data into a report from the specified data source such as database the report generator uses. For example, a Field item may be used to present SQL data, variables and parameters. NCReport handles data formatting for the different type of fields like numbers or texts.

### Line

The Line option enables you to create Line items. In general, Line items are used for drawing vertical, horizontal lines for headings, underlining titles or so on. Lines are defined by its start and the end point coordinates

### Rectangle

The Rectangle enables you to create Rectangle items. Rectangles are usually used for drawing boxes or borders around a specified area. Rectangle makes easier the box drawings instead of drawing four lines.

### Ellipse

The Ellipse item enables you to create circle or ellipse in report. Ellipses are mostly used for drawing charts or borders around a text.

## Image

The Image option enables you to create Image items. Image items are used to insert either static or dynamic into a report definition. Static images such as a company logo often displayed in the Report Header can be loaded from a static file or from report definition. Dynamic images can be loaded from the specified SQL data source.

## Barcode

The Barcode option enables you to create barcodes. Currently the EAN13 code format is supported. Barcodes might be either static or dynamic items similar to images. Static barcodes read it's value from the report definition, dynamic barcodes are loaded from the specified data source.

## Custom item / Graph

Graph/Custom item is a special member of NCReport items. This option enables you to render a QPainter drawn contents in reports using C++/Qt code. The typical field of application is using this feature for rendering graphs or similar contents.

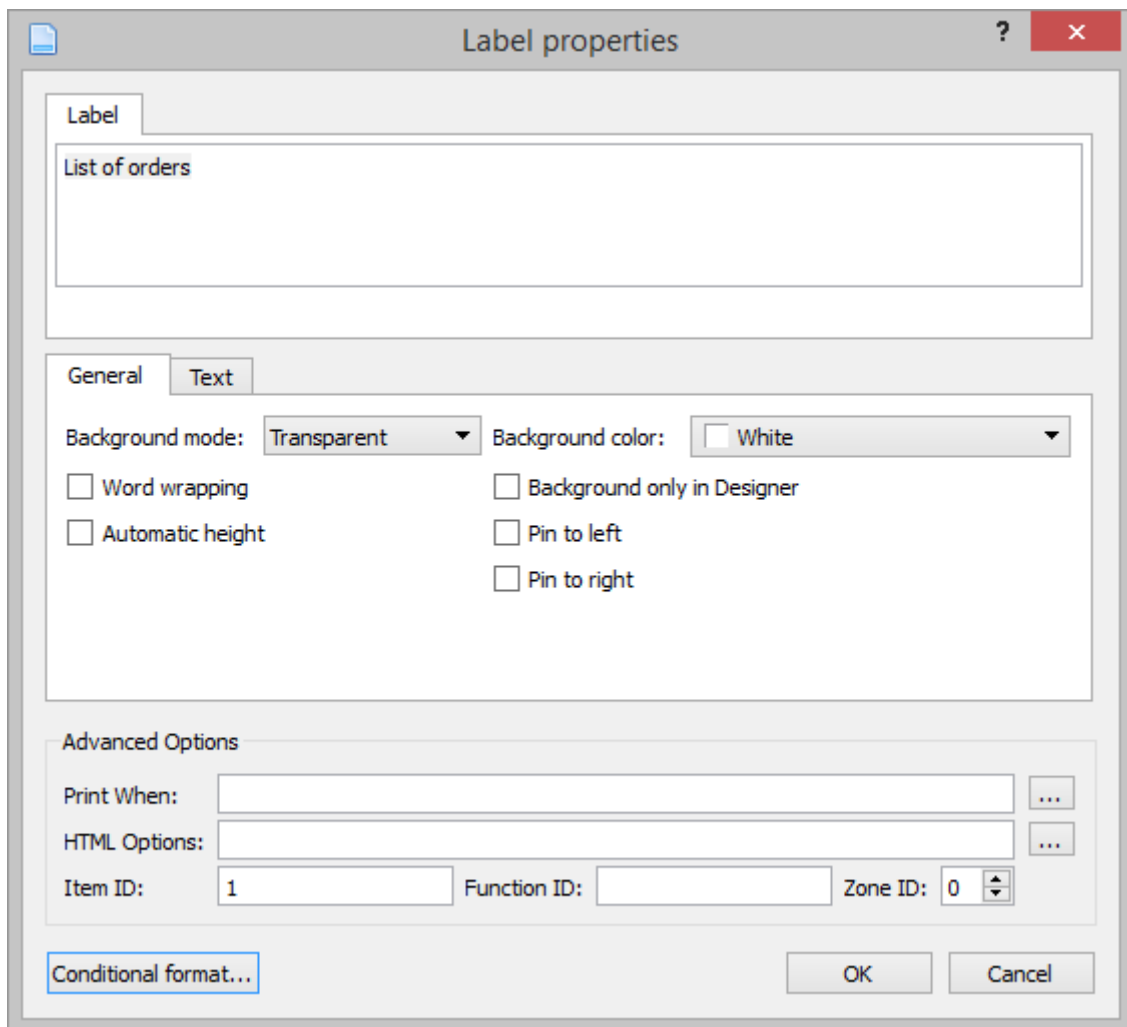
### 6.14.1. Adding heading Labels

First let's add the labels that represent the column header of data rows. To create a new Label object, first select the **Label** tool button or menu item in **Tools** menu. After that the cursor changes to a cross beam, then click in the section of the report definition where you want the Label to be located. (i.e. we add label to the report header.) Doing so will create the Label object in that section and opens the Object settings dialog. On the dialog you may then set the Label object's properties.

The following options are available for labels:

- **Text** Just enter here the text of the label
- **Automatic word wrapping** If this check box is enabled the text will be wrapped fitting to it's size.
- **Print when expression** This is a logical expression which enables you to define when the Label object is shown or not. See the details later.

### 6.14.2. Label dialog



The following buttons are available for apply or cancel settings: \* **OK** Select to apply your label settings. \* **Cancel** Closes the screen without saving any changes, returning you to the designer desktop.

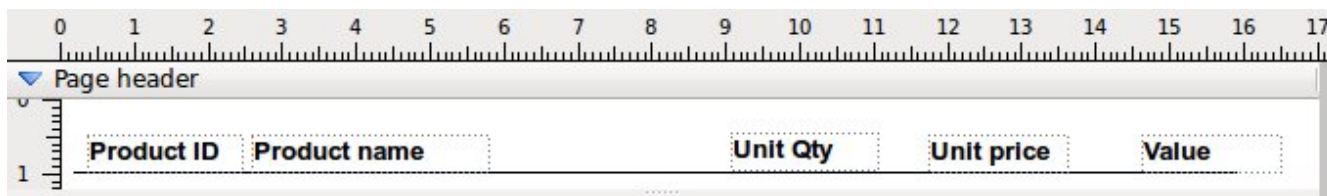
Add the following labels to the Page Header: Product ID, Product name, Unit Qty, Unit price, Value and move them by drag and drop to the place you want to be located. To move the added Label just drag (select) it with left mouse button and drop it to the location you want. To delete a Label, select it and press **Delete** button

### 6.14.3. Adding labels



### 6.14.4. Adding Line

To create a new Line object select the Line button in the tool bar or menu item in **Tools** menu. After that the cursor changes to a cross beam, then click in the section of the report definition where you want the line to be started and simply drag the line to the end position. To move the added line just drag (select) it with left mouse button and drop it to the location you want. To delete the line just select it and press **Delete** button



To open the line properties dialog just double click on the line, on the dialog you may then set the object's properties. In the dialog you are presented with the following options are available:

### Print when expression

This is a logical expression which enables you to define when the Line object is shown or not. See the details later.

## 6.14.5. Adding Fields

Now we have to add the most important items to the report. Field objects contain dynamic information retrieved from a data source, parameter or a variable. To create a new Field object, first select the **Field** tool button or the menu item in **Tools** menu. After that the cursor changes to a cross beam, then click in the section onto you want the Field to be located. This section is in generally the Detail section. Doing so will create the Field object in the specified section at that position and opens the Field property dialog. On the dialog you may then set the Field's properties.

The following options are available for fields:

### Field source type

The combo box contains the possible sources from where the field can pull data. Field's data can be loaded form the following sources: data source, Parameter, Variable, System variable, Expression. About various source types you can find informations in NCReport specification.

### Field column/expression

This property represents the name of the data column from where field's value are pulled. When SQL query data source is used by the field, this name equals the corresponding SQL column name included in SQL query. When other data sources such as Text, this value is often the number of the data column.

### Data type

The field's base data type. The following data types are supported: Text, Numeric, Date, Boolean

### Automatic word wrapping

If this check box is enabled the field will be wrapped fitting to it's size.

### QString::arg() expression

This is a string expression with %1 symbol for the same purpose what ::QString("String %1").arg(value):: code does. The field's value will be embedded into this expression.

### Call function

This feature currently is unavailable.

## Lookup class name

This feature currently is unavailable.

## Print when expression

This is a logical expression which enables you to define when the Field is shown or not. See the details later.

### 6.14.6. Field dialog

Field settings. ID:578FD

Field source type: Data source Data type: Text

Field

customers.CustomerID

Build expression...

General Text Numeric Date/Time

Description / title:

Template arg() string:

Background mode: Transparent Background color: User color 1

☐ Word wrapping ☐ Background only in Designer

☐ Automatic height ☐ Pin to left

☐ Hide repeated values ☐ Pin to right

Advanced Options

Print When: ...

HTML Options: ...

Item ID: 578FD Function ID: Zone ID: 0

Conditional format... OK Cancel

The following table summarizes the various formulas you can specify in fields as field column expression. The formula depends on what field source type you use.

### 6.14.7. Field column formulas

Filed source type	Field column formula	Description
-------------------	----------------------	-------------

Data Source	[data sourceID.]column	The column equals a valid SQL column name in your SQL query. If data sourceID is specified, the report engine will assign the named data source by this ID. If you don't specify data sourceID, the default (currently processing) data source is interpreted you have assigned before to the detail. The Data Source references can contain also functions. Read more in chapter Expressions.
Parameter	parameterName	The name/ID of the parameter
Variable	variableName	The name/ID of the variable
System variable	variableName	The name/ID of the system variable.
Expression	expression	You can use even a complex script expression for the field. Both data source data, Parameters, Variables can be used in expressions. For more informations about expressions see the Using expressions chapter.
Template	template expression	Template is a simple substitution of report items such as data source data, parameter or variable. All of them are joined into one string.

Some properties are available for different data types only. They are located on separated tab widgets within the dialog. The following additive options are available for numeric fields:

### Number formatting

If this option is checked, the number formatting will be turned on

### Use localized settings

If this option is checked, the report engine will use localized number formats by the current application's QLocale settings.

### Blank if value equals zero

If this option is checked, the field's current value will not appear when it's value equals zero.

### Decimal precision

The number of digits after the decimal point.

## Field width

Width of number in digits. Specifies the minimum amount of space that a is padded to and filled with the character fillChar. A positive value will produce right-aligned text, whereas a negative value will produce left-aligned text.

## Format character

This one digit option specifies the format code for numbers. Possibly values are: **e,E,f**. With **e**, **E** and **f**, precision is the number of digits after the decimal point. With 'g' and 'G', precision is the maximum number of significant digits. Used by ::QString::arg( double a, int fieldWidth = 0, char format = 'g', int precision = -1, const QChar fillChar):: function.

## Fill character

specifies the character the numeric value is filled with when formatting. See QString::arg() fillChar parameter.

### 6.14.8. Field dialog - numeric data

Field settings. ID:578FD

Field source type: Data source Data type: Text

Field

customers.CustomerID

Build expression...

General Text **Numeric** Date/Time

☐ Number formatting Decimal precision: 0

☐ Use localized settings Field width: 0

☐ Blank if value equals zero Format character: f

Fill character:

Advanced Options

Print When: ...

HTML Options: ...

Item ID: 578FD Function ID: Zone ID: 0

Conditional format... OK Cancel

The following buttons are available for apply or cancel settings:

- **OK** Select to apply your field settings.

- **Cancel** Closes the screen without saving any changes, returning you to the designer desktop.

To continue our instance report, add the following (four) fields to the detail section. Use the following names and data types in field column expression: ProductID (Numeric), ProductName (Text), QuantityPerUnit (Numeric), UnitPrice (Numeric), Value (Numeric)

#### 6.14.9. Details section with fields



#### 6.14.10. Adding Variables for totals

Before we add variable field to the report, let's see the handling of variables in NCReport. Variables are special items used for providing counts and totals. Each of the variables have name, function type, data type, and have an assigned data source column the variable based on. To add a variable open the **Report** menu and select **Variables...** menu item. Then appears a dialog on you can manage variables.

The following options are available for variables:

- **Variable ID** The name/ID of the variable.
- **Variable expression** This property represents the name of the data column from where variable's value is pulled from.
- **Function type** The function type of the variable. Supported function types: Sum, Count Count: The COUNT type of variable will increment by 1 for every detail row. Sum: The SUM (summary) variable will summarize the value of the specified data column returned by the field
- **Reset scope** If this check box is enabled the field will be wrapped fitting to it's size.
- **Initial value** Initial value of the Variable

#### 6.14.11. Variable dialog



The dialog box is titled "Variable settings". It contains a table with the following data:

ID	Expression	Function	Scope
1 CustomerTotal	Value	SUM	Group
2 OrderTotal	Value	SUM	Group

Below the table is a section titled "Variable properties" with the following fields:

- Variable ID: CustomerTotal
- Variable expression: Value
- Function type: SUM
- Reset scope: Group
- Initial value: 0

Buttons: Add, Remove, OK, Cancel.

The variables added to report are shown in the variable list view. Clicking on the list items the selected item becomes active. To delete the selected item just select the **Remove** button. The following buttons are available in the dialog:

### Add

Adds a new variable and enable the variable options to edit.

### Remove

Deletes the variable selected from the list

- **OK** Select to save your variable settings.
- **Cancel** Closes the dialog without saving any changes, returning you to the designer desktop.

Add a new variable by clicking the **Add** button and then specify the options by followings: Variable ID: total\_value, Variable expression: value, Function type: **SUM**, Reset scope: **Group** To add total first, we should add a new group to the detail. In the next section we explain how to use the grouping feature.

## 6.14.12. Adding group to detail

While most reports can be defined using a single Detail section having multiple columns and rows of data, others - just like our example report - require summary data, totals as subtotals. For reports requiring summary data, NCReport supports Group sections. Group sections have the following characteristics:

- Always associated with a Detail section

- Defined by Group Headers and Group Footers
- Group Headers always print above it's Detail section
- Group Footers always print below it's Detail section
- Reference database column on which Group Headers and Group Footers will break
- Force new Group Header each time the value of the referenced column changes
- Force a new Group Footer each time the value of the referenced column changes
- Unlimited level of groups allowed

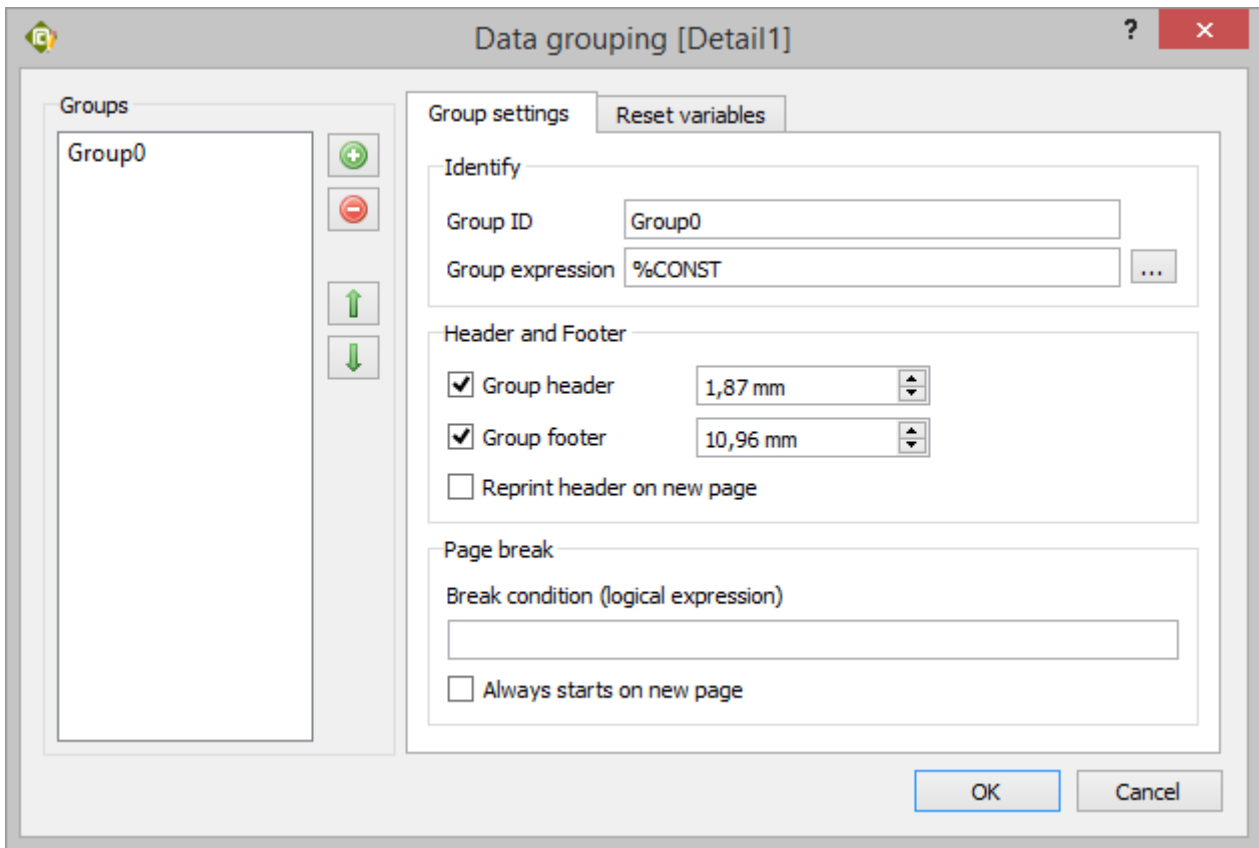
In the group dialog the groups added to the report are shown in the order you have added. The added group sections will appear in the designer after you applied the group settings. Groups are structured hierarchically. The first group will be the primary level of group, the second one is the second level and so on.

To add a new group to the detail, open the **Report** menu and select **Details and grouping....** Then the Detail settings dialog will appear. Select the Detail1 detail in the list, then to open the grouping dialog click on **Data grouping...** button. The Group settings dialog appeared, always belongs to the previously selected detail. To add a new group click on the **Add** button.

The following additive options are available for a group:

- **Group ID** The name/ID of the group for identification purposes
- **Group expression** The name of the data source column the group is based on. If the value of this referenced column changes, the group breaks. Also constant values such as 0 or 1 can be used as group expression. Then the group will never break just ends. This could be very useful for end-total fields.
- **Header and Footer** To enable or disable group header and footer, check on or off the specified check box. To set initial height of these sections you can use spin boxes near the check boxes.
- **Reset variables** This list contains the variable names are available to reset when the group ends. The variables that have Report reset scope status are visible only in the list.

### 6.14.13. Group dialog



The groups added to a detail appear in the group list. Clicking on the list items the selected item becomes active. To delete the selected group just select the **Remove** button. The following buttons are available in the dialog:

- **\*Add** \*Adds a group and enables the group options to edit.
- **Remove** Removes the group selected from the list
- **OK** Select to save your group settings.
- **Cancel** Closes the dialog without saving any changes, returning you to the Detail settings dialog.

So, let's add a new group with the following specification: Group ID: Group0, Group expression: 0, Show group header and footer, Reset total\_value variable. After you select **OK** button, the new group sections (header and footer) will appear in report document. Close the Detail settings dialog by clicking **OK** button.

## 6.15. Adding total variable field

Now we have a defined group with header and footer. Group footers in general a sections are usable for showing totals and subtotals. Let's add a new field to the report footer with the following parameters:

### Field source type

Variable, Field column: total\_value, Data type: Numeric

Now we have got almost all of fields we need. What we have to do also is just adding some missing lines, labels and adjusting the report.

## 6.16. Other items

We summarize the tasks below:

- Add a Total value: Label to the report footer section near the total field.
- Add a Line above the totals.
- Move the items adjusted to the appropriate columns.
- Add a Line to the Page footer similar to the line in the Page header
- Add a Field to the Page footer: Field source type: System variable, Field column: pageno, Data type: Numeric, QString::arg() expression: Page: %1

## 6.17. Adjustment and formatting

To finish the report now we can format and adjust the items. Here are tasks you should also do:

- Adjust the height of the sections for the fitting size by mouse dragging the base line of section or by geometry editor. The height of the detail is important, since it is often recurred many times.
- Select the labels in Page header and set the font weight to bold by clicking the **Bold** tool button in tool bar. Item multi-selection may used.
- Select ProductID field in Detail section and set its font weight to bold.
- Select and align right all of numeric fields to right by clicking the **Align right** tool button in tool bar.
- Set the number format options for numeric fields: Number formatting: on, Decimal precision: 2
- Set also Use localized settings on for value and total\_value fields

Save the report. Now you should get something similar this:

## 6.18. Report is ready

Page header				
Product ID	Product name	Unit Qty	Unit price	Value
Group header [Detail1.Group0]				
Detail [Detail1]				
ProductID	ProductName	QtyPerUnit	UnitPrice	Value
Group footer [Detail1.Group0]				
			Total value:	total_value
Page footer				
				pageno

## 6.19. Adding Variables for Totals

Before we add variable field to the report, let's see the handling of variables in NCReport. Variables are special items used for providing counts and totals. Each of the variables have name, function type, data type, and have an assigned data source column the variable based on. To add a variable open the **Report** menu and select **Variables...** menu item. Then appears a dialog on you can manage variables.

