

# **CuteReport**

# **User Manual**

version 1.3

# Table of Contents

<b>Introduction.....</b>	<b>3</b>
Licensing.....	4
<b>Designer.....</b>	<b>5</b>
Report Editor module.....	6
Page Editor module.....	8
Script Editor module.....	11
Dataset Editor module.....	13
Form editor.....	14
Translator module.....	16
Preview module.....	18
Designer Options.....	20
<b>Creating Report template.....</b>	<b>21</b>
Report objects.....	22
"Hello World" report example.....	23
Memo object.....	24
Rotating.....	24
HTML tags.....	25
Expressions.....	25
Text flowing.....	26
Memo Helper.....	27
Formatting.....	28
Bands.....	30
Storages.....	31
File System storage (Standard::Filesystem).....	31
GIT storage (Standard::GIT).....	31
Resource storage (Standard::Resource).....	31
SQL Storage (Standard::SQL).....	31
Datasets.....	33
"Customer List" example.....	34
Image object.....	37
Report with Images.....	38
Multi-lined text display.....	40
Text wrap of objects.....	43
Complex wrapping.....	44
Label printing.....	46
Multi-page report.....	49
<b>Script Engine.....</b>	<b>50</b>
Script objects.....	51
Script variables.....	52
Local variables.....	52
Global variables.....	52
Renderer variables.....	53
Forms.....	54
Functions.....	55
Spelling out.....	55
Formatting.....	57
Script Signals.....	59
<b>Using in custom application.....</b>	<b>62</b>
Project setting up.....	63
Embedded library.....	63
Standalone framework.....	63
Simple example.....	65
Custom application example.....	67
Datasets.....	68
Model Dataset.....	69
Setting of required translation.....	71

# Introduction

**CuteReport** is a report solution based on Qt framework and it can be easily used with any Qt application. In general, CuteReport consists of two parts: core library and template designer. Both are totally modular and their functionality can be easily extended by writing additional modules. It's totally abstract of used data and can use as storage: file system, database, version control system, etc. The project's goal is to provide powerful, but yet simple to use for inexperienced users and report designers, report solution.

## Key features:

- A number of data sources: SQL database, Text, File System, external data model (QAbstractItemModel);
- Various types of storage to keep report templates and report objects such as picture, database, template: File System, GIT, SQL database, embedded storage;
- Plain text or HTML support;
- Variety of drawing elements to construct great looking reports: Memo, Image, Barcode, Arc, Chart, Chord, Ellipse, Line, Pie, Rectangle;
- Picture sources: static, dataset, storage
- Unlimited number of detail bands within one report;
- Report Title and Summary;
- Page Header and Footer;
- Element grouping;
- Aggregate functions: count, min, max, avg, sum;
- Plugin system to support extending of any functionality;
- External parameters;
- Entire application full featured scripting engine to manage any aspect of report rendering;
- Measure units: Millimeters, Inches, Pixels;
- Standalone WYSIWYG template designer with ability to extend of any functionality by using custom plugin;
- Some pre-installed Designer plugins: Report Property editor, Page editor, Script editor, Dataset editor, Preview;
- Multi-platform;

# Licensing

CuteReport is distributed in 2 version:

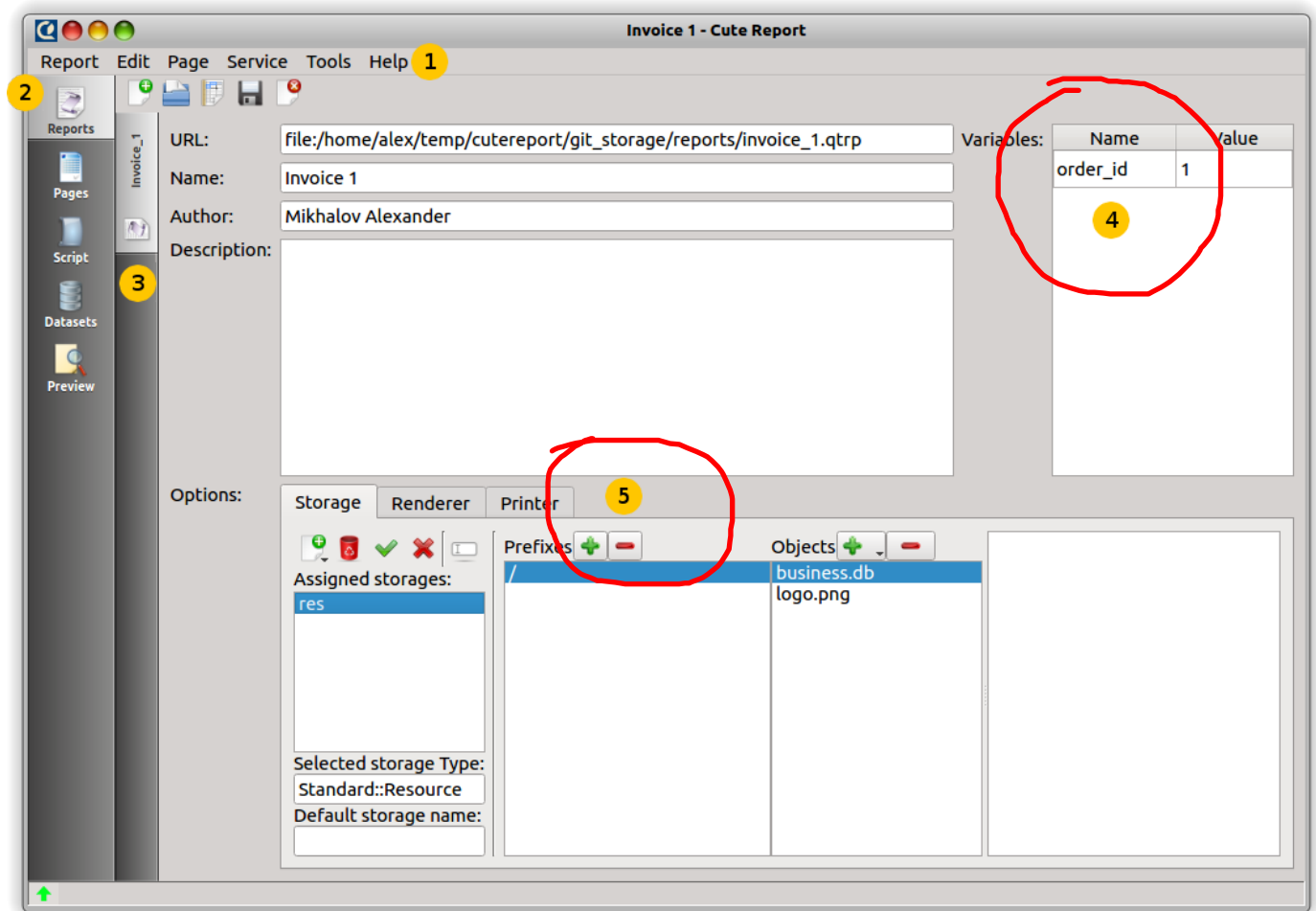
- *Community version* under GNU/GPLv3 license to help open source developers to add reporting functionality to their open source projects. Core Library is provided under LGPLv2 and can be dynamically linked to proprietary products. CuteReport Designer is provided under GPLv3 and it can not be compiled in to proprietary products. Read GPL/LGPLv2 license description for more information;
- *Professional version* under commercial license to provide high level support and highest priority of bug fixing and feature implementing. Also commercial package provides some CuteReport extensions that open source version does not have. In this documentation such features are marked as "Professional version only". To review commercial licenses visit project's web site <http://cute-report.com/en/article/licenses>;

# Designer

CuteReport solution comes with a standalone designer which helps to manage CuteReport templates. CuteReport Designer itself has only few basic functions and provides API to support modules. Modules are used to provide and extend any current or future designer's functionality. Module can provide user interface elements (GUI modules). Some of the basic GUI modules are: Report Editor, Page Editor, Script Editor, Dataset Editor, and Preview. Each module provides its own functionality and can be dependent of the other module(s). Also any module can be replaced by another one with extended functionality.

Lets take a look at the some of these modules.

# Report Editor module



## Key to report editor features:

1. main menu
2. modules bar
3. open reports tab
4. report parameters
5. embedded report modules

Report Editor is the first module on the designer tab bar. It is responsible for managing report objects and providing such report operations as: load, save, create, delete, etc. These operations are presented by controls on the Report Editor widget and also they are exported to the application's main menu. This module can support a number of open reports at the same time and switch between them. Also Report Editor manages embedded report modules like: storage, renderer, and printer. If there is no embedded object of storage, renderer or printer then the global one will be used. If you need special options for storage, renderer, or printer, you should add object of required type to the report object and set desired object's options. Report Editor has a table with global report variables and their values. These values are used in rendering report process and can be set manually by using this table or directly by an external application. You will often use this table to set default values for report parameters on testing stage before integrating it to your external application. We will review how to manage report

by your program later. For now we will focus on managing reports by using the Designer application.