The following options are available for variables:

### Variable ID

The name/ID of the variable.

### Variable expression

This property represents the name of the data column from where variable's value is pulled from.

### Function type

The function type of the variable. Supported function types: **Sum, Count**

Aggregate functions:

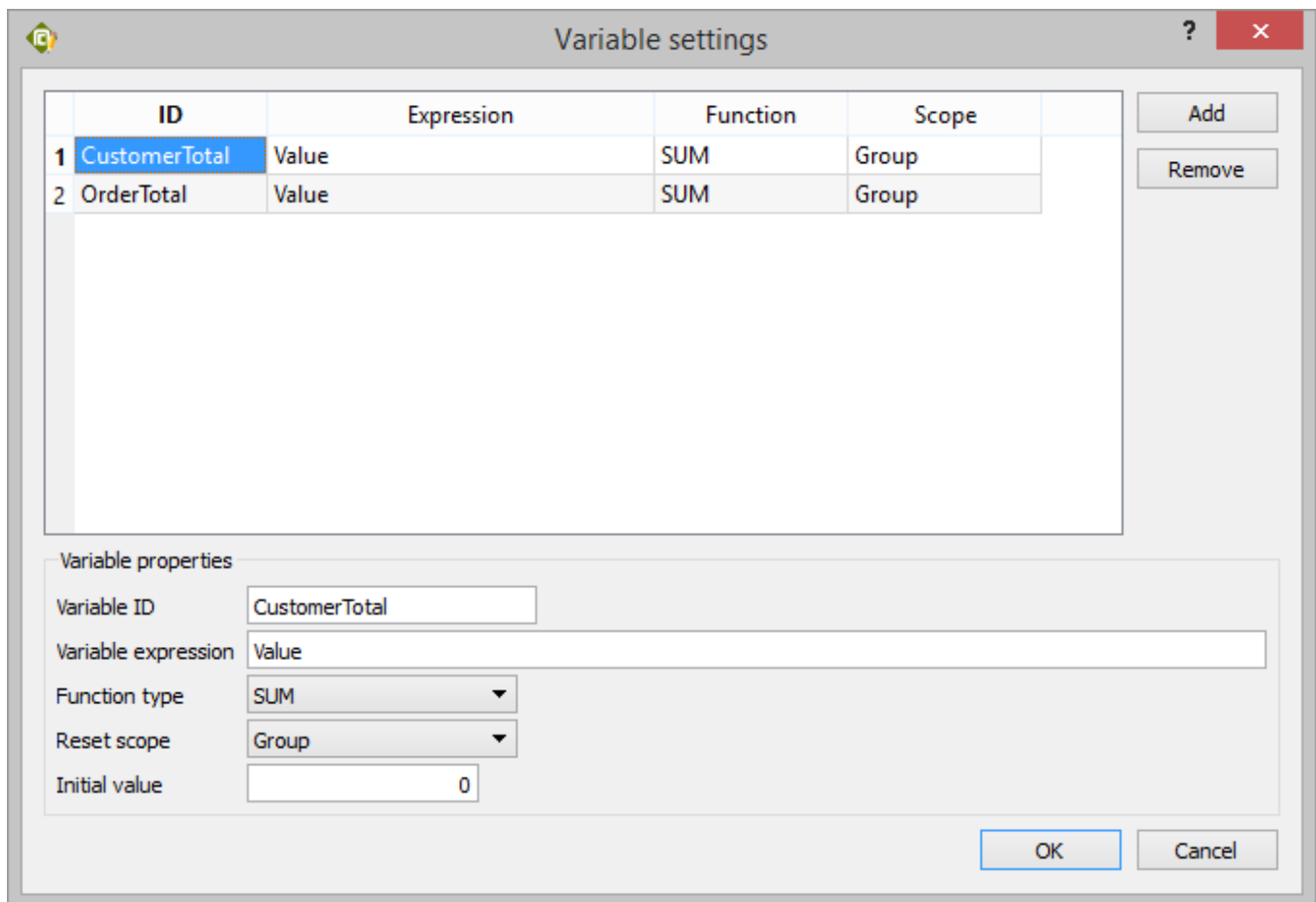
### Count

The COUNT type of variable will increment by 1 for every detail row.

### Sum

The SUM (summary) variable will summarize the value of the specified data column returned by the field Reset scope` If this check box is enabled the field will be wrapped fitting to it's size.  
Initial value` Initial value of the Variable

### 6.19.1. Variable dialog



The variables added to report are shown in the variable list view. Clicking on the list items the selected item becomes active. To delete the selected item just select the **Remove** button. The following buttons are available in the dialog:

**Add** Adds a new variable and enable the variable options to edit. **Remove:** Deletes the variable selected from the list

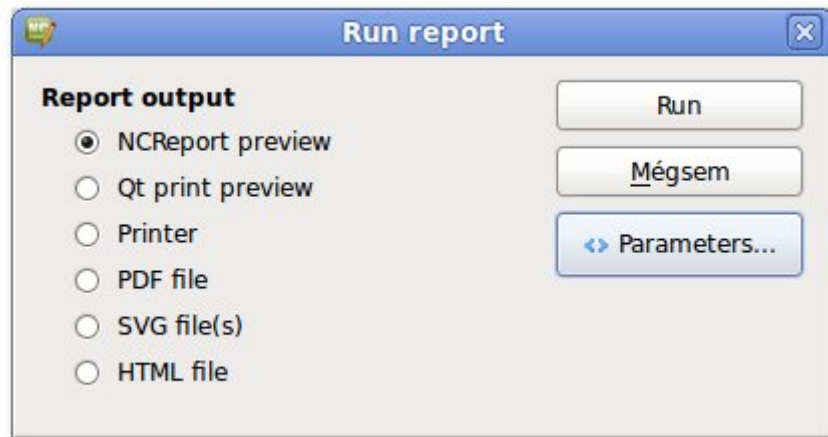
- **OK** Select to save your variable settings.
- **Cancel** Closes the dialog without saving any changes, returning you to the designer desktop.

Add a new variable by clicking the **Add** button and then specify the options by followings: Variable ID: total\_value, Variable expression: value, Function type: SUM, Reset scope: Group

To add total first, we should add a new group to the detail. In the next section we explain how to use the grouping feature.

## 6.20. Running the report

For running report from the Designer window open **Report** menu and select **Run report...** menu item. Then the report runner dialog will appear. You may add and remove parameters by **Add/Remove** buttons. About parameters in example see the next section. Select the output where you want the report to go to and then start the report by clicking **OK** button



Running the report to Preview window now you should see something similar:

### 6.20.1. Preview output - page 1

The 'NCRReport 2.6.1 preview' window displays a table with 5 columns: Product ID, Product name, Unit Qty, Unit price, and Value. The table contains 18 rows of data. The window has a menu bar (File, View, Navigate, About), a toolbar with icons for print, save, zoom, and navigation, and a status bar at the bottom indicating 'Page: 1/2'.

Product ID	Product name	Unit Qty	Unit price	Value
3	Aniseed Syrup	10.00	9.00	90.00
4	Chef Anton's Cajun Seasoning	22.00	20.00	440.00
5	Chef Anton's Gumbo Mixj	21.35	19.00	405.65
6	Grandma's Boysenberry Spread	25.00	21.00	525.00
7	Uncle Bob's Organic Dried Pears	30.00	25.00	750.00
8	Northwoods Cranberry Sauce	40.00	34.00	1360.00
9	Mishi Kobe Niku	97.00	72.00	6984.00
10	Ikura	31.00	26.00	806.00
11	Queso Cabrales	21.00	19.00	399.00
12	Queso Manchego La Pastora	38.00	32.00	1216.00
13	Konbu	6.00	5.00	30.00
14	Tofu	23.00	21.00	483.00
15	Genen Shouyu	15.50	14.00	217.00
16	Pavlova	17.45	15.00	261.75
17	Alice Mutton	39.00	33.00	1287.00
18	Camarvon Tigers	62.50	46.00	2875.00

The second page:

### 6.20.2. Preview output - page 2

NCReport 2.6.1 preview

File View Navigate About

100 % Page: 2/2 Close

74	Longlife Tofu	10.00	9.00	90.00
75	Rhanbrau Klosterbier	7.75	7.00	54.25
76	Lakkalikaari	18.00	16.00	288.00
77	Original Frankfurter grane Soae	13.00	11.00	143.00
Total value:				98 013,66

Page: 2

Page: 2/2



# Advanced Features

To create more complex, professional reports, we need additional features and functions. This section describes these important advanced functions of the NCReport reporting system.

# Chapter 7. Data/Script Expressions

NCRReport since 2.0 version can handle data or script expressions using Qt Script module the new powerful feature of Qt>=4.3. Qt Script is based on the ECMAScript scripting language, as defined in standard ECMA-262. Fields can even contain script codes instead of a simple data source column, parameter or variable. In this case the report engine evaluates the specified script code each time when the fields are refreshed. Report items can also have **Print only when expression is true** (short name: **printWhen**) property. Print when expressions are script expressions too but they must always return boolean result. To use script expression in fields you have to set **Expression** field source type in the **Field property dialog**



In Qt6 version the Qt Script module is no longer available. Instead, the QJSScript can be used, so the report script expressions are going to be based on QJSScript (Qt's Javascript Engine). The Javascript grammar is a little bit different than Qt Script grammar. You may need to upgrade some script expressions in your report if you migrate to Qt6

## 7.1. Using references in expressions

Expressions can contain the following references: Data source data, Parameter, Variable, Field result. When expressions are evaluated the references always replaced with their current value. The syntax formats of the references are the following:

Syntax	Description
<code>\$D{[datasourceID.]column}</code>	Data source column reference. Returns the current value of the data source column from the current data row/record. If <code>datasourceID</code> is not specified the default current data source (what is assigned to the current detail) is considered.
<code>\$P{parameterID}</code>	Parameter reference. Returns the value of the parameter by name/ID
<code>\$V{variableID}</code>	Variable reference. Returns the current value of the variable by name/ID.
<code>\$F{fieldID}</code>	Field reference. Returns the current display value of the specified Field. FieldID is the auto generated but editable ID value of the field generated first when it's added to a section.



If an expression contains any inserted reference with string/text type, quote marks are needed at the beginning and the end of the token. For example: `"$D{ds.lastname}"=="Smith"` You don't need quote marks for numeric or boolean values, for example `$D{price}=750.0` is correct formula

## 7.2. References in templates

Template is a text in which you can simply embed any data reference without using script formulas. Templates can be used in fields as a source type or in rich texts if template mode is on.

Example of using data references in templates:

```
First name: $D{datasource1.firstname} Last name: $D{datasource1.lastname} Date from:
$P{date_from}
```

## 7.3. Reference examples

Example of using script expression in fields

```
"$D{datasource1.productName}+" first string "+" second string "+"$P{parametername}"
```

Example of using script expression as **"Print only when expression is true"** property. The expression must return logical value.

```
$D{productPrice}<1500
```

Example of using data references in templates:

```
"Date period: $D{ds.datefrom} - $D{ds.dateto}"
"Dear $D{ds.firstname} $D{ds.lastname},"
```

## 7.4. Testing Field Expression

Now we try out how expressions are working with Fields. We use our last report example. Let's open the report in the designer and select the productName Field in the detail section. Open the Field properties dialog by double clicking on the field item. Change the Field source type to Expression and then the Field column expression we modify to the following script expression:

```
if ($D{ProductID}>40) "Product: "+"$D{ProductName}"; else "";
```

## 7.5. Field expression

Field source type:  Data type:

Field

```
if ($D{ProductID}>40) "Product: "+$D{ProductName}"; else "";
```

In this case the report engine first replaces the references in the code and then evaluates the script code before each rendering action. Close the dialog by clicking **OK** button and then save the report. Now we just run report to preview window. Let's see the result:

## 7.6. Result of field expression

Product ID	Product name	Unit Qty	Unit price	Value
17		39.00	33.00	1 287,00
3		10.00	9.00	90,00
40		18.40	16.00	294,40
60	Product: Camembert Pierrot	34.00	28.00	952,00
18		62.50	46.00	2 875,00
38		263.50	131.00	34 518,50
39		18.00	16.00	288,00
4		22.00	20.00	440,00
5		21.35	19.00	405,65
48	Product: Chocolate	12.75	11.00	140,25
58	Product: Escargots de Bourgogne	13.25	12.00	159,00
52	Product: Filo Mix	7.00	6.00	42,00
71	Product: Flotemysost	21.50	19.00	408,50
33		2.50	2.00	5,00
15		15.50	14.00	217,00
56	Product: Gnocchi di nonna Alice	38.00	32.00	1 216,00

This example is spectacular but not the most effective way of using expressions. In most cases when you use expressions in fields you don't need too complex code. If you need a condition by your field should be visible or not, we recommend to use **"Print only when expression is true"** feature instead. We test this feature in the next section.

## 7.7. Print When Expressions

This is always a logical script expression that returns true or false. It is considered only if defined for a specified item or a section. When the PrintWhen expression is set the specified item (or section) will be printed ONLY WHEN the expression returns true. As usual the expression can contain any data source, parameter or variable reference.

Examples:

Hide an item when DataSource1.intcolumn data source value is less than 10:

```
${DataSource1.intcolumn}>=10
```

A boolean column print When example:

```
${DataSource1.boolcolumn}
```

Your item appears when DataSource1.stringcolumn is not empty:

```
"${DataSource1.stringcolumn}">"
```

## 7.8. Testing Print when expression

So, print when script expressions are codes that return boolean result. They often called as logical expressions. To test it just open the Field properties dialog by double clicking on the same field item. Type the following code to Print when logical expression:

```
${ProductID}>40
```

## 7.9. Print only when expression is true condition

**Print only when expression is true:**

```
${ProductID}>40
```

Zone ID:

Then modify the previous Field column expression by the following:

```
"Product: "+"${ProductName}"
```

After you validate the settings and save the report run the report again. We have to get the same result.

## 7.10. Templates in Fields and Texts

Templates are special expressions when the data references are simply included in a text. It is not a script hence you cannot use script language elements but data source, parameter and variable references only. For example:

```
Customer name: ${ds1.name} Address: ${ds1.address}  
Interval: ${P{datefrom}} - ${P{dateto}}
```



Template expressions are faster than script expressions because it requires no

evaluation but a simple insertion only.

## 7.11. Script expressions in special locations

Instead of data reference, parameter reference or variable, it is even possible to use script expression at some special locations: as group expression, text item. This feature makes the grouping or text manipulation even more flexible.

### 7.11.1. Script as group expression

If you want to use a direct script as group expression you can write a script expression between `<%` `%>` markers. Having a single line property it's not recommended to build too complex expression here. Example:

```
<% if (${D{ds.age}<12) "KID"; else "MALE"; %>
```

You don't need the markers if you assign to a pre-defined script expression:

```
$$S{myscript}
```

### 7.11.2. Script in HTML Texts

The HTML text item has many flexible feature in terms of scripting. You can embed data references or expressions if template mode is turned on. When you want to include a script expression within a text you need to use the `<%` `%>` markers. Within the markers you can write a script, the result of the script will be inserted into the text at the script location.

## 7.12. Data Source Functions

Field expressions can contain some simple function references. These functions helps to apply basic operations on data or getting meta information from data sources. The functions can be data source level and data column level functions. The function name you may insert after the data source ID or the column ID depending on the function. It is separated by a dot character.

## 7.13. Data Source related (meta) functions

A data source function syntax:

```
DataSourceId.function()
```

In scripts or templates:

```
$$D{DataSourceId.function()}
```

**rowCount()**

Returns the number of rows of the data source.

Example:

```
products.rowCount() or ${products.rowCount()}
```

**isAvailable()**

Returns the isAvailable() method result of the data source class.

Example:

```
products.isAvailable() or ${products.isAvailable()}
```

**isValid()**

Returns the isValid() method result of the data source class.

Example:

```
products.isValid() or ${products.isValid()}
```

**isEmpty()**

Returns true if the data source has no data record.

Example:

```
products.isEmpty() or ${products.isEmpty()}
```

**isNotEmpty()**

Returns true if the data source has at least 1 data record.

Example:

```
products.isNotEmpty() or ${products.isNotEmpty()}
```

**update()**

Forces the update() function on the data source. This can be useful if you may want to manually update a user defined data source. This function use carefully.

Example:

```
products.update() or ${products.update()}
```

## 7.14. Data Source Column related (Value) functions

The data source column functions are introduced for helping some basic text operation.

The data source column function syntax:

```
DataSourceId.column.function()
```

In a Detail section when using the assigned data source:

```
column.function()
```

In scripts or templates:

```
$D{DataSourceId.column.function()}
```

In a Detail section when using the assigned data source:

```
$D{column.function()}
```

### **MID(n,m)**

Returns a string that contains n characters of this string, starting at the specified m position index.

Example:

```
DataSource1.firstname.MID(2,5) or $D{DataSource1.firstname.MID(2,5)}
```

### **LEFT(n)**

Returns a substring that contains the n leftmost characters of the string.

Example:

```
DataSource1.firstname.LEFT(3) or $D{DataSource1.firstname.LEFT(3)}
```

### **RIGHT(n)**

Returns the isValid() method result of the data source class.

Example:

```
DataSource1.firstname.RIGHT(2) or $D{DataSource1.firstname.RIGHT(2)}
```



## USERFUNC()

Executes the `NCReportDataSource::getUserFunctionValue(value, arguments)` method and returns its value. You may want to use it a custom implemented data source.

Example:

```
DataSource1.lastname.USERFUNC() or $D{DataSource1.lastname.USERFUNC()}
```

# Chapter 8. Script Editor

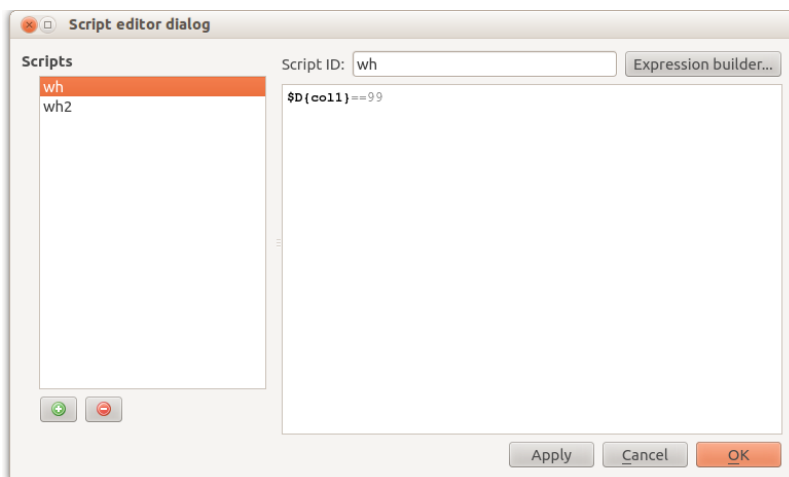
This function provides the ability to store predefined custom script functions within the report. The goal of storing scripts in the report is to avoid duplicating script code assigned to report items or anywhere in the report's scope. For example, it can be used efficiently for "print when" expressions.

To reference the script anywhere in the report, use the following token: `$$scriptId` Where *scriptId* is the ID you assigned to the script.



Use predefined scripts when you need to define complex code or use a script expression more than once in a report. This is typically useful for multiple instances of the same "print when" expression.

To define your scripts in the designer, open the *Report > Scripts...* menu item (the equivalent tool button is found on the toolbar). A dialog will appear where you can add, edit, and remove scripts. To add a new script, click the *Add* button and type the script ID. To remove a script, choose *Remove*. If you want to use the expression builder, select *Expression builder...* to open the expression builder dialog.



## 8.1. Script ID

The script identifier. When you call the script anywhere in the report, you can reference it with this ID. For example: `$$myscript1`

## 8.2. Script Definition

Insert your code here. You can use any report data reference in the code according to the usual rules of NCReport data reference.

## 8.3. Available Buttons

### Apply

Select to apply your settings without closing the dialog.

**OK**

Select to apply your settings and close the dialog.

**Cancel**

Closes the editor without saving any changes, returning you to the designer desktop.

# Chapter 9. Data Formatting

You can specify formatting options within fields. Fields support **string**, **integer**, **float**, **date**, **datetime** data types. All of them often require formatting. NCReport can format data in fields, you can set the format options in the Designer at the **Field Settings Dialog** → **Text**, **Numeric**, **Date/Time** tabs depending on what data type we specify in the **Data type** option combo box.

## 9.1. Text formats

If the **Data type** is text use the **Text** tab for special format settings, The base (font, size, alignment) settings can be done by the toolbars. We can specify the following parameters:

### Vertical alignment

The vertical alignment of the text block: Top, Center, Bottom

### Character length

We can explicitly set the string width (number of characters) if the fill character is going to be used.

### Rotation angle

The data can be rotated. 0 is the default position, 90 or 270 make the text vertical

### Fill character

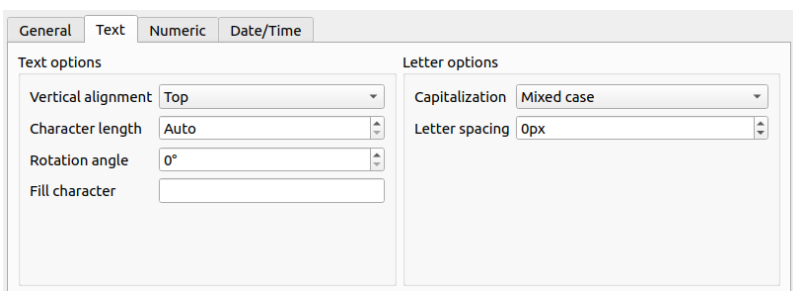
It's possible to fill the empty characters with a specified character.

### Capitalization

Mixed case (default), All uppercase, All lowercase, Small caps, Capitalize

### Letter spacing

Spacing between letters in pixels: 0px is the default.



## 9.2. Numeric formats

If the **Data type** is numeric use the **Numeric** tab for special format settings is necessary. The base (font, size, alignment) settings can be done by the toolbars. We can specify the following parameters:

### Numeric formatting

You can enable or disable the number formatting. It's turned of by default.

## Use localized settings

If this is turned on the number formatting gets based on the QLocale settings

## Blank if value equals zero

if it's turned on and the numeric data equals zero, it will show nothing, the zero value gets hidden.

## Substitution string if equals zero

Turn on this setting if you want to exchange the zero values to a string symbol. For instance you can set '-'

## Substitution string

You can specify the substitution string here.

## Decimal precision

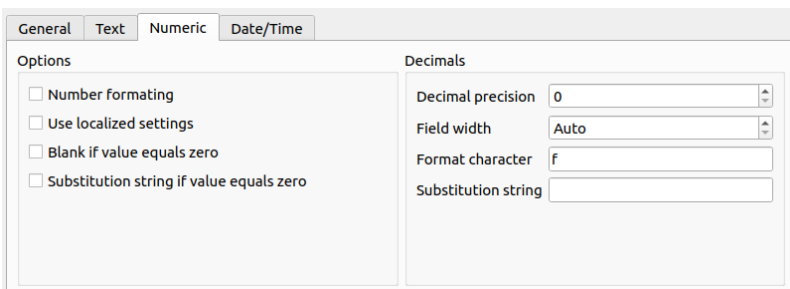
This integer number holds the number of decimal places. It is considered only if the formatting is turned on.

## Field width

Number field width in characters

## Format character

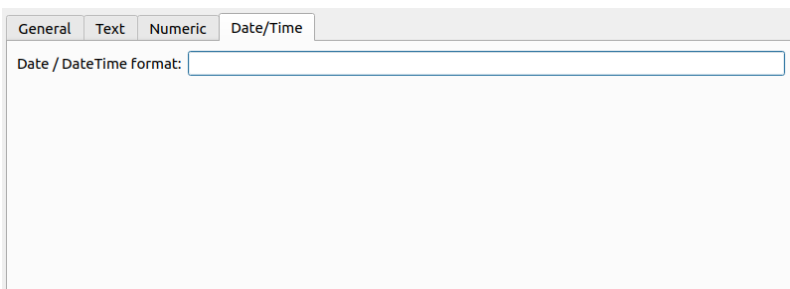
Formatting character: **f** or **g**



The screenshot shows the 'Numeric' tab of a settings dialog. It has four sub-tabs: 'General', 'Text', 'Numeric' (selected), and 'Date/Time'. The 'Options' section on the left contains four checkboxes: 'Number formatting' (checked), 'Use localized settings' (unchecked), 'Blank if value equals zero' (unchecked), and 'Substitution string if value equals zero' (unchecked). The 'Decimals' section on the right contains four input fields: 'Decimal precision' (set to 0), 'Field width' (set to Auto), 'Format character' (set to f), and 'Substitution string' (empty).

## 9.3. Date Formats

In a Field you can of course represent a date or date time data. The date can be formatted in the **Field settings** dialog at the **Date/Time** tab. You can specify the date format in two modes: statically or dynamically



The screenshot shows the 'Date/Time' tab of the same settings dialog. It has the same four sub-tabs: 'General', 'Text', 'Numeric', and 'Date/Time' (selected). The 'Date / DateTime format:' label is followed by a large, empty text input field.

### 9.3.1. Static Date Format

The static date format is based on the Qt date format expression:

Expression	Output
d	The day as a number without a leading zero (1 to 31)
dd	The day as a number with a leading zero (01 to 31)
ddd	The abbreviated localized day name (e.g. 'Mon' to 'Sun'). Uses the system locale to localize the name, i.e. QLocale::system().
dddd	The long localized day name (e.g. 'Monday' to 'Sunday'). Uses the system locale to localize the name, i.e. QLocale::system().
M	The month as a number without a leading zero (1 to 12)
MM	The month as a number with a leading zero (01 to 12)
MMM	The abbreviated localized month name (e.g. 'Jan' to 'Dec'). Uses the system locale to localize the name, i.e. QLocale::system().
MMMM	The long localized month name (e.g. 'January' to 'December'). Uses the system locale to localize the name, i.e. QLocale::system().
yy	The year as a two digit number (00 to 99)
yyyy	The year as a four digit number. If the year is negative, a minus sign is prepended, making five characters.

### 9.3.2. Dynamic Date Format

The dynamic date format uses a format string from one of the data source types, it can be a parameter, data source reference. Use one the normal data embedding formulas:

Example:

```
$D{ds.dformat} or $P{mydateformatparam}
```

Field settings, ID:1

Field source type: Data source Data type: Text

Field

col1

Build expression...

General Text Numeric Date/Time

Date / DateTime format: \$P{mydateformat}

Identify Print when HTML Dynamic position Dynamic style TOC

Item ID: 1 Zone ID: 0

Function ID:

☐ Page break after printing this item

Conditional format... OK Cancel

The dynamic date format is recognized by the report engine and evaluated during rendering. It's up to you to use parameter or data source for storing a dynamic date format.

# Chapter 10. Zones

This feature is available since version 2.2.0. Zones are virtual bands within a report section. All items can have a specified Zone ID. Items with the same zone id, just like a group, represent a horizontal zone as a virtual band inside the section. When the section's automatic height option is enabled, the report engine will process the rendering of zones in order by zone ID sequentially, one after another. If a content of a zone is empty for example because the **printWhen** expression of all items in the zone return false, then the zone won't be printed and the section will be shrunk. The rendering order of zones matches the order of zone IDs.

To set the Zone ID of a report item use the item property dialogs.

## 10.1. Zone ID in property dialog

Field properties

Field source type: Datasource Data type: Text

Field

text

General Numeric Date/Time Other

Description / title:

Template arg() string:

☒ Word wrapping ☐ Automatic height ☐ Hide repeated values

Rotation angle: 0°

Background mode: Transparent

Background color: User color 1

☐ Design time background

Print only when expression is true:

Zone ID: 2

Mégsem OK

## 10.2. Zones in Design mode

Detail [Detail1]																		
0	col3	col4	col5	col6	col7	col8	col9	col10	col11	col12	col13	col14	col15	col16	col17	Zone 0		
1	col3	col4	col18													Zone 1		
2																		



### *Example 1. Information*

Zones are not visible in design mode. The specified region is determined by only the zone IDs of the report items

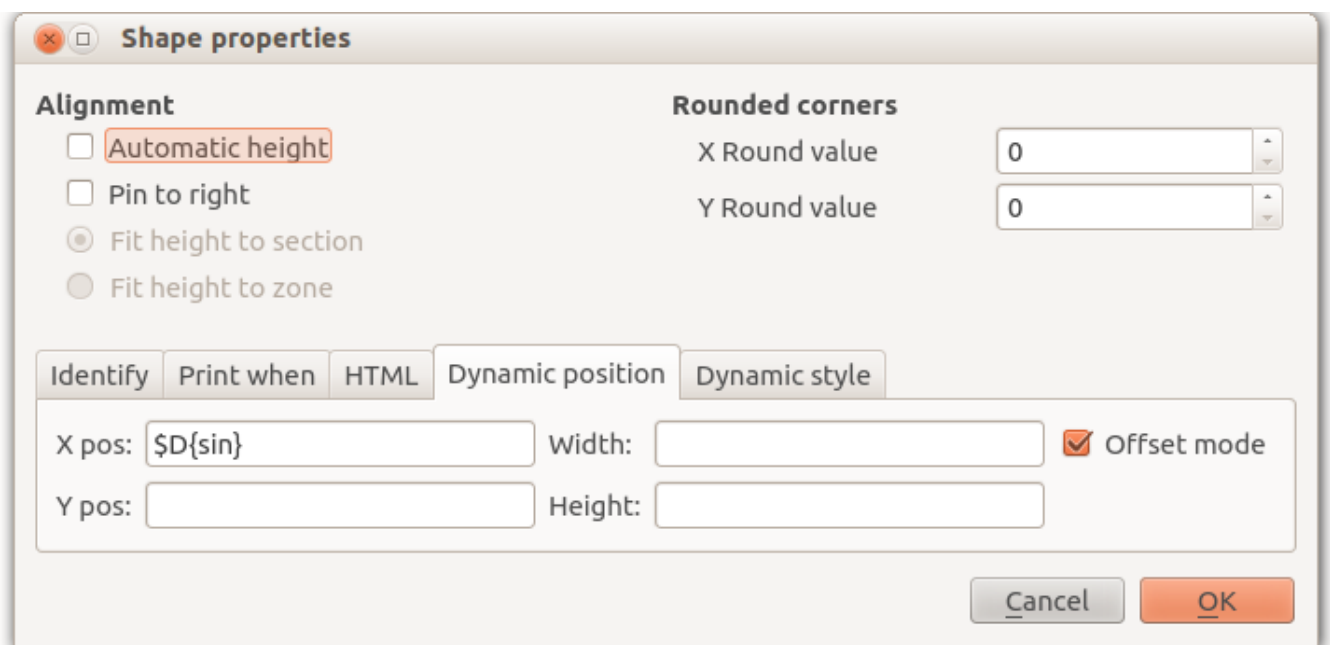
# Chapter 11. Dynamic data driven size and position

This feature makes it easy for you to manage the position and the size of the report items dynamically, driven by a data source, parameter or a variable or even a script expression.

For example it is possible to define the x, y coordinates of objects in the data source, then the positions will be managed by data source. If you want to show graphical objects such as bar, line, you can even use them for generating vertical charts.

To set the item's dynamic position and size management in designer open any report item's setting dialog by double clicking on a report item. Then appears the item settings dialog. Choose the **[ Dynamic position ]** tab of the bottom side property panel.

## 11.1. Dynamic position and size settings



The image shows a 'Shape properties' dialog box with the 'Dynamic position' tab selected. The dialog is divided into several sections. At the top, there are 'Alignment' and 'Rounded corners' sections. The 'Alignment' section has four radio buttons: 'Automatic height' (selected), 'Pin to right', 'Fit height to section', and 'Fit height to zone'. The 'Rounded corners' section has two spin boxes for 'X Round value' and 'Y Round value', both set to 0. Below these are five tabs: 'Identify', 'Print when', 'HTML', 'Dynamic position' (selected), and 'Dynamic style'. The 'Dynamic position' tab contains four input fields: 'X pos:' with the value '\$D{sin}', 'Width:', 'Y pos:', and 'Height:'. To the right of these fields is a checkbox labeled 'Offset mode' which is checked. At the bottom right of the dialog are 'Cancel' and 'OK' buttons.

You can set the following properties. If you leave an option empty - this is the default - the setting is turned off and not considered. The values are always counted in millimeter and relative to their container band's coordinate system.

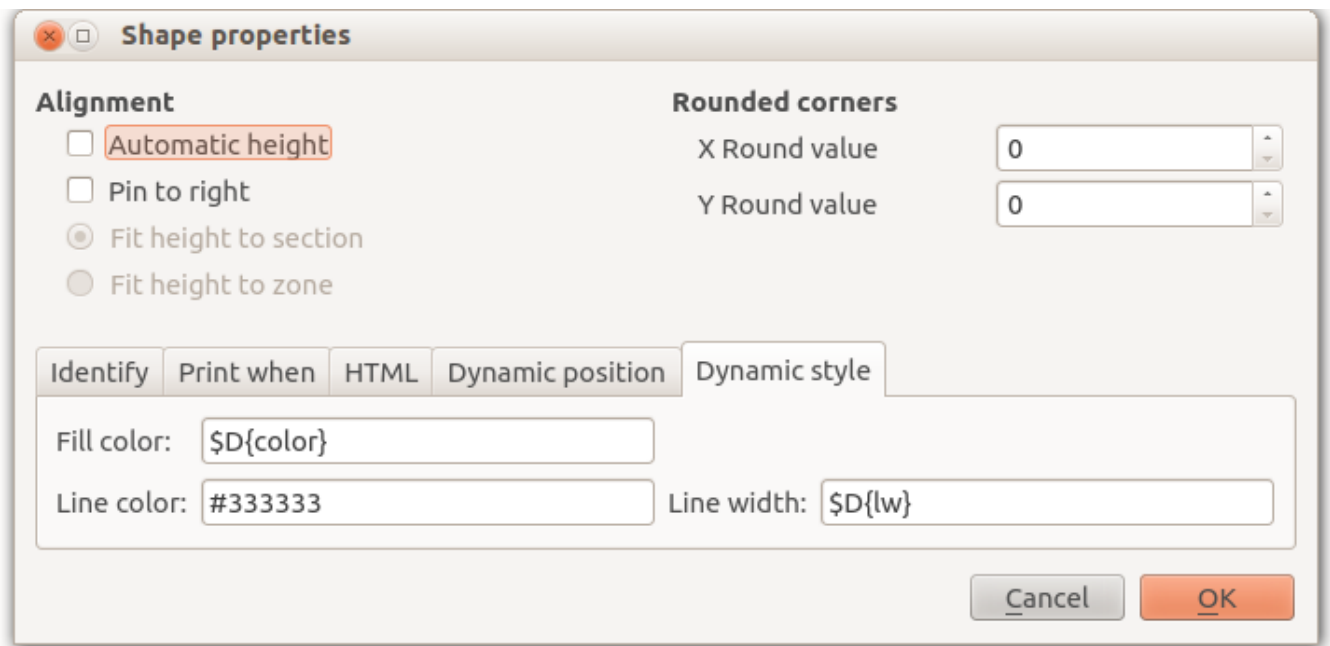
- **X pos** The source of item's x position.
- **Y pos** The source of item's y position.
- **Width** The source of item's width.
- **Height** The source of item's height
- **Offset mode** If this option is enabled then the positions are relative to the original static position, otherwise they are explicit values.

# Chapter 12. Dynamic data driven shape style

The shapes like line, rectangle, ellipse can also have dynamic style. It is possible for you to manage the line width, line color and background color dynamically, driven by a data source, parameter or a variable or even a script expression.

To set the shape item's dynamic style in designer open any shape item's setting dialog by double clicking on an existing item. Then appears the item settings dialog. Choose the [ **Dynamic style** ] tab of the bottom side property panel.

## 12.1. Dynamic style settings



You can set the following properties. If you leave an option empty - this is the default - the setting is turned off and not considered. The width is counted in millimeter.

- **Fill color** The source of item's background/fill color.
- **Line color** The source of item's line color.
- **Line width** The source of item's line width.

# Chapter 13. Page Breaks

Normally the paging is automatic procedure of the report engine but there are various ways to force a page break by an event or a logical expression in the report result. We have the following options to do that:

## 13.1. Detail page break condition

A logical expression can be specified in **Detail setting dialog › General tab › Page Break › Break Condition** If the current result of the expression gets true a page break is induced before the detail section is printed.

## 13.2. Group page break condition

If you define a group in a detail section there is an additional option to break the page by a condition. A logical expression can be specified in **Detail setting dialog › Groups... › General tab › Page Break › Break Condition** If the current result of the expression gets true a page break is induced before the group header section is printed.

## 13.3. Report item page break

Any report item can cause a page break if we turn on the following item's static property: **Report page break after printing this item** In the designer you can turn on this switch by **Report › Edit selected item... › Identify Tab › Report page break after printing this item** If this property is turned on the report engine will break the page after this item gets rendered. Within a section the report items are printed in ID order so users should be able to manage what items should already been printed before the page break.

## 13.4. Report header page break

There is an option of the report header that can statically be turned on or off: **Page break after this section**. If turned on the page breaks after printing the report header. This setting is at **Section properties › Options**

# Chapter 14. Text Document printout mode

TextDocument mode feature allows to render and print multi-page HTML `QTextDocument` based rich texts. The text source can be a file or any data source, so the text can be static, dynamic or even a static template filled with dynamic data. In this mode you can use only a page header, page footer and one or more detail sections in the report definition. The result will be a multi page HTML text correctly broken to pages, printer ready document. In TextDocument mode the `pagecount` system variable can be used by default without any additional setting

## 14.1. Steps of usage

To create a text document printout report use the Designer application

- In the Designer select the menu:[ReportPage options] menu. Then the report page settings dialog will appear. Set the menu:[Report type] combobox to **[ Text document ]**
- Add a text report item into (the only one) Detail section. Set the text's properties by using its property dialog. The text may come from any source as usual.
- Specify the report's page header and footer (Not required)



In this mode, only one detail section within a single text item is supported. The horizontal position and the width of the text item are maintained during rendering.

## 14.2. Text Document printout report example

To see how it's working try `textdocument_printout.xml` demo report. It prints a long Qt class documentation HTML file.

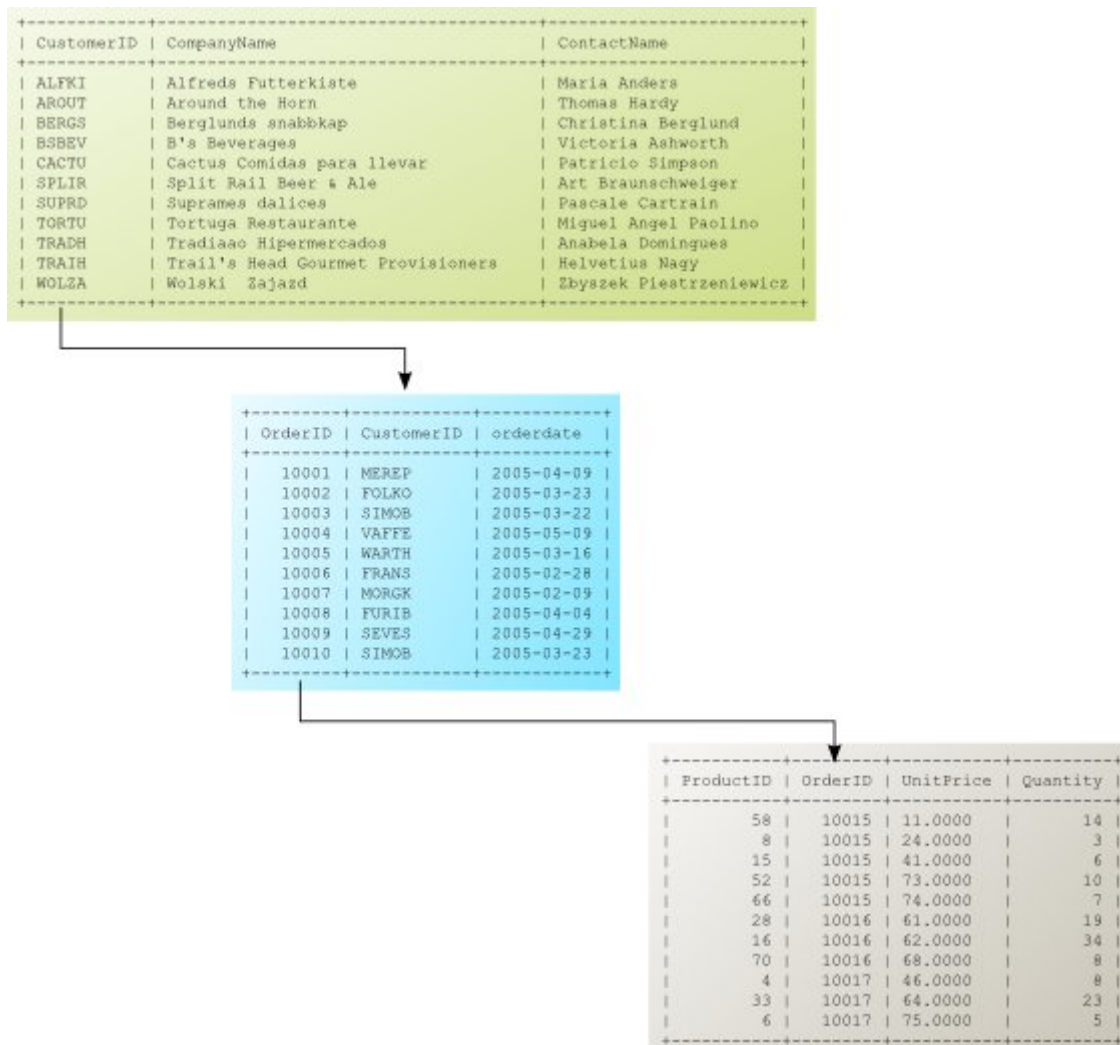
# Chapter 15. Data Relation system

This feature is also named as Sub-Query or Sub-Data Source system Database systems almost always have master/detail data relation between tables. When defining reports for a typical kind of documents such as invoices, orders etc. There are at least one header and a related detail data is used which are linked via primary and foreign key. The goal of the data relation system that child data sources are updated runtime row by row driven by a parent data source. This works by an ID column which is the primary key of the parent and the foreign key of the child. The data source relation is very useful option for SQL data sources where the data are fetched from database tables via SQL command and for Item Model data sources as well where you can manage the data source content from code.



The data source relation system currently works for SQL data source and Item Model data source only. The other data source types are not supported by this feature, expect the [ **Item model** ] data source.

The following example shows a 3 level parent/child structure.



In the the next section you can overview how to define the data sources of master/detail relation. We will create a three level data source relation in the following example.

## 15.1. Defining a parent data source

You can add the master data source in Data Source settings Dialog. In the Designer select menu:[ReportData sources...] and add a new SQL data source. Set the **Opening role** to **[ Beginning of the report ]**. It means that the query will be executed only once at the beginning of the report. Type the data source ID, set the connection properties and edit the SQL query in the SQL editor text box.

This is our Northwind database example master query that queries the customers:

```
SELECT customers.CustomerID, customers.CompanyName, customers.CompanyName
FROM orders
INNER JOIN customers ON orders.CustomerID=customers.ContactName
WHERE OrderDate between '2005-03-01' and '2005-03-31'
GROUP BY CustomerID
```

## 15.2. Defining child data sources

At the same (Data source) dialog we have to add two more data sources within the parent/child structure. Doing the first one add a new SQL data source again. Set the **Opening role** to **[ Child datasource (subquery) ]**. It means that the query will be executed repetitively every time when the next master record is processed. Type the **[ data source ID ]**, set the connection properties in the **SQL connection tab**. After type the **[ Parent datasource id ]** which is the ID of previously defined parent data source. (customers)

### Warning

The **[ Parent datasource id ]** is case sensitive. It must be equal to the already existed parent data source ID

Edit the SQL query in the sql editor text box. This is the 1st child query, it queries the order headers between a date period and is related to a customer:

```
SELECT OrderID, CustomerID, EmployeeID, OrderDate, ShipName
FROM orders
WHERE CustomerID='$D{customers.CustomerID}'
AND OrderDate between '2005-03-01' and '2005-03-31'
ORDER BY OrderID
```

As here can be seen, the data relation is managed by a data reference expression: **\$D{customers.CustomerID}** We have to insert the key value of parent data source into the SQL command.

After comes the second child data source. This is the third level of the relation. Set the **Opening role** to **[ Child data source (sub-query) ]** too and type the **[ Parent data source id ]** which is the ID of

its parent data source (orders). Edit the SQL query in the SQL editor text box. This query retrieves order items are related to a specified order ID:

```
SELECT OrderID, orderitems.UnitPrice, Quantity, Itemno,  
products.productname, orderitems.UnitPrice*Quantity as Value  
FROM orderitems INNER JOIN products ON orderitems.productID = products.productID  
WHERE OrderID=$D{orders.OrderID}  
ORDER BY Itemno
```

At this level the data relation is managed by the following data reference expression: `$D{orders.OrderID}` Accordingly the parent key will always be evaluated and the query is executed when the parent key change occurs. (When its parent row is changed by report processor)

## 15.3. Setting up the detail section

In this step we have to assign the appropriate data source to the Detail section. Doing that open menu:[ReportDetails and grouping...] menu (or the tool button on the toolbar), then appears the Detail section properties dialog. Select the previously defined data source which is the lowest level in hierarchy, in our example: **[ items ]**.



When defining a sub-query, always the lowest level child query should be assigned to the actual Detail section. This because the report engine handles sub-queries by iterating on child level data source records.

## 15.4. Designing the report

After we defined the data sources and assigned them to the Detail we have to add the appropriate groups also to the Detail by using **[ Data grouping... ]** button. As usual each data source level is related to a group level.

Add the other report sections and report items and set the alignments. The following figure appears the ready to run report. (The name of this example report file: `list_of_orders_complex.xml`)

## 15.5. Sub-query report example in Designer



▼ Page header	
1	<b>List of orders</b>
2	<i>This report demonstrates the <b>subquery</b> usage. In other words it's named also "<b>child query</b>" which means the often applied parent-child data <b>relation</b>.</i>
▼ Group header [Detail1.G_OUT]	
	Outmost Group Header   tg.xdata
▼ Group header [Detail1.Group_customer]	
	<b>customers.Custo</b>   <b>customers.CompanyName</b>
▼ Group header [Detail1.Group_order]	
1	<div> <div>\$D{orders.CustomerID} / \$D{orders.ShipName}</div> <div>Order ID: orders.Orc</div> <div>Order Date: lers.OrderDate</div> </div> <div> <div>Itemno</div> <div>Product name</div> <div>Quantity</div> <div>Unit price</div> <div>Total</div> </div>
▼ Detail [Detail1]	
	<div> <div>ItemNo</div> <div>ProductName</div> <div>Quantity</div> <div>UnitPrice</div> <div>Value</div> </div>
▼ Group footer [Detail1.Group_order]	
	Order total: OrderTotal
▼ Group footer [Detail1.Group_customer]	
	customers.CustomerID CustomerTotal
▼ Group footer [Detail1.G_OUT]	
	Outmost Group Footer   tg.xdata
▼ Page footer	
	appinfo   pagenc

The report preview result of our example looks like this: (The name of this example report file: [list\\_of\\_orders\\_complex.xml](#))

## 15.6. Result of a sub-query report example

NCReport 2.6.1 preview

File View Navigate About

100 % Page: 1/55 Close

## List of orders

2011-04-30T21:25:15

This report demonstrates the **subquery** usage. In other words it's named also "**child query**" which means the often applied parent-child data **relation**.

Outmost Group Header ALFKI

### ALFKI

ALFKI / Germany

Order ID: **10692** Order Date: **2005-03-31**

Itemno	Product name	Quantity	Unit price	Total
1	Singaporean Hokkien Fried Mee	43	63,00	2 709,00
Order total:				2 709,00

### ALFKI

ALFKI / Germany

Order ID: **10835** Order Date: **2005-03-03**

Itemno	Product name	Quantity	Unit price	Total
1	Queso Manchego La Pastora	55	59,00	3 245,00
2	Escargots de Bourgogne	13	77,00	1 001,00
Order total:				4 246,00

ALFKI total: **6 955,00**

### ANTON

ANTON / Mexico

Order ID: **10507** Order Date: **2005-03-11**

Itemno	Product name	Quantity	Unit price	Total
1	Rassle Sauerkraut	46	43,00	1 978,00
2	Laughing Lumberjack Lager	12	48,00	576,00
Order total:				2 554,00

### ANTON

ANTON / Mexico

Order ID: **10677** Order Date: **2005-03-26**

Itemno	Product name	Quantity	Unit price	Total
1	Boston Crab Meat	31	26,00	806,00
2	Rassle Sauerkraut	2	33,00	66,00
Order total:				872,00

### ANTON

ANTON / Mexico

Order ID: **10856** Order Date: **2005-03-16**

Itemno	Product name	Quantity	Unit price	Total
1	Ipoh Coffee	19	2,00	38,00
2	Perth	14	42,00	588,00
Order total:				626,00

ANTON total: **4 052,00**

Page: 1/55

## 15.7. Changes in 2.13 version

Data Source Relations has been extended from version 2.13. This is now much better supported general feature. The function has been extended to Item Models. The reports that is created by the old sub-query/relation system are not compatible anymore with the new version of data source relation function.



The reports that uses sub-query function and created in previous version of NCReport, must be upgraded. This function is not compatible with the old report versions.

Changes in the function: (you have to change this in old reports)

- The detail's data source must be the root parent data source
- All fields and expressions must have its data source identifier i.e: `datasource.column`

To use the new data source relation system follow these rules:

- A data source relation can be defined by simply set "child data source" and giving the parent data source id. (as usual)
- 1 parent can have only 1 child (1 to many relation)
- You can specify the primary key column index. If a primary key column is defined for the parent data source, you can use {PK} or {ID} expression in the child data source query. (This is useful only in SQL data sources)
- If you assign a data source relation to a detail section always set the root parent data source to the detail. In earlier version we had to set the last child data source, but it is outdated in 2.13.
- Use `dataSourceUpdateRequest(const QString dataSourceID, const QString foreignKeyValue);` signal to handle data source updates.
- Use `!$D{datasource.isEmpty()}` print when expression of a detail section to hide the empty children data

# Chapter 16. Double pass mode

Double pass mode is a report option that influences the running mode of report engine. When double pass mode is enabled the report is executed two times - this two running cycle is called primary (test) and secondary (real) pass. When the two pass mode is necessary? In normal (1 pass) mode the report generator simply runs the report without anticipatory counting and calculations such as determining the total page numbers.



The `pagecount` system variable always returns zero in 1 pass (normal) mode. If the `pagecount` system variable is needed you have to enable the double pass mode option.

## 16.1. Setting double pass mode

The double pass option is part of the report options are saved into the report definition. To enable or disable this option in Designer select ReportReport and Page Options... To read more: `<xref linkend='pagesettings'/>`.

## 16.2. Example using of `pagecount` variable

Use `$V{pagecount}` expression as field in expression or template mode ore use in text in expression mode

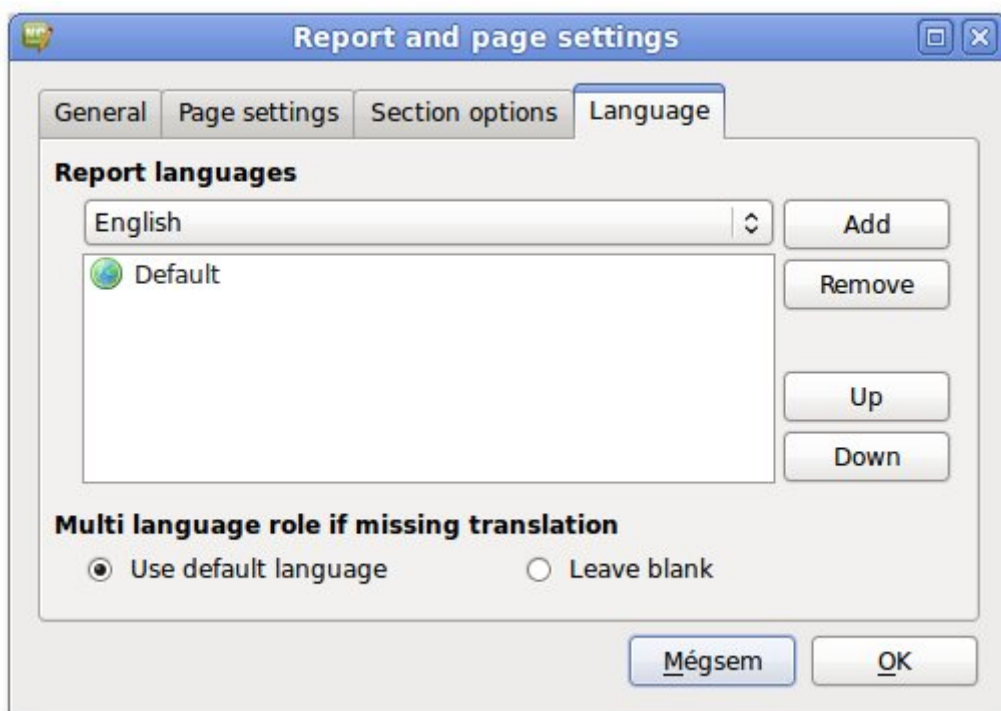
Expression mode example: `$V{pagecount}` Template mode example: Page `$V{pagenum}` of `$V{pagecount}`

# Chapter 17. Internationalization

Since version 2.5, reports have the ability to support multiple languages. This is an important aspect for international applications. The goal of this feature is to allow fields and labels to store multiple texts according to predefined languages.

## 17.1. Adding languages

- To set languages use Report and page settings menu and choose **Language** tab in the dialog.
- To add more languages select the language from combo box and add to language list using **[ Add ]** button

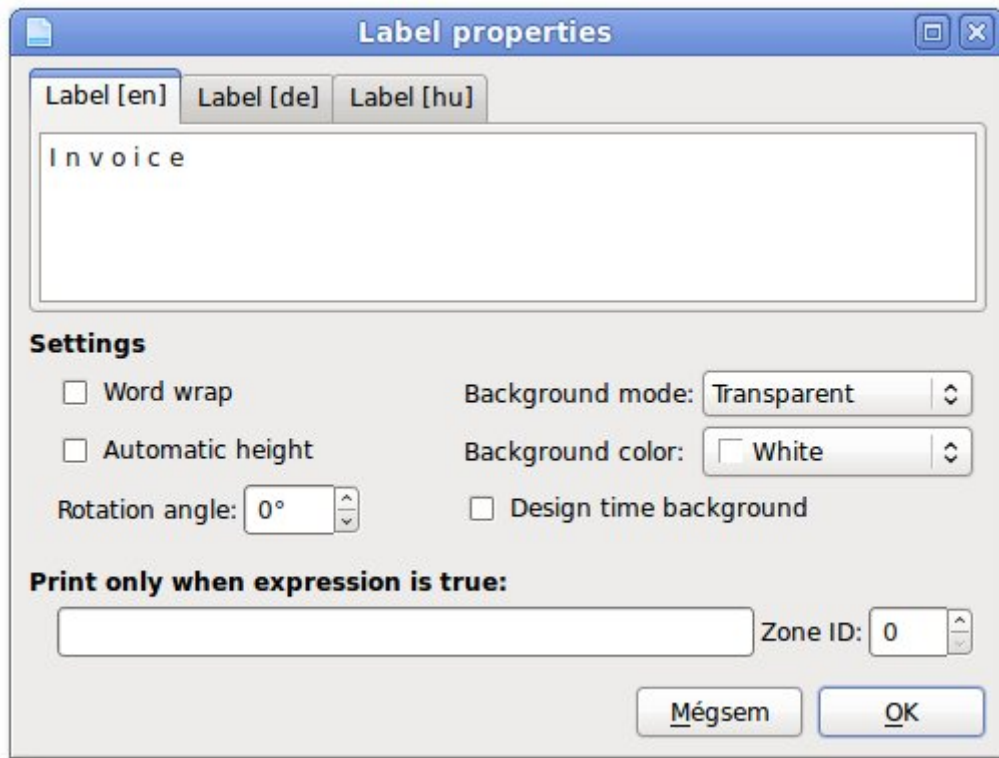


Leave the **Default** language first in the list. This represents the original language of the report.

- Set the **Multi language role**. If not all labels or fields are translated and the current language translation is missing, two options can be chosen. In order to choose **Use default language** the default text will appear otherwise the label or field will not be printed (This is the **Leave blank** option)

## 17.2. Adding translations of Fields or Labels

Insert a Field or Label item as usual. The property dialog appears with tabs of each language that was defined previously. Type the translations to the appropriate language tab control. Empty translation tab means a missing translation.



## 17.3. Setting up the current language

The current language of the report can be set both in design mode and in running mode. In Designer select Report language from the Report menu or the **Languages** tool button from the toolbar and select the language what you want from the submenu

To set the language from application code use `setCurrentLanguage( const QString & langcode )` function where **langcode** is the international two letter language code.

## 17.4. Setting up the language

```
NCReport* report = new NCReport(parent);
report->setLanguage("de");
```

To set the language from console running mode use `-l` command line parameter the international two letter language code.

## 17.5. Setting up the current language from command line

```
ncreport -f myreport.ncr -l de
```

# Chapter 18. Sub-Report iteration

The feature called **Sub-Report** refers to a repeated report generation process that traverses a dedicated data source. The data source type: **subreport**. This approach is similar to the **classic sub-report** model but supports only one level. It is particularly useful when a complex report or a multi-detail report needs to be repeated while processing different data records. The function utilizes a dedicated **parent** data source as the repetition source.

The Sub-Report function provides a great opportunity for creating simple one-to-many relationship reports. Note that you shouldn't set your master data source as **child data source** the report will use the **subreport** data source for repeating the reports.

## 18.1. Sub-Report data source

To set the data source on which the iteration based, you have to add a data source to the report as usual. Set the **Opening role** to **Sub-Report iteration**

## 18.2. Reference to master data source

You can place any reference to master data source in the SQL data source queries. For example:

```
SELECT product.name, product.code WHERE id=$D{master.id}
```

For non SQL data sources such as Item Model data source it is possible to use the SIGNAL/SLOT mechanism. Use the following signal of NCReport object:

```
signals:  
void dataSourceUpdateRequest(const QString& dataSourceID, const QString& data);
```



All data sources are updated repeatedly when a sub-report cycle begins, after the last cycle finished, except the master data source. The function is similar to a parent/child relation but not the same.



# Chapter 19. Table View Rendering

Table View item is a report item destined to rendering QTableView tables with full WYSIWYG print support or rendering an item model content. The function is originally aimed to print the tables in the same rate as the existed QTableView screen widget. The table view item should follow the formats of the original QTableView widget. The cells gets display outlook information from the table's item model. Some basic table settings such as header background, line type, line color, etc. are currently fixed.

## 19.1. Adding TableView item

In Designer to add a TableView item into a section select the **Table View** tool button or menu item from Tools menu. After the cursor changes to a cross beam click in the section where you want the item to be located. The **Table View** item is created and its settings dialog appears.

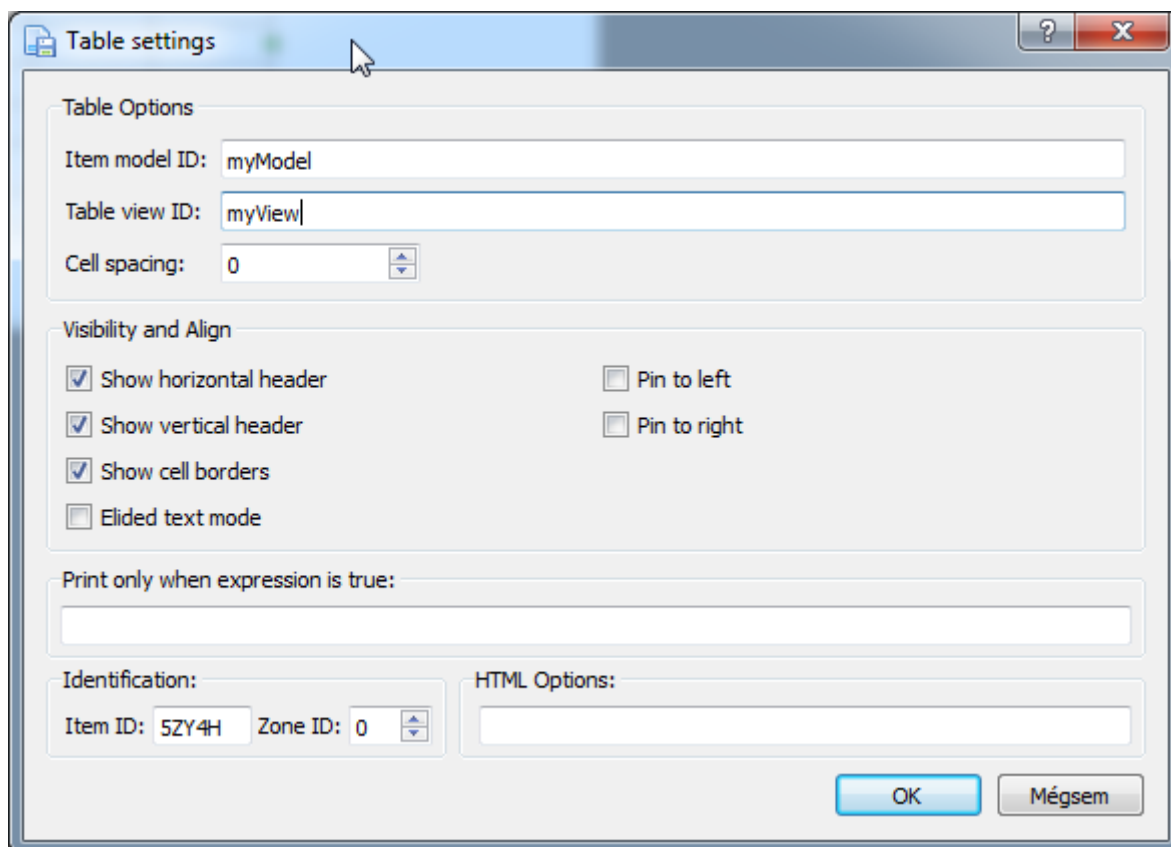


It is strongly recommended to add **Table View** to a Detail section, not into the headers or footers. Since the table may fill the available space both horizontally and vertically, no other report items should add to this section. The table view has its own data source assignment so it is independent from the data source of the detail which should be unique and should have 1 record. You can use an 1 row static text data in a detail section for tables.

Specify the same ID values in the Table View settings dialog that you will apply when setting the table view and the model from code. The report engine will identify the objects by the specified IDs.

## 19.2. Table View Dialog





The dialog options are as follows:

- **Item Model ID** Identifies the model object pointer related to the QTableView.
- **Table View ID** Identifies the QTableView object pointer you want to render.
- **Cell spacing** Spacing value for cells. Has no affect.
- **Show horizontal header** If enabled then the horizontal table header will appear.
- **Show vertical header** If enabled then the vertical table header will appear.
- **Elided text mode** When this option is enabled the multi-line texts will not be rendered, but partially the first line only with three dots.
- **Pin to left** The table will automatically be adjusted to the left margin.
- **Pin to right** The table will automatically be adjusted to the right margin.

## 19.3. Setting the object references

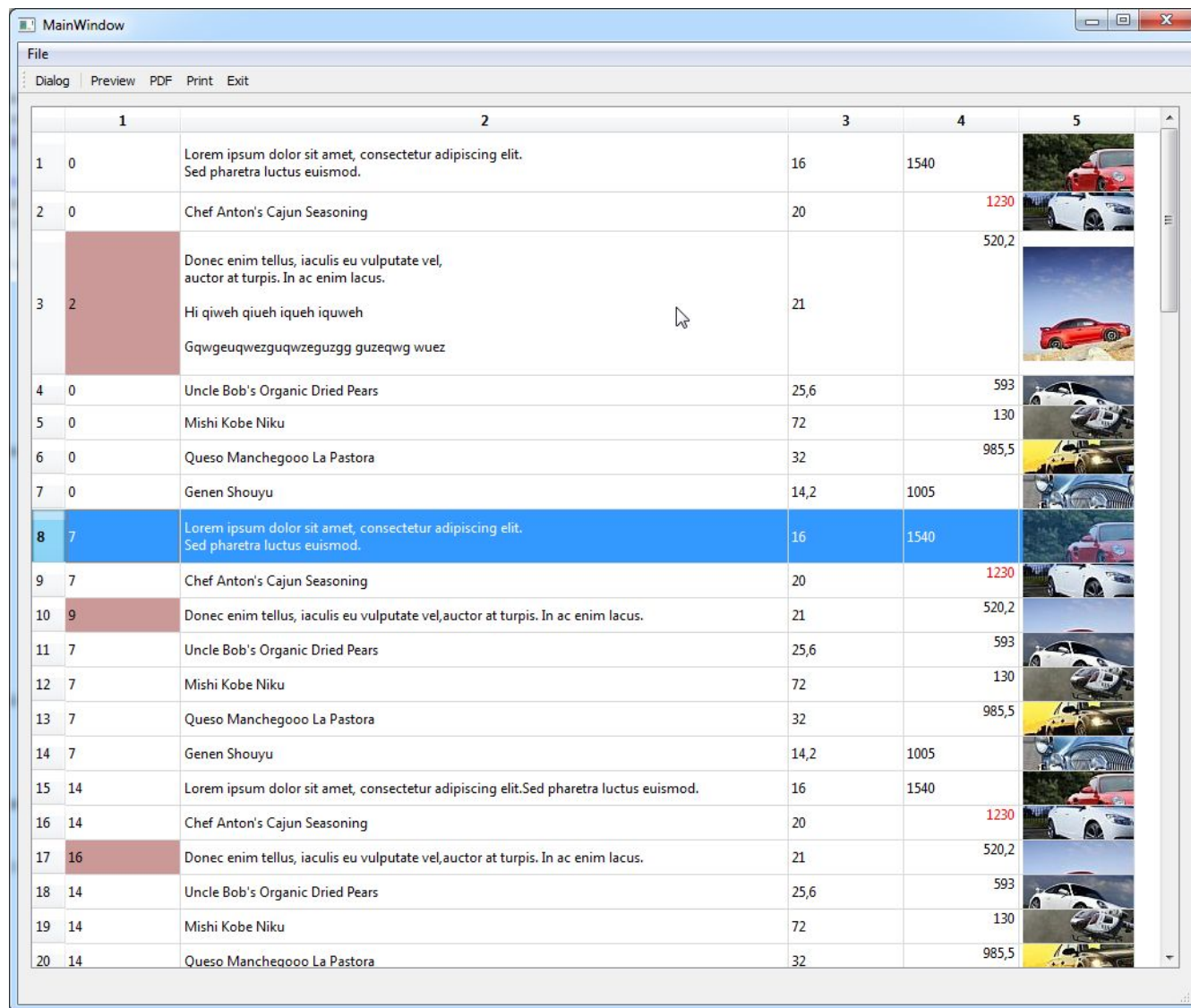
Use the following API functions for defining the QTableView object and its model for NCReport. You have to set the appropriate IDs to identify the objects. This because it is possible to assign multiple object pointers to NCReport. You don't need this in design time but only when running the report.









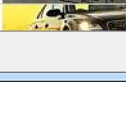







```
NCReport* report = new NCReport( this );
report->addTableView( ui->tableView, "myView");
report->addItemModel(ui->tableView->model(), "myModel");
```

## 19.4. Example

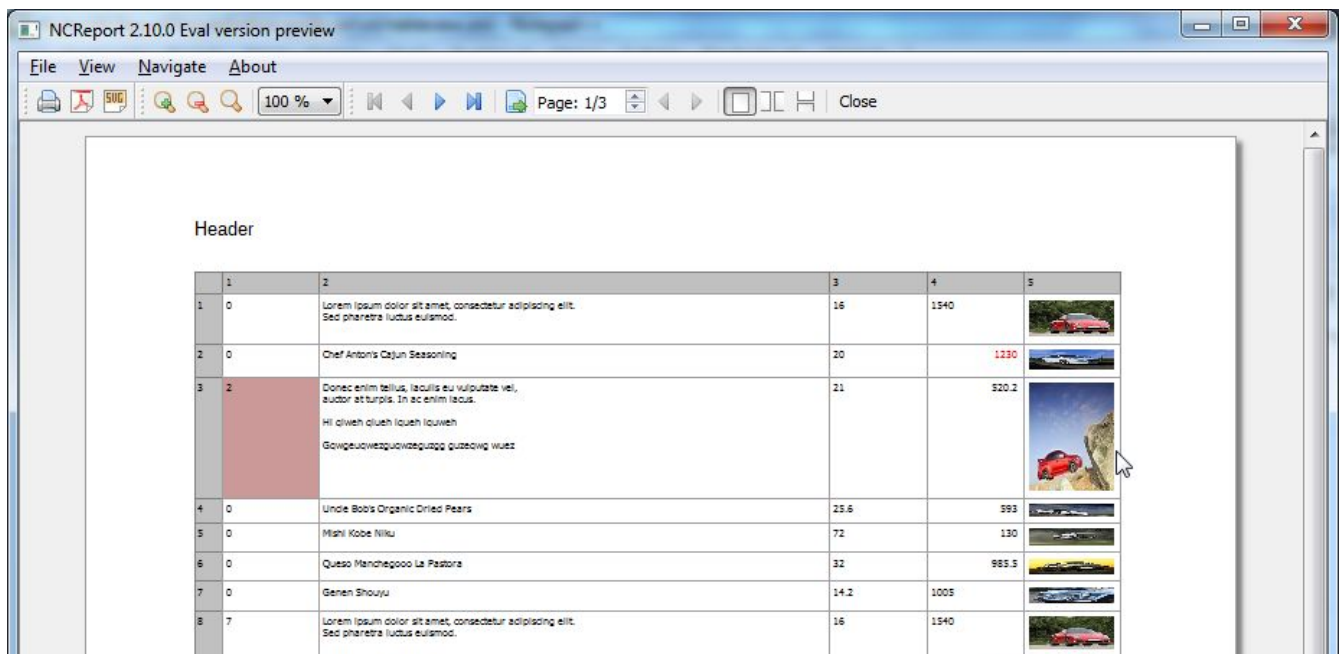
The following example shows how a printed QTableView widget looks like on the screen. The table is filled with test data and even images.

## 19.5. QTableView widget



	1	2	3	4	5
1	0	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed pharetra luctus euismod.	16	1540	
2	0	Chef Anton's Cajun Seasoning	20	1230	
3	2	Donec enim tellus, iaculis eu vulputate vel, auctor at turpis. In ac enim lacus. Hi qiweh qiueh iqueh iquweh Gqwgeuqwezguqwguzgg guzeqwg wuez	21	520,2	
4	0	Uncle Bob's Organic Dried Pears	25,6	593	
5	0	Mishi Kobe Niku	72	130	
6	0	Queso Manchego La Pastora	32	985,5	
7	0	Genen Shouyu	14,2	1005	
8	7	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed pharetra luctus euismod.	16	1540	
9	7	Chef Anton's Cajun Seasoning	20	1230	
10	9	Donec enim tellus, iaculis eu vulputate vel, auctor at turpis. In ac enim lacus.	21	520,2	
11	7	Uncle Bob's Organic Dried Pears	25,6	593	
12	7	Mishi Kobe Niku	72	130	
13	7	Queso Manchego La Pastora	32	985,5	
14	7	Genen Shouyu	14,2	1005	
15	14	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed pharetra luctus euismod.	16	1540	
16	14	Chef Anton's Cajun Seasoning	20	1230	
17	16	Donec enim tellus, iaculis eu vulputate vel, auctor at turpis. In ac enim lacus.	21	520,2	
18	14	Uncle Bob's Organic Dried Pears	25,6	593	
19	14	Mishi Kobe Niku	72	130	
20	14	Queso Manchego La Pastora	32	985,5	

## 19.6. QTableView table in print preview



```
void setCellGraphRenderer(const QString id, int column);
```

Example implementation of a custom cell / column renderer class

```
#include "ncreportgraphrenderer.h"

class MyCustomCellRenderer : public NCReportGraphRenderer
{
public:
    MyCustomCellRenderer();
    virtual ~MyCustomCellRenderer() override;

    virtual void paintItem( QPainter* painter, NCReportOutput* output, const QRectF&
rect, const QString& itemdata ) override;
};

#include "mycustomcellrenderer.h"
#include <QPainter>

MyCustomCellRenderer::MyCustomCellRenderer() {}

MyCustomCellRenderer::~~MyCustomCellRenderer() {}

void MyCustomCellRenderer::paintItem(QPainter *painter, NCReportOutput *output, const
QRectF &rect, const QString &itemdata)
{
    Q_UNUSED(output)
    Q_UNUSED(itemdata)

    MyLogoPainter cp;
    cp.setMyProperty(50);
    cp.setState(MyLogoPainter::Basic);

    painter->save();
    cp.paint(painter, rect.toRect());
    painter->restore();
}
```

To assign your renderer, use the **addItemRenderingClass** API function:

```
MyCustomCellRenderer *renderer = new MyCustomCellRenderer();
renderer->setId("mycell");
...
NCReport* report = new NCReport( this );
report->addItemRenderingClass(renderer);
```

In the table setting dialog you need set the ID for the table item in the report:



## 19.9. Handle progress signal of table rendering

It is possible to connect the progress signal of the table rendering into a slot in your application. This option can be useful in order to printing large tables. The signal is emitted row by row:

```
NCReport::dataRowProgress(int row)
```

# Chapter 20. Cross-Tab Tables

Reports often contain tables or data presented in a tabular layout. Sometimes it's necessary to rotate results so that columns are displayed horizontally and rows are displayed vertically. This is known as creating a PivotTable, a cross-tab report, or rotating data. In cross-tab tables, the data source records are represented as horizontal columns, and the cross-tab rows are printed as data source columns. Tables often include horizontal and/or vertical summarization as well.

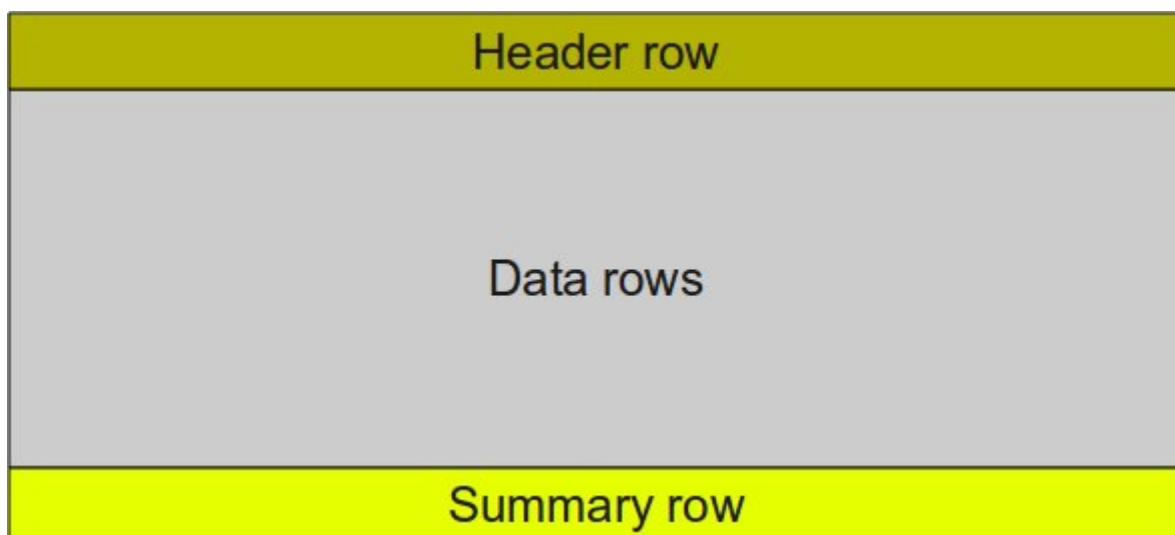
	Jan	Feb	Mar	Apr	May	June	Total
Income	19,80	23,30	35,70	43,90	28,70	30,50	181,90
Expense	20,10	19,80	18,50	18,60	19,60	21,20	117,80
Assets	10,00	8,00	16,00	17,00	17,30	14,10	82,40
Liability	47,30	36,60	54,10	31,80	42,90	53,20	265,90
Total	97,20	87,70	124,30	111,30	108,50	119,00	648,00

A cross-table has a unique data source assigned. In the report, a unique data source needs to be defined for the table. When the report generator renders cross tables, they behave as follows:

- Horizontally expandable: If the table is wider than the available space, it continues in a new table below. Table columns are represented as data source records.
- Vertically expandable: Each row represents a data column from the specified data source and can break into multiple pages.

## 20.1. Table Structure

Cross-tables are composed of cells, each with its own function depending on its location. The primary elements of tables are the rows and columns. The following figures show the cross-tab row and column structure with their named functions:



Header column	Data columns	Summary column
------------------	--------------	-------------------

*Table Rows*

0	1	1	1	2
3	4	4	4	5
3	4	4	4	5
3	4	4	4	5
3	4	4	4	5
3	4	4	4	5
6	7	7	7	8

*Table Columns*

#### *Cell Structure*

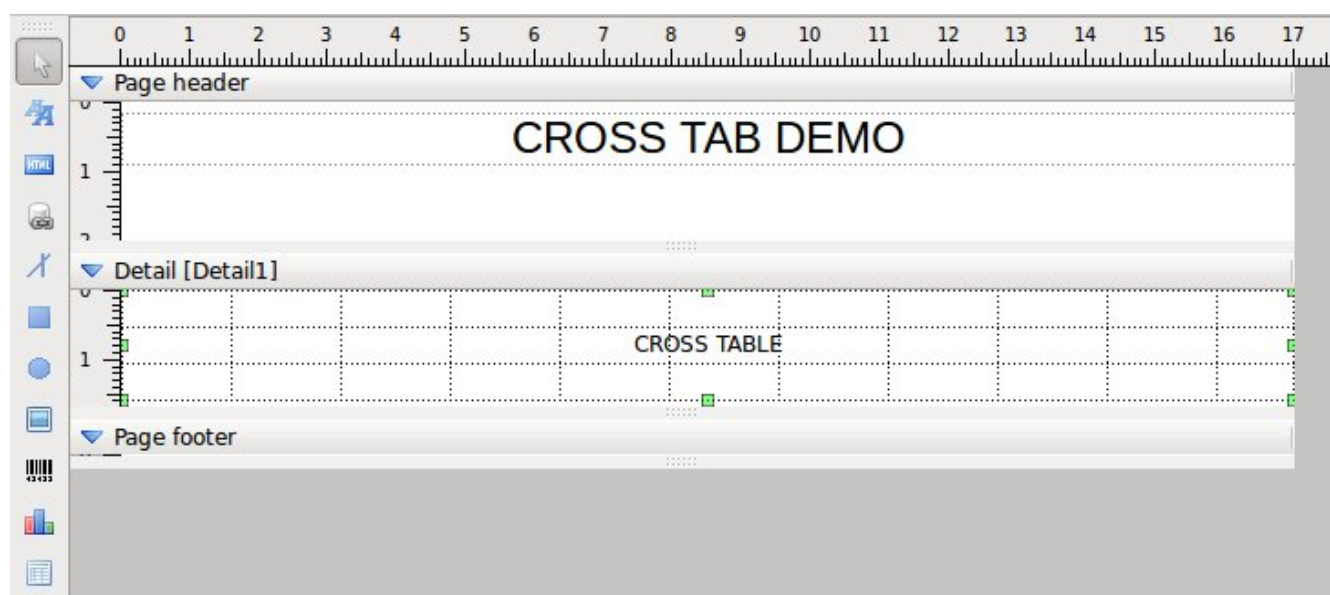
The cell structure of cross-tables is as follows:

Cell Function	Description
0	Corner header
1	Column header
2	Side summary header
3	Row header
4	Data
5	Side summary data
6	Bottom summary header
7	Bottom summary data
8	Cross summary data



## 20.2. Using Cross-Table in Designer

To add a cross-tab to a report, select the Cross table item from the toolbar or the Insert menu.



### *Cross-tab in Designer*

This creates a new Cross tab object in the selected section at the clicked position and opens the Cross table property dialog. In this dialog, you can set all the table's properties.

A screenshot of the 'Cross tab settings' dialog box. It has two tabs: 'Table properties' and 'Cell properties'. Under 'Table properties', there are fields for 'Data source ID' (DS\_TABLE), 'Hidden columns' (id), and 'Column title source' (order\_date). Below these are three columns of settings: 'Sizes and spaces' (Column widths: 24,00 mm, Row heights: 7,00 mm, Cell padding: 0,50 mm, Cell spacing: 1,00 mm, Table spacing: 7,00 mm), 'Section sizes' (Header column width: 28,00 mm, Data column width: Auto, Total column width: Auto, Header row height: Auto, Data row height: Auto, Total row height: Auto), and 'Show table parts' (checked: Column header, Row header, Bottom summary, Side summary; unchecked: Break table when page breaks). At the bottom, there is a field for 'Print only when expression is true:' and a 'Zone ID' field set to 0. 'Mégsem' and 'OK' buttons are at the bottom right.

### *Cross-tab settings dialog*



The property dialog is divided into the following tabs: Table properties and Cell properties. You can find the "Print only when expression" option at the bottom of the dialog. If a logical expression is defined, the table will be shown or hidden based on the result of the expression.

### 20.2.1. Table Level Properties

- Table data source: ID of the defined data source related to the table. The selected data source should be unique and independent from the data source of any detail because cross tables have their own data processing.
- Hidden columns: Comma-separated list of valid data source columns we don't want to show in the table.
- Column title source: Data column ID of column header titles. If not specified, the column numbers appear.
- Sizes and spaces: General sizes of cross-tab table elements.
  - Column widths: General width of columns
  - Row heights: General height of table rows
  - Cell padding: Gap size inside the cells, equal to internal cell margin
  - Cell spacing: Spacing size between the cells
  - Table spacing: Spacing between the tables when the cross-tab is multi-line.
- Section sizes: Sizes of cross-tab table sections.
  - Header column width: Width of the header (left/first) column
  - Data column width: Width of data columns
  - Total column width: Width of total/summary column, mostly the last, rightmost column
  - Header row height: Height of the header (first) row
  - Data row height: Height of the data rows
  - Total row height: Height of the total/bottom summary row, mostly the last row of the table.
- Show table parts: Switches to enable or disable specified table parts.
  - Column header: Show or hide column header
  - Row header: Show or hide row header
  - Bottom summary: Show or hide summary row
  - Side summary: Show or hide side summary column
  - Break table when page breaks: If enabled, the table can break within its rows when the page breaks.

### 20.2.2. Cell Level Properties

The cell properties are related to the specified cells, represented by their function.

Cross tab settings

Table properties | **Cell properties**

Corner header  
**Column header**  
Side summary header  
Row header  
Data  
Side summary data  
Bottom summary header  
Bottom summary data  
Cross summary data

**Foreground** | Background | Number | Borders

**Data**

Data type: Text

Static cell text:

Format arg string:

**Style**

Font: Liberation Sans

Font size: 8

☒ Bold ☐ Italic ☐ Underline

Text color: #000000

Rotation: 0,00°

**Align**

Horizontally: Right

Vertically: Center

**Print only when expression is true:**

Zone ID: 0

Mégsem OK

# Chapter 21. Conditional Formatting

This feature enables the use of dynamic, data-driven text styles in reports based on the current value of any data source columns, parameters, variables, or script expressions. Conditional formatting is available for Labels or Fields only.

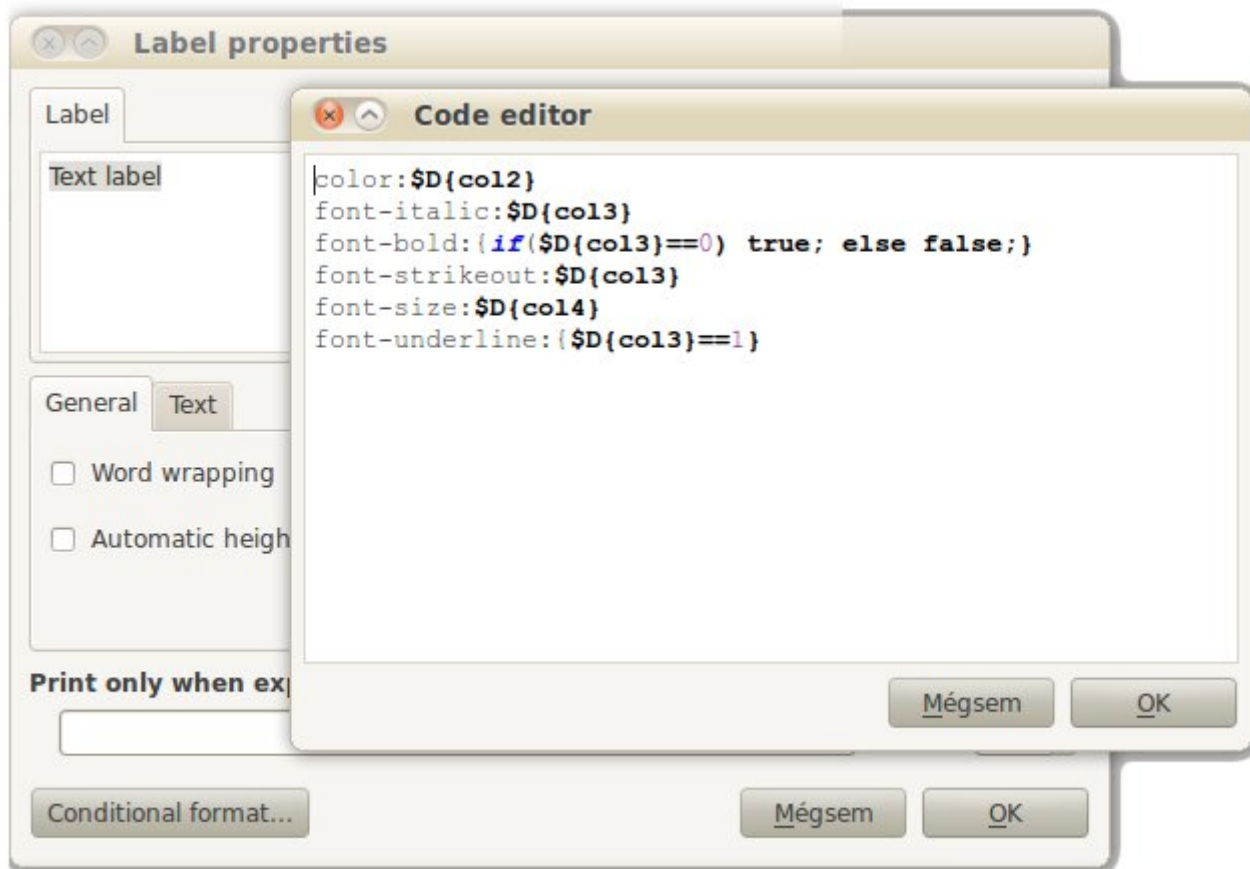
HTML texts can be dynamically formatted by embedding dynamic tags within HTML code. The format definition is a code text with style tag symbols and expressions similar to generic CSS style code. The style tag and its value/expression are divided by a colon. Each row represents one style definition. Script expressions must be enclosed in curly braces.

## 21.1. Dynamic Style Tag Symbols

Tag symbol	Description	Examples
<b>color:</b>	Text foreground color	<code>color:#ff0000,</code> <code>color:\${D{ds.color}},</code> <code>color:#{if(\${D{ds.price}&gt;500)}</code> <code>"#ff0000";}</code>
<b>background-color:</b>	Text background color	<code>background-color:#ff0000,</code> <code>background-</code> <code>color:\${D{ds.bgcolor}}</code>
<b>font-family:</b>	Font family name	<code>font-family:Arial, font-</code> <code>family::\${D{ds.font}}</code>
<b>font-bold:</b>	Font bold on/off	<code>font-bold:true, font-</code> <code>bold:\${D{ds.isBold}}</code>
<b>font-italic:</b>	Font italic on/off	<code>font-italic:true, font-</code> <code>italic:\${D{ds.isItalic}}</code>
<b>font-weight:</b>	Font weight integer value. Higher value results in bolder text.	<code>font-weight:50, font-</code> <code>weight:\${D{ds.fweight}}</code>
<b>font-underline:</b>	Font underline on/off	<code>font-underline:true, font-</code> <code>underline:\${D{ds.isUnderline}}</code>
<b>font-size:</b>	Font size in points. Integer value.	<code>font-size:12, font-</code> <code>size:\${D{ds.size}}</code>
<b>font-strikeout:</b>	Font strikeout on/off	<code>font-strikeout:true, font-</code> <code>strikeout:\${D{ds.fstrikeout}}</code>
<b>letter-spacing:</b>	Text letter spacing value. Greater value results in bigger spacing	<code>letter-spacing:1.5, letter-</code> <code>spacing:\${D{ds.letterspacing}}</code>
<b>capitalization:</b>	Rendering option for text font applies to. Integer value from 0-4. Equals QFont::Capitalization enumeration property	<code>capitalization:\${D{ds.cap}}</code>

## 21.2. Editing Style Code in Designer

To define conditional text formatting for a Label or a Field, click on the "Conditional formatting..." button at the bottom of the item property dialog. Then the conditional format code dialog will appear. Type or paste the format code while adhering to the syntax rules. Click [OK] to save the code.



### NOTE

Style tag and its corresponding value should be in one line! Multiple lines of style definitions are not evaluated.

## 21.3. Default Style

If a condition (script or data) returns an empty value, the default style formatting option is applied. The default style settings are what you set statically in the report as usual.

# Chapter 22. General TEXT output

Text output is a very powerful feature in NCReport. The function provides the ability of generating various kind of text outputs like HTML, XML, Plain text, etc. Text Output requires an additional template to be existed. Before running a report you have to specify the text template file as well.



TEXT output is generated very fast, because data is processed and substituted directly into the text template without any graphical rendering.

## 22.1. Text template manager tags

The following manager keywords/tags are available when you create a text template. Each start and end tags represents a specified section. Tags are enclosed in standard HTML comment tokens, according to HTML

## 22.2. Text template tags

Tag keyword	Description
<code>&lt;!-- BEGIN {DH} --&gt;</code>	Document header begins. Document means the current text output. For example the HTML header part.
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {DF} --&gt;</code>	Document footer begins. For example the HTML document footer part.
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {PH} --&gt;</code>	Page header section begins.
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {PF} --&gt;</code>	Page footer section begins.
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {RF} --&gt;</code>	Report header section begins.
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {RF} --&gt;</code>	Report footer section begins.
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {D.DetailID} --&gt;</code>	Detail section begins. Section is identified by DetailID
<code>&lt;!-- END --&gt;</code>	Section ends
<code>&lt;!-- BEGIN {GH.DetailID.GroupID} --&gt;</code>	Group header section begins. Section is identified by both DetailID and GroupID
<code>&lt;!-- END --&gt;</code>	Section ends

<code>&lt;!-- BEGIN {GF.DetailID.GroupID} --&gt;</code>	Group footer section begins. Section is identified by both DetailID and GroupID
<code>&lt;!-- END --&gt;</code>	Section ends

## 22.3. Examples

The following example shows how a typical usage of text template

```

<!-- BEGIN {DH} -->
SIMPLE TEXT REPORT OUTPUT
<!-- END -->
<!-- BEGIN {PH} -->
Customer ID      Name      Address
-----
<!-- END -->
<!-- BEGIN {D.Detail1} -->
${D{custid}}     ${D{custname}}  ${D{address}}
<!-- END -->

```

# Chapter 23. Batch Report Mode

Batch report mode is a feature that enables running multiple reports into one output. Read more at [Part Using NCRReport API \[batch\]](#)

# Chapter 24. Special Detail Sections

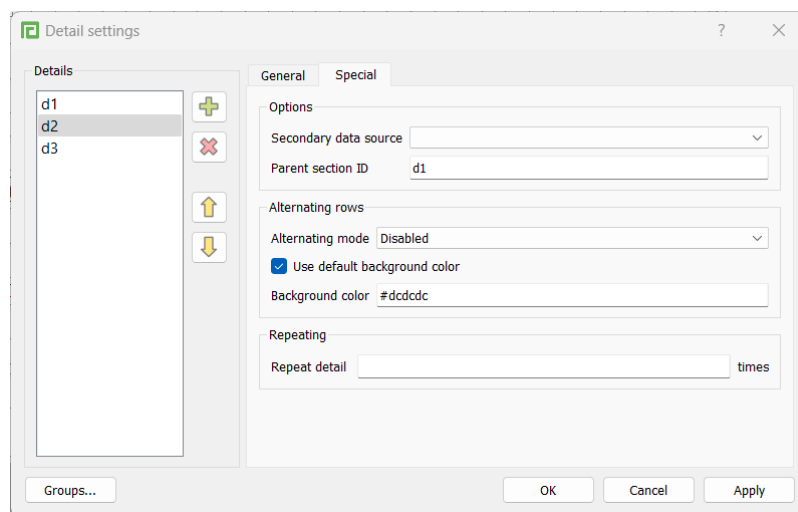
There are special features can be done with detail sections that might be is very useful sometimes. It is possible to render some special details at non normal location, these functions are subdetail, repeated detail or odd even content.

## 24.1. Sub (Detail) Sections

Sometimes a single report section (detail) is not enough to handle all contents, it would be good to have one or more additional sections for adding more items. If we don't use zones a section is the least band area in which we can keep a group of items together on a page. Sub section is a possibility of separating a simple detail section to an additional content area that is linked with its parent section. The parent can be a detail or a group header (or anything else) The sub section is always rendered below its parent. Using the **printWhen** option of the section we conditionally hide / show the section so this feature can make the report design very flexible.

## 24.2. Adding a Sub-Section

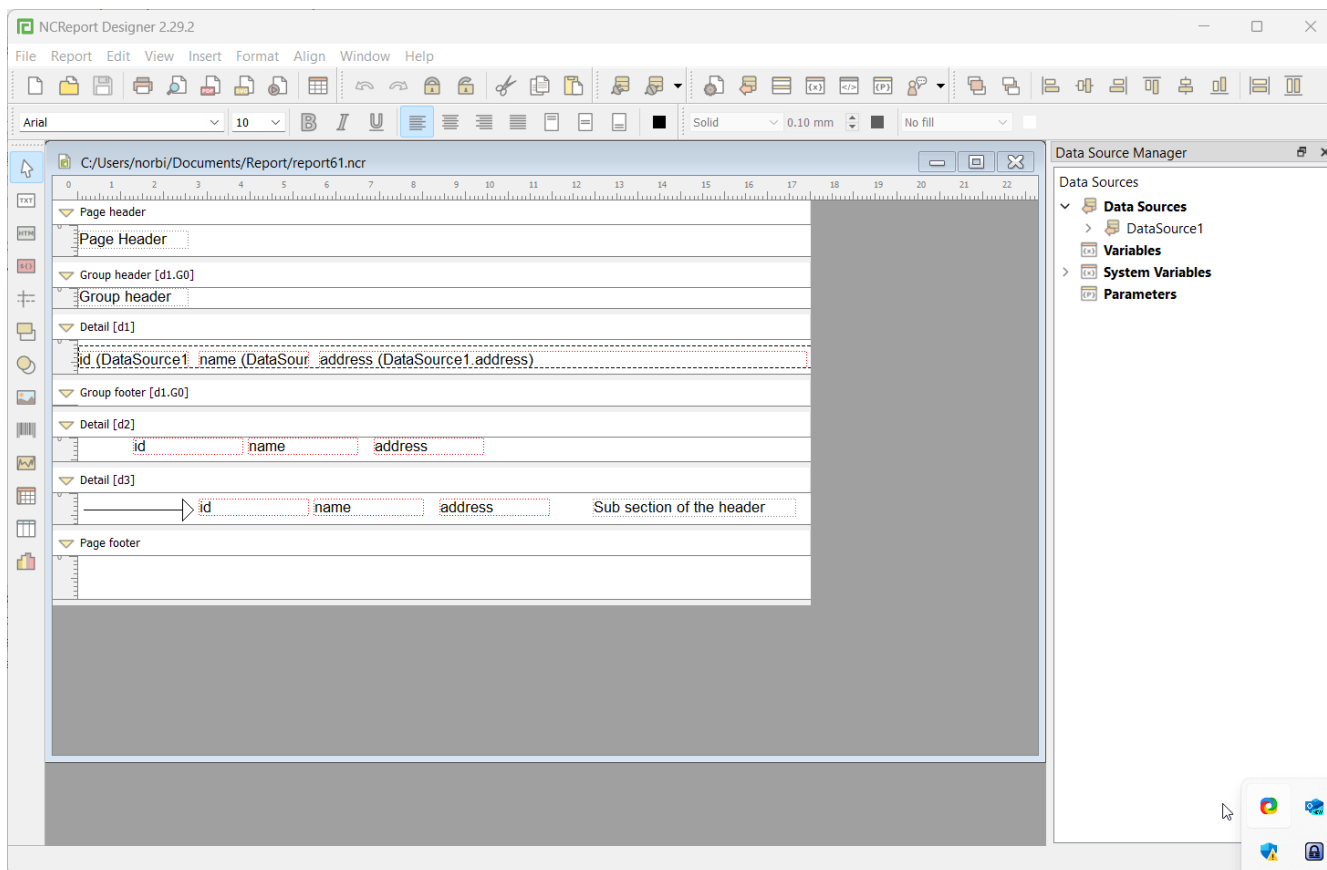
To add a sub section just open the menu::[Details & groups] and add a new Detail section. By default a detail section needs to have an assigned data source, but a sub section doesn't, you must set the **Parent Section ID** on the **Special** tab. The ID is the string identifier of the section you want to link with.



The sub section will appear just after its assigned parent regardless of the other details.

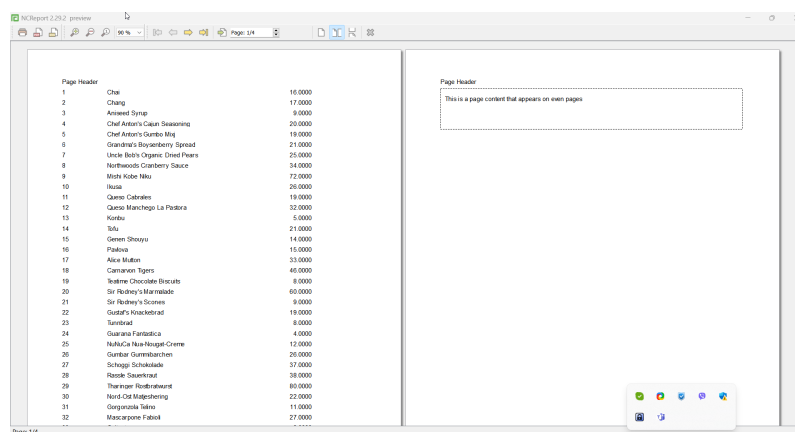
## 24.3. Example Sub-Sections





## 24.4. Odd / Even Pages

Odd / even page feature is a special use case of the sub-section. It is possible to create a report that have odd and even page after each other, especially another page content gets inserted at every n or n+1 page. You can specify a dedicated detail section for representing an odd page or an even page for this purpose.



To create odd or even page, add a new detail to your report and set one of the following identifiers at **Detail settings** → **Special** tab **Parent section ID**. It's recommended to assign a dummy / one row data source to such details, to ensure that their content gets printed only once.

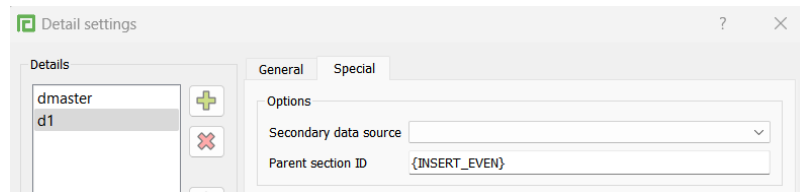
{INSERT\_EVEN} or {INSERT\_ODD}

**{INSERT\_EVEN}**

Represents a content that gets inserted at every even page

**{INSERT\_ODD}**

Represents a content that gets inserted at every odd page



## 24.5. Repeated detail by constant or dynamic value

By default every detail section gets printed only one time per data source row but sometimes we need to repeat the detail content more times in special circumstances for instance if we design a report aimed to be such like ticket, barcode or label. To set the number of times the detail should be repeated use the **Detail settings** → **Special** tab **Repeat detail** field. It's possible to set an explicit constant integer value or an integer data expression here. The feature works for normal detail sections only.

Example: 5 or  $\$P\{\text{numberofdetails}\}$ , or  $\$D\{\text{num}\}$  ...

A screenshot of a report preview. It shows a table with three columns: a sequence number, a description, and a value. The data is grouped into sections: 'Chai' (5 rows), 'Chang' (5 rows), and 'Aniseed Syrup' (5 rows). Each section is repeated multiple times, as indicated by the 'Repeat detail' setting in the previous image.

Page Header		
1	Chai	16.0000
1	Chai	16.0000
1	Chai	16.0000
1	Chai	16.0000
1	Chai	16.0000
2	Chang	17.0000
2	Chang	17.0000
2	Chang	17.0000
2	Chang	17.0000
2	Chang	17.0000
3	Aniseed Syrup	9.0000
3	Aniseed Syrup	9.0000
3	Aniseed Syrup	9.0000
3	Aniseed Syrup	9.0000
3	Aniseed Syrup	9.0000

# Command Line Tool

This part specifies the report engine command line tool

# Chapter 25. Command line client

NCReport engine is also available in command line client executable.

## 25.1. To run command line executable

Running the command line engine use the [ **NCReport** ] command in the installed /bin directory:

```
NCReport [options]
```

## 25.2. Command line options

### **-?, --help**

Display this help

### **-v, --version**

NCReport version

### **-f, --report-file [filename]**

Name of the report definition XML file. If this is set, the report definition will be parsed from this file instead of a database

### **q, --sql-driver [driver]**

Qt sql driver name for database connection. Available drivers: QDB2, QIBASE, QMYSQL, QOCI, QODBC, QPSQL, QSQLITE, QTDS

### **-h, --host [hostname]**

Database host name or IP address (default is localhost)

### **-u, --user [username]**

Database user login name

### **-p, --password [password]**

Database user login password

### **-d, --database [dbname]**

Database name to use

### **-pt --port [port]**

Database port number

### **-c, --connection-id [id]**

Sql connection name/id

### **-cs, --connect-string [string]**

Joined connection string in [user]/[pass]@[host/sid] format

**-co, --connect-option [opt]**

Connect option string/id

**-o, --output [output type]**

Output types: print, preview, pdf, svg, html, image. Use **[print]** for send report to printer or **[preview]** to show report in a preview window. The default output is preview

**-of, --output-file [filename]**

Output file name. Required for file based output types.

**-n, --printer [printername]**

Name of the target printer

**--copies [1..50]**

Number of copies in case output is printer

**--force-copy**

Use forced copy printing method. This is useful for documents in which has to be known the number of current copy.

**--nodialog**

Runs report to default printer without showing printer dialog.

**-dbid, --report-db-id [id]**

ID number of the report definition (xml) text in a database record. If this is set the report definition will be parsed from database/table/record instead of a file.

**-par, --add-parameter [parametername],[value]**

Adds a custom parameter to report. You must specify the value and name of the parameter separated by comma. The `$P{parametername}` expression can be used in the report definition (Example: firstname,Robert)

**-l, --language [filename]**

International 2 letters language code of the current report language Works only when the specified language is defined in a multi lingual report.

**-td, --template-dir [dir]**

Sets default template directory when using additional files, such as charts. Works only when the specified language is defined in a multi lingual report.

**-hs, --htmlstrategy**

HTML output generation strategy. 1 = section is translated as one table (default) 2 = sections are always unique tables

**-css, --css-file**

HTML output style sheet file to be generated. If not set, the stylesheet is included in the target HTML file.

# Using NCRReport API

This part shows how to implement NCRReport API code in your application

# Chapter 26. Using NCRreport API

This chapter shows you how to use the report API from a C++ (Qt) application, how to create an NCRreport object and how to use it from your application. As we described earlier NCRreport system consists of two parts: Report renderer library and a report designer GUI application. Of course the report engine can be used separately from Designer.

If you want to call NCRreport from your application, first you have to integrate NCRreport in your project. There are several ways to do so:

- To add the whole sources to your project and build it together with your application.
- To use NCRreport engine as shared library. For using NCRreport library like other libraries in your project you need to specify them in your project file. For more information see the Qt documentation in **qmake** manual at chapter Declaring Other Libraries.
- Statically linking NCRreport library for your project. For more information see the Qt documentation in **qmake** manual at chapter Declaring Other Libraries

## 26.1. Project file settings

You have to add to your .pro file at least the following lines:

```
NCREPORT_LIBPATH = /home/ncreport/lib

CONFIG(release, debug|release) {
    win32|win64: LIBS += -L$$NCREPORT_LIBPATH -lNCRreport2
    unix:!macx:  LIBS += -L$$NCREPORT_LIBPATH -lNCRreport
    else:macx:   LIBS += -framework NCRreport
}
CONFIG(debug, debug|release) {
    win32|win64: LIBS += -L$$NCREPORT_LIBPATH -lNCRreportDebug2
    unix:!macx:  LIBS += -L$$NCREPORT_LIBPATH -lNCRreportDebug
    else:macx:   LIBS += -framework NCRreportDebug
}
INCLUDEPATH += /home/ncreport/include
...
```

## 26.2. Initialize NCRreport class

This section covers the fundamental steps that most users should take when creating and using NCRreport class. We present each of the activities in the suggested order.

## 26.3. Include directives

To include the definitions of the module's classes, use the following directive:

```
#include "ncreport.h"
#include "ncreportoutput.h"
#include "ncreportpreviewoutput.h"
#include "ncreportpreviewwindow.h"
```

## 26.4. Creating NCRReport class

Create the report class just like as another QObject based class:

```
NCRReport *report = new NCRReport();
```

If the class has created earlier and passed as a parameter to your method in which you use the report object you should initialize the report by calling `reset()` method. Otherwise, if creating a new object in the scope, you don't need to call the reset.

```
report->reset();
```

## 26.5. Connecting to SQL database

SQL connection is required only when your data source uses internal database connection. In other words Internal connection means an already existing database connection which is established before running the report. On the other hand reports can also use external (defined in the report / built-in) connection as well. Other data sources don't require database connection.

This example code shows a typical SQL database connection with error handling:

```
QSqlDatabase defaultDB = QSqlDatabase::addDatabase("QMYSQL", "myconn" );

if ( !defaultDB.isValid() ) {
    QMessageBox::warning( 0, "Report error", QObject::tr("Could not load database driver.") );
    delete report;
    return;
}
defaultDB.setHostName( "host" );
defaultDB.setDatabaseName( "database" );
defaultDB.setUserName( "user" );
defaultDB.setPassword( "password" );

if ( !defaultDB.open() ) {
    QMessageBox::warning( 0, "Report error", QObject::tr("Cannot open database:")
+defaultDB.lastError().databaseText() );
    return;
}
```



## 26.6. Setting the Report's source

Report source means the way of NCReport handles XML report definitions, in other words the source of report definition XML data. Report definitions may be opened from a file - in most cases it is suitable, but it can be loaded also from an SQL database's table. For information of configuring and using the different report sources see the parts 1-7

In current example we apply File as report source:

```
report->setReportFile( fileName );
```

This code is equivalent with this code:

```
report->setReportSource( NCReportSource::File );  
report->reportSource()->setFileName( fileName );
```

## 26.7. Adding parameters

If your report uses parameters you have to add parameter object(s) to NCReport object before running the report. To create and add a parameter do this:

```
report->addParameter( "id", "value" );
```

where ID is a QString, the value is a QVariant object.

## 26.8. Running the Report

Now we are ready to run the Report and catch the error message if an error occurs. There are at least two ways to start running the report engine.

## 26.9. Running the Report by One Step

This running mode is the most simple but with less custom configuration is available.

```
// run report to printer  
bool result = report->runReportToPrinter(1, true, parent);  
// run report to pdf file  
bool result = report->runReportToPDF( "file.pdf" );  
// or  
bool result = report->runReportToPdfWriter( "file.pdf" ); //recommended way  
// run report to svg files  
bool result = report->runReportToSVG( "file.svg" );  
// run report to preview output  
bool result = report->runReportToPreview();
```

```
// run report to QPrintPreview dialog
bool result = report->runReportToQtPreview();
```

This way, if we want to preview the report we also have to create and show `NCReportPreviewWindow`. See the next section.

## 26.10. Running the Report in customized mode

This running report mode allows more flexible configuration. First we have to initialize the output object, after the report is ready to run.

## 26.11. Initializing Report's Output

The next issue is to create and specify the report's output. As rendering target, `NCReport` applies a class derived from `NCReportOutput` base class. There are pre-defined classes for the mostly used outputs:

- `NCReportPrinterOutput`
- `NCReportPreviewOutput`
- `NCReportPdfOutput`

To define the specified output use a code similar to this:

```
NCReportOutput *output=0;

if ( rbPreview->isChecked() ) {
    output = new NCReportPreviewOutput();
    output->setAutoDelete( FALSE );
    report->setOutput( output );
} else if ( rbPrinter->isChecked() ) {
    output = new NCReportPrinterOutput();
    output->setCopies(1);
    output->setShowPrintDialog(TRUE);
    report->setOutput( output );
} else if ( rbPdf->isChecked() ) {
    QString fileName = QFileDialog::getSaveFileName(this, tr("Save PDF File"),
"report.pdf", tr("Pdf files (*.pdf)"));
    if ( fileName.isEmpty() ) {
        delete report;
        return;
    } else {
        output = new NCReportPdfOutput();
        output->setFileName( fileName );
        report->setOutput( output );
    }
}
```

## 26.12. Running the Report

Now we are ready to run the Report and catch the error message if an error occurs:

```
QApplication::setOverrideCursor(QCursor(Qt::WaitCursor));
report->runReport();
bool error = report->hasError();
QString err = report->lastErrorMsg();
QApplication::restoreOverrideCursor();
```

## 26.13. Previewing Report

If we specified `NCReportPreviewOutput` as report's output, it does not run the preview form automatically. After the report engine successfully done we need to initialize an `NCReportPreviewWindow*` object for previewing. The following code shows the way of doing this. It is suggested to catch the error first, before running preview dialog.:

```
if ( error ) {}
    QMessageBox::information( 0, "Report error", err );
} else {
    if ( rbPreview->isChecked() ) {
        //-----
        // PRINT PREVIEW
        //-----
        NCReportPreviewWindow *pv = new NCReportPreviewWindow();
        pv->setReport( report );
        pv->setOutput( (NCReportPreviewOutput*)output );
        pv->setWindowModality(Qt::ApplicationModal );
        pv->setAttribute( Qt::WA_DeleteOnClose );
        pv->exec();
    }
}
```



We must not delete the output object after we added to the `NCReportPreviewWindow` object. The preview window will delete its output object when destroys.



For the best performance and code quality we should not delete `NCReport` object until we close preview dialog. Add the report object to the preview object by

```
report->setReport( NCReport* report );
```

If it's done the printing from preview will result the original printout quality, since it will run report again instead of printing the lower quality preview pages.



Since 2.8.4 version it's possible to show the preview widget in dialog mode, just like QDialog. **NCReportPreviewMainWindow::exec()** function shows the preview window and keeps application event loop while preview. This is good when you use a locally defined report object, because the report object will not be deleted until user closes the preview window.

## 26.14. Deleting Report object

After report running action you may want to delete the report object. Note, that preview window requires NCReport object to be existed. If NCReportPreviewWindow::exec() is used, then the application event loop stops until the preview window gets closed.

```
delete report;
```

## 26.15. Using other data sources

NCReport allows you to use non SQL data sources of the likes of QString based text, QStringList, or custom defined data source. Depending on the data source type, data may come from a File, NCReportParameter or by another way.

### 26.15.1. QString based text data source

NCReport allows to use QString texts as simplest data source. Each text row represents one data record (rows are separated by the linefeed character) and the data columns are separated by a specified delimiter character. For column identification in fields use the number of the column as reference in the report by the following:

0 for 1st column, 1 for 2nd column ... etc. or col0 for 1st column, col1 for 2nd column ... etc.

For using text data sources add a Text data source to your report in Designer. You have the following ways:

- Storing a static text in report definition. For doing so, in Designer select Static location type and add static text to the edit box by using the specified delimiter character.
- Using an existed text file. For doing so, in Designer select File location type and specify the name of the text file you want to use. Also you must specify the delimiter character.
- Adding text to NCReport by NCReportParameter. For doing so, first add the data text as a parameter to NCReport by **addParameter()** function. Select Parameter location type in Designer and specify the ID of the parameter you have added. Also you must specify the delimiter character.

Example of adding a QString text to NCReport as parameter and Tab character as column delimiter:

```
NCReport report;
```

```

QString data;

data += "1 \tChai \t16.0000\t1\t1540\t0\n";
data += "2 \tChang \t17.0000\t1\t 874\t0\n";
data += "3 \tAniseed Syrup \t 9.0000\t1\t1687\t0\n";
data += "4 \tChef Anton's Cajun Seasoning \t20.0000\t1\t1230\t0\n";
data += "5 \tChef Anton's Gumbo Mixj \t19.0000\t2\t1900\t0\n";
data += "6 \tGrandma's Boysenberry Spread \t21.0000\t2\t 520\t0\n";
data += "7 \tUncle Bob's Organic Dried Pears \t25.0000\t3\t 540\t0\n";
data += "8 \tNorthwoods Cranberry Sauce \t34.0000\t3\t 120\t0\n";
data += "9 \tMishi Kobe Niku \t72.0000\t3\t 130\t0\n";
data += "10 \tIkura \t26.0000\t3\t2247\t0\n";
data += "11 \tQueso Cabrales \t19.0000\t4\t 741\t0\n";
data += "12 \tQueso Manchego La Pastora \t32.0000\t4\t 512\t0\n";
data += "13 \tKonbu \t 5.0000\t4\t1470\t0\n";
data += "14 \tTofu \t21.0000\t4\t 978\t0\n";
data += "15 \tGenen Shouyu \t14.0000\t4\t1005\t0\n";

report.addParameter( "data1", data );

```

## 26.15.2. QStringList data source

NCRReport allows you to use also QStringList as a data source. First you should define a QStringList. Each QStringList item represents one data record and the data columns are separated by a specified delimiter character.

For column identification in fields use the number of the column as reference in the report by the following:

0 for 1st column, 1 for 2nd column ... etc. or col0 for 1st column, col1 for 2nd column ... etc.

For using QStringList data sources add a **StringList** data source to your report in Designer. You can only use **Static** location type since QStringList can be added to NCRReport by one mode only: using **addStringList()** function. You have to specify an id with the list for identifying purposes.

Example of using QStringList as data source:

```

NCRReport report;

QStringList list;
list << "24|Renate Moulding|Desert Hot Springs,CA|1|2008-01-01";
list << "78|Alfred Muller|Miami Beach, FL|1|2008-01-03";
list << "140|Angela Merkel|Munchen, Germany|1|2008-01-07";
list << "139|Bob Larson|Dallas, TX|0|2008-01-20";

report.addStringList( list, "s10" );

```

### 26.15.3. Item Model data source

Item/Model/View architecture is a very useful new feature of Qt4. NCReport allows you to use a data source based on QAbstractItemModel. First you have to create your item model. Each model row represents one data record. For column identification in fields use the number of the column as reference in the report by the following:

0 for 1st column, 1 for 2nd column ... etc. or col0 for 1st column, col1 for 2nd column ... etc.

For using Item Model data sources add an Item Model data source to your report in Designer. You can only use Static location type, other locations are undefined. In your code add the Item model to NCReport using `addItemModel(...)` function. You have to specify an id to the model for identifying purposes. The same ID you must specify for item model data source in the designer.

Example of using Item Model as data source:

```
QStandardItemModel *model = new QStandardItemModel( 2, 4 );
QStandardItem *item =0;

// -----
item = new QStandardItem();
item->setData( 1, Qt::EditRole );
model->setItem( 0, 0, item);

item = new QStandardItem();
item->setData( "Chai", Qt::EditRole );
model->setItem( 0, 1, item);

item = new QStandardItem();
item->setData( 16.0, Qt::EditRole );
model->setItem( 0, 2, item);

item = new QStandardItem();
item->setData( 1540.0, Qt::EditRole );
model->setItem( 0, 3, item);

// -----
item = new QStandardItem();
item->setData( 2, Qt::EditRole );
model->setItem( 1, 0, item);

item = new QStandardItem();
item->setData( "Chef Anton's Cajun Seasoning", Qt::EditRole );
model->setItem( 1, 1, item);

item = new QStandardItem();
item->setData( 20.0, Qt::EditRole );
model->setItem( 1, 2, item);

item = new QStandardItem();
```

```

item->setData( 1230.0, Qt::EditRole );
model->setItem( 1, 3, item);

report.addItemModel( model, "model1" );

```

## 26.16. Custom data sources

NCRReport allows you to create your own data source by subclassing `NCRReportData` source abstract class. By this way you can use anything as data for NCRReport. You only have to implement the the required class methods. For adding your class to NCRReport use `addCustomDataSource()` function. The following example demonstrates the way of defining and using custom data source class:

### 26.16.1. Declaration

```

#include "ncreportdatasource.h"
#include <QDate>

struct TestData {
    int id;
    QString name;
    QString address;
    bool valid;
    QDate date;
};

class TestDataSource : public NCRReportDataSource
{
    Q_OBJECT
public:
    TestDataSource( QObject *parent=0 );
    TestDataSource() {}

    void addData( const TestData& );

    bool open();
    bool close();
    bool first();
    bool last();
    bool next();
    bool previous();
    int size() const;
    QVariant value( const QString& ) const;
    QVariant value( int ) const;
    bool read( NCRReportXMLReader* );
    bool write( NCRReportXMLWriter* );
private:
    QList<TestData> list;

```

```
};
```

## 26.16.2. Implementation

```
TestDataSource::TestDataSource(QObject * parent) : NCReportDataSource( parent )
{
    data sourcetype = Custom;
    location = Static;
    recno =0;
}

bool TestDataSource::open()
{
    if ( list.isEmpty() ) {
        error->setError( tr("No data in TestDataSource data source") );
        return false;
    }
    recno =0;
    m_opened = true;
    return true;
}

bool TestDataSource::close()
{
    recno =0;
    m_opened = false;
    return true;
}

bool TestDataSource::next()
{
    recno++;

    if ( recno >= list.count() ) {
        recno--;
        flagEnd = true;
        return false;
    }

    flagBegin = false;
    return true;
}

int TestDataSource::size() const
{
    return list.count();
}
```



```

bool TestDataSource::prevoius()
{
    recno--;

    if ( recno < 0 ) {
        recno = 0;
        flagBegin = true;
    }
    return true;
}

bool TestDataSource::first()
{
    recno=0;
    return true;
}

bool TestDataSource::last()
{
    recno = list.count()-1;
    return true;
}

QVariant TestDataSource::value(const QString & column ) const
{
    if ( column == "id" )
        return value( 0 );
    if ( column == "name" )
        return value( 1 );
    if ( column == "address" )
        return value( 2 );
    if ( column == "valid" )
        return value( 3 );
    if ( column == "date" )
        return value( 4 );
    else
        return QVariant();
}

QVariant TestDataSource::value( int column ) const
{
    QVariant v;
    switch (column) {
        case 0: v = list.at(recno).id; break;
        case 1: v = list.at(recno).name; break;
        case 2: v = list.at(recno).address; break;
        case 3: v = list.at(recno).valid; break;
        case 4: v = list.at(recno).date; break;
    }
    return v;
}

```

```

bool TestDataSource::read(NCReportXMLReader *)
{
    return true;
}

bool TestDataSource::write(NCReportXMLWriter *)
{
    return true;
}

void TestDataSource::addData(const TestData & data)
{
    list.append( data );
}

```

### 26.16.3. Using the TestDataSource class

Now we have to take our TestDataSource class. For using the TestDataSource data source add a Custom data source to your report in Designer. You can only use **Static** location type for this data source. Specify the class ID id with the list for identifying purposes both in Designer and in the class by **setID()** function.

```

NCReport report;

TestDataSource *ds = new TestDataSource();
ds->setID("cds0");

TestData d1;
d1.id = 123;
d1.name = "Alexander Henry";
d1.address = "HOT SPRINGS VILLAGE, AR";
d1.valid = true;
d1.date = QDate(2008,01,10);
ds->addData( d1 );

TestData d2;
d2.id = 157;
d2.name = "Julius Coleman";
d2.address = "Coronado, CA";
d2.valid = false;
d2.date = QDate(2008,01,12);
ds->addData( d2 );

TestData d3;
d3.id = 157;
d3.name = "Peter Moulding";
d3.address = "San francisco, CA";
d3.valid = true;

```

```
d3.date = QDate(2008,01,07);
ds->addData( d3 );

report.addCustomDataSource( ds );
```

## 26.17. Custom items in NCRReport

Custom items are powerful members of the report system. Custom item feature enables you to render special, custom defined contents in reports. The typical field of application is using this feature for rendering graphs or such kind of contents.

- Add a Graph (Custom) item into your report in the designer and specify the size and the location of the object. Specify the class ID of the item. This ID is used for identification.
- Subclass **NCRReportAbstractItemRendering** class and implement its **paintItem** method.
- Set the ID of your class for identification by **setID()** function.
- Add your item class to NCRReport by using **addItemRenderingClass()** function.

Let's take an example for custom item class:

### 26.17.1. Declaration

```
#include "ncreportabstractitemrendering.h"

class TestItemRendering : public NCRReportAbstractItemRendering
{
public:
    TestItemRendering();
    ~TestItemRendering();

    void paintItem( QPainter* painter, NCRReportOutput* output, const QRectF& rect,
const QString& itemdata );
};
```

### 26.17.2. Implementation

```
#include "testitemrendering.h"
#include "ncreportoutput.h"

#include <QPainter>
#include <QColor>

TestItemRendering::TestItemRendering()
{
}

TestItemRendering::~TestItemRendering()
```

```

{
}

void TestItemRendering::paintItem(QPainter * painter, NCRReportOutput* output, const
QRectF & rect, const QString & itemdata)
{
    switch ( output->output() ) {
    case NCRReportOutput::Printer:
    case NCRReportOutput::Pdf:
    case NCRReportOutput::Preview:
        break;
    default:
        return;
    }

    const int numcols = 10;
    const int cw = qRound(rect.width()/numcols);
    painter->setPen( Qt::NoPen );
    int ch=0;
    QColor color;
    color.setAlpha( 128 );

    for ( int i=0; i < numcols; ++i ) {
        if ( i%3 == 0 ) {
            color.setRgb(0xAAAAFF);
            ch = qRound(rect.height()*0.8);
        } else if ( i%2 == 0 ) {
            color.setRgb(0xAAFFAA);
            ch = qRound(rect.height()*0.4);
        } else {
            color.setRgb(0xFFAAAA);
            ch = qRound(rect.height()*0.6);
        }
        painter->setBrush( QBrush(color) );
        painter->drawRect( rect.x()+i*cw, rect.y()+qRound(rect.height())-ch , cw, ch
    );
    }

    painter->setPen( QPen(Qt::black) );
    painter->setBrush( Qt::NoBrush );

    painter->drawRect( rect );

    painter->setFont( QFont("Arial",8) );
    painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter | Qt::TextWordWrap,
    QString("GRAPH EXAMPLE: %1").arg(itemdata) );
}

```

### 26.17.3. Using the class

```
NCReport report;

TestItemRendering *irc = new TestItemRendering();
irc->setID("testitem0");
report.addItemRenderingClass( irc );
```

## 26.18. Batch report mode

Batch report mode is a feature that enables running multiple reports into one output. Batch mode makes possible to join two or more reports in a specified order and run them as one report. Page numbering doesn't change, each member reports keep its own number of pages. In reports the `reportno` and `reportcount` system variables can be used for determining the current report number and the total number of reports in batch.

### 26.18.1. Using batch mode

Batch mode is enabled when a report XML definition string is added by `addReportToBatch(...)` function. This example shows how we can easily prepare a batch report from existing report files.

```
...
report->clearBatch();
QString report1;
Utils::fileToString( "/home/anywhere/report1.xml", report1 );
report->addReportToBatch( report1 );

QString report2;
Utils::fileToString( "/home/anywhere/report2.xml", report2 );
report->addReportToBatch( report2 );

QString report3;
Utils::fileToString( "/home/anywhere/report3.xml", report3 );
report->addReportToBatch( report3 );
```

The order of reports in batch equals the order of applied `addReportToBatch(...)` commands

# Specification

This document is essentially a specification of NCReport Reporting System XML template structure.  
This is a brief documentation of report definition XML file structure.

# Chapter 27. Specification

Report XML template structure specification

## 27.1. Data sources

Since the report generator builds a printable representation of data from data, the the data source is one of the most important parts of the system. Data may come from an SQL query using Qt's database SQL database connection drivers or from other sources: **Text**, **XML**, **String list**, **Item model** or **custom** defined data source written in Qt/C++. One report can contain multiple data sources and each details can be connected it's own data source. Whenever a data source is not assigned to any detail, the initial (first) row of data in the data source is considered

### 27.1.1. SQL data source

SQL queries (sql data source) are the most common and widely used data sources for NCReport. It requires an SQL database connection using Qt's database driver plugins. The database connection can be either internal or external.

For an internal connection (the default), a valid database connection must be established by the application using NCReport before running the report. If an external connection is used, the connection parameters must be specified accordingly.

#### XML syntax

```
<datasource>[SQL query]</datasource>
<datasource>[query filename]</datasource>
```

#### Tag properties

##### id

data source ID. Identification name of the data source. Details are assigned to data source by this ID.

##### type

Specifies the data source type. Equals **SQL** for SQL data sources. Possible values are: **SQL**,**txt**,**xml**,**list**,**model**,**custom**

##### source

The source of the data source definition. Depending on this option the SQL query is stored and read from the report definition or from a specified file. Possibly values are: **static**,**file**,**parameter**

##### connection

Specifies the SQL database connection handling method. Possibly values are: **internal**,**external** With internal (the default) connection a valid database connection must be established by the application uses NCReport before running the report. If external connection is specified, the

report generator connects to the database when opening the data source. If this occurs the **host,database,user,password,port** possible connection parameters are used.

#### **connID**

The database connection's name that is used when the **QSQLDatabase::addDatabase(...)** method is called in the report engine. This ID is required for running SQL query which is assigned to the data source

#### **parentID**

If the data source is a sub-item of a parent data source (sub-query system) then this ID equals to the ID of parent data source. Valid for SQL data sources only

#### **driver**

The name of the Qt's SQL database driver. The possible values are: **QDB2, QIBASE, QMYSQL, QOCI, QODBC, QPSQL, QSQLITE2, QSQLITE, QTDS**

#### **host**

Host name for SQL database connection. Used only when external connection is defined.

#### **database**

Database name for SQL database connection. Used only when external connection is defined.

#### **user**

Host name for SQL database connection. Used only when external connection is defined.

#### **password**

Password for SQL database connection. Used only when external connection is defined.

#### **port**

Port number for SQL database connection. Used only when external connection is defined.

### **27.1.2. Text data source**

Texts, text files, are able to be as a data source for NCReport. The data columns of a text are usually delimited by tab or other delimiter character. Even it's possible to avoid SQL database connection when using this kind of data source. It's necessary to set the delimiter type, this delimiter separates the columns and each row represents a data record. Text data sources can be static, stored in XML definition or can be a file

#### **XML syntax**

```
<datasource>[static text]</datasource>
<datasource>[filename]</datasource>
```

#### **Tag properties**



**id**

data source ID. Identification name of the data source. Details are assigned to data source by this ID.

**type**

Specifies the data source type. Equals **txt** for text data sources.

**source**

The source of the data source definition. Depending on this option the text is stored and read from the report definition or from a specified file. Possibly values are: **static,file**

### 27.1.3. XML data source

Extensible Markup Language (XML) format is also can be a data source for NCReport. If using

The following example shows how the data structure should look like

**Example**

```
<textobject><textdata fileref="codes/xmltestdata.xml" /></textobject>
```

**XML syntax**

```
<datasource>...</datasource>
```

**Tag properties****id**

data source ID. Identification name of the data source. Details are assigned to data source by this ID.

**type**

Specifies the data source type. Equals **xml** for XML data sources.

### 27.1.4. String list data source

It's possible to use also QStringList as data source for NCReport. Before running report, a QStringList must be assigned to the specified data source and also is necessary to set a delimiter character for separating columns in each list item that represents a data record.

**XML syntax**

```
<datasource></datasource>
```

## Tag properties

### id

data source ID. Identification name of the data source. Details are assigned to data source by this ID.

### listID

ID of the list. This identification name specifies the id of the `<classname>QStringList</classname>` added to NCReport.

### type

Specifies the data source type. Equals `**list` for string list data sources.

## 27.1.5. Item model data source

Qt's item model classes provide a generic model for storing custom data. For example `<classname>QStandardItemModel</classname>` can be used as a repository for standard Qt data types. It is one of the Model/View Classes and is part of Qt's model/view framework. It's possible to use item models as data source for NCReport. Before running report, a `<classname>QAbstractItemModel</classname>` based class must be created and assigned to the specified data source.

## XML syntax

```
<datasource></datasource>
```

## Tag properties

### id

data source ID. Identification name of the data source. Details are assigned to data source by this ID.

### modelID

ID of the model. This identification name specifies the id of the Model added to NCReport.

### type

Specifies the data source type. Equals **model** for item model data sources.

## 27.1.6. Custom data source

Often data is stored in special repository such as lists, arrays etc. You can build your custom data source class derived from `<classname>NCReportdata source</classname>` base class. It is an abstract class - you just have to implement the required methods.

## 27.2. Report sections

Report sections are the representations of the function specific areas inside the report. The whole

report is builded from sections. They are often a recurring areas such as detail, headers and footers. The most important section is called Detail since details can contain the fields are changed row by row. Each sections can contain all kinds of report items. Item's coordinates are always relative to it's parent section.

One report can contain the following sections: Report header, report footer, page headers, page footers, group headers and footers and details

### 27.2.1. Page header

Page headers is used to contain page headings. Page headers have the following characteristics: \* Always print at the top of a page\* Always contain the first information printed on a page\* Only display one (current) row of data returned by a data source \* Only one allowed per page

#### XML syntax

```
<pageheader>...</pageheader>
```

#### Tag properties

##### height

The height of the page header section in millimeters

### 27.2.2. Page footer

Page Footer are commonly used to close the pages. Page footers have the following characteristics:

- Always print at the bottom of a page\* Only display one (current) row of data returned by a data source\* Only one allowed per page
- Page footer is usually used to display informations like number of the page, report titles and so on.

#### XML syntax

```
<pagefooter>...</pagefooter>
```

#### Tag properties

##### height

The height of the page footer section in millimeters

### 27.2.3. Report header

Report header is used to contain report headings. Report header has the following characteristics:

- Always printed after the page header

- Report header is printed only once at the beginning of the report
- Displays only one (current) row of data returned by a data source

#### XML syntax

```
<reportheader>...</reportheader>
```

#### Tag properties

##### height

The height of the report header section in millimeters

### 27.2.4. Report footer

Report footer is commonly used to close the report. Report footer has the following characteristics:

- Always printed before the page footer at the end of the report
- Only display one (current) row of data returned by a data source
- Only one allowed per report

#### XML syntax

```
<reportfooter>...</reportfooter>
```

#### Tag properties

##### height

The height of the report footer section in millimeters

### 27.2.5. Details

The core information in a report is displayed in its Detail section. This section is the most important part of the report since it contains the row by row data from the data source. Detail sections have the following issues:

- Generally print in the middle of a page\* Always contain the core information for a report
- Display multiple rows of data returned by a data source\* The detail sections generally contain fields or dynamic objects.
- Multiple independent details are allowed in one report, each detail after the other\* Each detail is assigned to one specified data source

#### XML syntax

```
<detail>...</detail>
```

Structure:

```
<details>
  <detail>
    <items>...</items>
    <groups>...</groups>
  </detail>
  <detail>
    <items>...</items>
    <groups>...</groups>
  </detail>
  ...
</details>
```

### Tag properties

#### id

Name/ID of the detail for identification purposes

#### height

The height of the group header section in millimeters

#### data source

The data source name/id the detail section is assigned to

## 27.2.6. Group sections

While most reports can be defined using a single Detail section having multiple columns and rows of data, others require summary data - such as subtotals. For reports requiring summary data, the report writer supports Group sections. Group sections have the following characteristics:

- Always associated with a Detail section\* Defined by Group Headers and Group Footers
- Group Headers always print above it's Detail section\* Group Footers always print below it's Detail section
- Reference database column on which Group Headers and Group Footers will break
- Force new Group Header each time the value of the referenced column changes
- Force a new Group Footer each time the value of the referenced column changes
- Unlimited level of groups allowed

The groups added to XML definition are shown in the order you have added. They are structured hierarchically. The first group will be the primary level of group, the second one is the second level and so on. The added group sections will appear in the designer after you applied the group settings.

## XML syntax

```
<groups>
  <group>
    <groupheader>...</groupheader>
    <groupfooter>...</groupfooter>
  </group>
</groups>
```

### Tag properties

#### id

Identification label for naming the group

#### groupExp

Group expression or data source column. Specifies the name of the data source column on which Group Headers and Group Footers will break. The expression also can be a constant value, in this case the detail row won't break. The constant group expression: **%CONST**

#### resetVariables

The variable list appears the existed variables in the report. Just select the items represent the variables will be reset when the current group ends. Selecting the specified variables is very useful when for example you want to reset a total or a count variable.

#### reprintHeader

Item's Y coordinate in millimeter within the current section.

## 27.2.7. Group header

Group headers are used to contain group heading items such as column head titles or so on. They are always printed above it's Detail section. A new Group Header is forced each time the value of the referenced column changes.

## XML syntax

```
<groupheader>...items...</groupheader>
```

### Tag properties

#### height

The height of the group header section in millimeters

## 27.2.8. Group footer

Group footers are used to contain group footing items such as totals, subtotals. They are always printed below it's Detail section. A new Group Footer is forced each time the value of the referenced column changes.

## XML syntax

```
<groupfooter>...items...</groupfooter>
```

## Tag properties

### height

The height of the group footer section in millimeters

## 27.3. Report Parameters

Parameters are data pulled from outside of the report generator. The application that calls NCReport object passes informations as parameter to NCReport class by **addParameter(...)** method. Parameters are evaluated within SQL queries and fields or script expressions. Field objects may have a parameter data source type, so they can be presented as data in the report. Parameters mostly used in queries. If you want to embed a parameter into the query or an expression use this syntax:

```
$P{parametername}
```

Example of using parameter in SQL query:

```
SELECT productId, productName FROM db.products WHERE primaryKey=$P{parametername}
```

## 27.4. Variables

Variables are specific items of the report. Variables are special fields used for providing counts and totals. Each of the variables have name, function type, data type, and have an assigned data source column the variable based on. We will explain what the different function types mean:

### Count

The [ **COUNT** ] type of variable will increment by 1 for every row returned by a query.

### Sum

The [ **SUM** ] (summary) variable will summarize the value of the specified data source column. It requires numeric field type. To embed a parameter into an expression use this syntax:

```
$V{variablename}
```

## 27.5. System Variables

System variables are special variables that provide some report system informations such as page number, current date/time etc. for fields Names of available system variables are:

**pageno**

Returns the current page number

**pagecount**

Returns the count of total pages of the report. Works only for Text document printout mode.

**forcecopies**

Returns the number of total force copies

**currentcopy**

Returns the current number of force copy

**currentrow**

Returns the current detail row number

**date**

Returns the current date

**time**

Returns the current time

**datetime**

Returns the timestamp

**appname**

Returns the name of this application

**applongname**

Returns the long name of this application

**appinfo**

Returns the full info string of this application

**appversion**

Returns the version of this application

**appcopyright**

Returns the copyright info of this application

**qtversion**

Returns the Qt version

**os**

Returns the operation system

For variable fields or to embed a parameter into an expression use this syntax:



```
$V{systemvariablename}
```

## 27.6. Expressions

NCRReport since 2.0 version handles script expressions using Qt Script the new powerful feature of Qt 4.3. Qt Script is based on the ECMAScript scripting language, as defined in standard ECMA-262. Fields and group expressions may be script codes instead of data source column. The report engine evaluates the specified script code in each time when fields are refreshed. Report items can have **printWhen** property. They are also script expressions that return boolean result. To use script expression in fields the `fType="exp"` field property must be specified.

## 27.7. References in expressions

Expressions can contain and evaluate references such as

- data source data
- parameter
- variable

The references are always replaced to their current value before the expression is evaluated. The syntax of references are the following: `$D{[data source.]column[,n]}` data source column reference. Returns the current value of the data source column from the current row/record. If data source. is not specified the current data source (assigned to the current detail) is interpreted.

If `<parameter>n</parameter>` is specified then first, the data source will be positioned to `<parameter>n</parameter>` th. record. Works only if the `::seek( int )` method is defined in the appropriate data source class.

**`$P{paramatername}`**

Parameter reference. Returns the value of the parameter by name/ID

**`$V{variablename}`**

Variable reference. Returns the current value of the variable by name/ID.

## 27.8. Using script expression in field:

```
"$D{db.productName}"+ " "+"some string"+"$P{paramatername}"
```

Using script expression in **printWhen** property

```
$D{price}<1500
```

Quotation mark in expressions is required only if a string data are applied. Otherwise (i.e for

number) the quotation mark is not necessary.

## 27.9. Report items

### 27.9.1. Text label

The Label represents simple text or label items. Label items are used to display descriptive information on a report definition, such as titles, headings, etc. Labels are static item, it's values don't change when rendering the report.

#### XML syntax

```
<label>Text label...</label>
```

#### Tag properties

##### id

Identification number for internal purposes (temporarily not used)

##### posX

Item's X coordinate in millimeter within the current section.

##### posY

Item's Y coordinate in millimeter within the current section.

##### width

Label's width in millimeter.

##### height

Label's height in millimeter.

##### resource

Resource of the label. Not used for labels since they are always static.

##### fontName

Font style/face name

##### fontSize

Font size in points.

##### fontWeight

Font weight. Possible values are: **bold**,**demibold**

##### alignmentH

Label's horizontal alignment. Possible values: **left**,**right**,**center**

**alignmentV**

Label's vertical alignment. Possible values: **top,center,bottom**

**forecolor**

The foreground color of the label in **#RRGGBB** format

**zValue**

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

**printWhen**

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.

## 27.10. Fields

Field is the matter of report items. It represents the data Field objects. By data type Fields may be text, numeric and date. Field items are used for pulling dynamically generated data into a report from the specified data source such as database the report generator uses. For example, a Field item may be used to present SQL data, variables and parameters. NCReport handles data formatting for the different type of fields like numbers or texts.

### 27.10.1. XML syntax

```
<field>[data sourcename.]column</field>
<field>[expression]</field>
<field>[parametername]</field>
<field>[variablename]</field>
<field>[system variablename]</field>
```

### 27.10.2. Tag properties

**id**

Identification number for internal purposes (temporarily not used)

**type**

The field's base data type. The following data types are handled:

**txt**

Text data\* **num** Numeric data. All numeric formatting options are valid only when this option is set\* **date** Date data. The date formatting options are valid for date type data only\* **bool** Boolean data. It's value might be Yes/True or Not/False

**ftype**

This property represents what kind of field source expression is used by the field. Field's value

are pulled from the specified source is set by this property. The possible sources are:

### **ds/SQL**

The field gets data from the default or the specified data source\* **par** The field gets data from the specified parameter\* **var** The field gets data from the specified variable\* **sys** The field gets data from the specified system variable\* **exp** The field evaluates the script expression and it's result will be rendered

### **posX**

Item's X coordinate in millimeter within the current section.

### **posY**

Item's Y coordinate in millimeter within the current section.

### **width**

Field's width in millimeter.

### **height**

Field's height in millimeter.

### **resource**

Not used for fields since they are always dynamic.

### **fontName**

Font style/face name

### **fontSize**

Font size in points.

### **fontWeight**

Font weight. Possible values are: **bold**, **demibold**

### **alignmentH**

Field's horizontal alignment. Possible values: **left**, **right**, **center**

### **alignmentV**

Field's vertical alignment. Possible values: **top**, **center**, **bottom**

### **forecolor**

The foreground color of the field in **#RRGGBB** format

### **formatting**

If the field's data type is numeric, this option tells the report engine if number formatting is turned on or off. The possible values are: **true**, **false**

### **numwidth**

Width of number in digits. The **fieldWidth** value specifies the minimum amount of space that a is padded to and filled with the character **fillChar**. A positive value will produce right-aligned

text, whereas a negative value will produce left-aligned text. Works only when the number formatting is turned on

### **format**

This one digit option specifies the format code for numbers. Possibly values are: **e,E,f**. With **e,E** and **f**, precision is the number of digits after the decimal point. With 'g' and 'G', precision is the maximum number of significant digits. Used by `<function>QString::arg( double a, int fieldWidth: = 0, char format = 'g', int precision = -1, const QChar & fillChar)</function>` function.

### **precision**

The number of digits after the decimal point for numeric data.

### **fillchar**

The **numwidth** value specifies the minimum amount of space that a is padded to and filled with the character **fillchar**. A positive value will produce right-aligned text, whereas a negative value will produce left-aligned text.

### **callFunction**

Specifies the Field level custom function is called when the field is evaluated. Not used currently.

### **lookupClass**

Similar to callFunction. Temporarily is not used.

### **dateFormat**

Date formatting expression. This expression uses the same format **QDate::fromString()** uses. Works only when the field's type is date

### **localized**

Specifies if localization is turned on or off. Works for numeric data only. The possible values are: **true, false**

### **blankifzero**

If **true**, If the field's value equals zero, the field will not be displayed.

### **arg**

This expression specifies the **QString::arg(...)** string of field's value to be replaced or formatted. The field gets a copy of this string where a replaces the first occurrence of %1. The '%' can be followed by an 'L', in which case the sequence is replaced with a localized representation of a. The conversion uses the default locale, set by **QLocale::setDefault()**. If no default locale was specified, the "C" locale is used.

### **zValue**

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

### **printWhen**

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or

1) the item is shown, otherwise the item is hidden.

## 27.11. HTML Text

HTML Text represents the rich texts in Html format.

### 27.11.1. XML syntax

```
<text>Static (encoded) Html text</text>
<text>[data source].column</text>
<text>[filename]</text>
```

### 27.11.2. Tag properties

#### id

Identification number for internal purposes (temporarily not used)

#### posX

Item's X coordinate in millimeter within the current section.

#### posY

Item's Y coordinate in millimeter within the current section.

#### width

width in millimeter.

#### height

height in millimeter.

#### resource

Resource of the text. Not used for labels since they are always static.

#### fontName

Font style/face name. Effects only if system settings is enabled.

#### fontSize

Font size in points. Effects only if system settings is enabled.

#### fontWeight

Font weight. Possible values are: **bold**, **demibold** Effects only if system settings is enabled.

#### forecolor

The foreground color of the label in **#RRGGBB** format. Effects only if system settings is enabled.

#### zValue

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

## printWhen

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.

## 27.12. Line

The Line option enables you to create Line items. In general, Line items are used for drawing vertical, horizontal lines for headings, underlining titles or so on. Lines are defined by its start and the end point coordinates

### 27.12.1. XML syntax

```
<line></line>
```

### 27.12.2. Tag properties

#### id

Identification number for internal purposes (temporarily not used)

#### lineStyle

Specifies the line drawing style of the item. Possible values are: **solid** Solid line, **dash** Dashed line, **dot** Dotted line, **dashdot** Dash+dotted line, **dashdotdot** Dash+dot+dot line, **nopen** No line painted. Unavailable for lines

#### fromX

X coordinate of the start point of line in millimeters within the current section.

#### fromY

Y coordinate of the start point of line in millimeters within the current section.

#### toX

X coordinate of the end point of line in millimeters within the current section.

#### toY

Y coordinate of the end point of line in millimeters within the current section.

#### resource

Not used for lines since they are always static.

#### lineWidth

The width of the line in millimeters

#### lineColor

The color of the line in #RRGGBB format

### zValue

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

### printWhen

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.

## 27.13. Rectangle

The Rectangle enables you to create Rectangle items. Rectangles are usually used for drawing boxes or borders around a specified area. Rectangle makes easier the box drawings instead of drawing four lines.

### 27.13.1. XML syntax

```
<rectangle></rectangle>
```

### 27.13.2. Tag properties

#### id

Identification number for internal purposes (temporarily not used)

#### lineStyle

Specifies the line drawing style of the rectangle. Possible values are:

#### solid

Solid line\* **dash** Dashed line\* **dot** Dotted line\* **dashdot** Dash+dotted line\* **dashdotdot** Dash+dot+dot line\* **nopen** No line painted. The rectangle is rendered without outline

#### fillStyle

Specifies the fill style or painting brush of the rectangle. Possible values are:

#### no

Rectangle is not filled.\* **solid** Solid fill\* **dense1** Extremely dense brush pattern fill\* **dense2** Very dense brush pattern fill\* **dense3** Somewhat dense brush pattern fill\* **dense4** Half dense brush pattern fill\* **dense5** Half dense brush pattern fill\* **dense6** Somewhat sparse brush pattern fill\* **dense7** Very sparse brush pattern fill\* **hor** Horizontal lines pattern fill\* **ver** Vertical lines pattern fill\* **cross** Cross lines pattern fill\* **bdiag** Backward diagonal lines pattern fill\* **fdiag** Foreword diagonal lines pattern fill\* **diagcross** Crossing diagonal lines pattern fill

#### posX

Rectangle's X coordinate in millimeters within the current section.



**posY**

Rectangle's Y coordinate in millimeters within the current section.

**width**

Rectangle's width in millimeters.

**height**

Rectangle's height in millimeters.

**resource**

Not used for rectangles since they are always static.

**lineWidth**

The width of the outline in millimeters

**lineColor**

The color of the rectangle's outline in **#RRGGBB** format

**fillColor**

The fill color of the rectangle in **#RRGGBB** format

**zValue**

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

**printWhen**

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.

## 27.14. Image

The Image option enables you to create Image items. Image items are used to insert either static or dynamic into a report definition. Static images such as a company logo often displayed in the Report Header can be loaded from a static file or from report definition. Dynamic images can be loaded from the specified SQL data source.

### 27.14.1. XML syntax

```
<image>[image in Base64 encoded format]</image>  
<image>[image file name]</image>  
<image>[data source.]column</image>
```

### 27.14.2. Tag properties

**id**

Identification number for internal purposes (temporarily not used)

**resource**

Specifies the resource of the image item. Possible values are:

**static**

Image is loaded from report definition. The image must be saved into XML definition in Base64 encoded format\* **data source** Image is loaded from data source (SQL database)\* **file** Image is loaded from the specified file. File might be with full path or relative to the program's directory

**posX**

Image's X coordinate in millimeters within the current section.

**posY**

Image's Y coordinate in millimeters within the current section.

**width**

Image's width in millimeters.

**height**

Image's height in millimeters.

**scaling**

Logical option that specifies the image if is scaled or not. Possible values: **true,false**

**aspectRatio**

If scaling option is switched on, this property specifies the scaling method. Possible values:

**ignore**

The size of image is scaled freely. The aspect ratio is not preserved.\* **keep** The size is scaled to a rectangle as large as possible inside a given rectangle, preserving the aspect ratio.\* **expand** The size is scaled to a rectangle as small as possible outside a given rectangle, preserving the aspect ratio.

**zValue**

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

**printWhen**

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.

## 27.15. Barcode

The Barcode option enables you to create barcodes. Currently the EAN13 code format is supported. Barcodes might be either static or dynamic items similar to images. Static barcodes read it's value

from the report definition, dynamic barcodes are loaded from the specified data source.

### 27.15.1. XML syntax

```
<barcode>[code]</barcode>  
<barcode>[data source.]column</barcode>
```

### 27.15.2. Tag properties

#### id

Identification number for internal purposes (temporarily not used)

#### resource

Specifies the resource of the barcode item. Possible values are:

#### static

Barcode is loaded from report definition. The barcode's code must be specified in XML definition\* **data source** Barcode is loaded from data source

#### posX

Barcode's X coordinate in millimeters within the current section.

#### posY

Barcode's Y coordinate in millimeters within the current section.

#### width

Barcode's width in millimeters.

#### height

Barcode's height in millimeters.

#### barcodeType

The type name of the barcode. Possible values: **EAN13**

#### showCode

The logical property specifies if the code is shown under the barcode or not. Possible values: **true, false**

#### sizeFactor

This integer property specifies the zooming factor of the barcode when it is rendering. This property is very useful when we print barcodes to a high resolution device such as printer. (Suggested value=10)

#### fontSize

The font size of the barcode's text in points.

**zValue**

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

**printWhen**

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.

## 27.16. Graph or custom item

Graph/Custom item is a special member of NCReport items. This option enables you to render special, custom defined contents in reports. The typical field of application is using this feature for rendering graphs or such contents. For using this feature you need to do the followings:

- Add a Graph (Custom) item into your report in the designer and specify the size and the location of this object.
- Set the class ID of the specified item
- If need, add a static item definition for the object. If you set it's resource to data source and fill out the data source column, this information will come from the specified data source column.\* Derive the `<classname>NCReportAbstractItemRendering</classname>` class implementing it's `paintItem` method. You may 'stick' this class to your graph or any kind of rendering class by multiple inheritance. The `paintItem` method gets the following parameters:

**QPainter\* painter**

this is the painter pointer.

**NCReportOutput\* output**

the output object pointer.

**const QRectF& rect**

the rectangle of the object in the specified output. The geometry of the rectangle is depending on the output's resolution.

**const QString& itemdata**

item definition information comes from data source or report definition for custom purposes.

- Set the string ID of your class for identification by **setID(...)** method.
- Create your custom rendering object (it must to be derived from `<classname>NCReportAbstractItemRendering</classname>` class) and add it to NCReport object by using **addItemRenderingClass(...)** method.

### 27.16.1. XML syntax

```
<graph></graph>
```

## 27.16.2. Tag properties

### **id**

Identification number for internal purposes (temporarily not used)

### **classID**

Class ID text for custom item class identification

### **resourceSpecifies**

the resource of the graph item. Possible values are:

### **static**

Graph definition is loaded from report definition. The definition text must be existed in XML definition\* **data source** Graph definition text is loaded from data source

### **posX**

Graph's X coordinate in millimeters within the current section.

### **posY**

Graph's Y coordinate in millimeters within the current section.

### **width**

Graph's width in millimeters.

### **height**

Graph's height in millimeters.

### **zValue**

This integer number specifies Z-order value of the item. This value decides the stacking order of sibling (neighboring) items.

### **printWhen**

This logical script expression specifies the item's visibility. If this expression is not empty, the report engine evaluates it each time before rendering. If the logical expression returns true (or 1) the item is shown, otherwise the item is hidden.