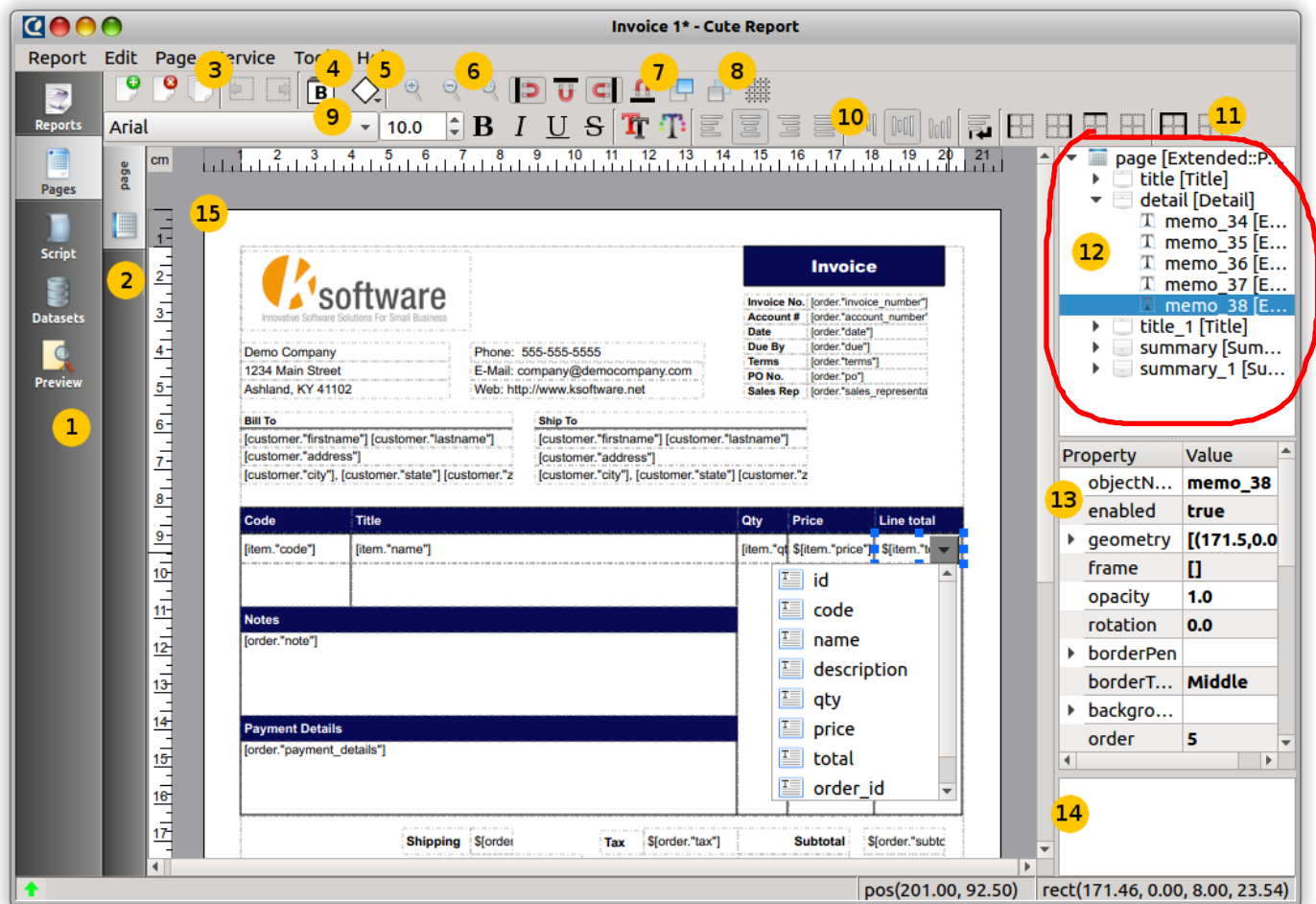
There all open reports are shown on the left bar (#3). Use it to switch between report templates. Main module view contains some fields: URL, name, author, description. URL is set automatically and shows path where report is stored. "Name" field contains report name. "Author" field contains report author. And description, as you can guess, contains report description. You can use these fields depending on your needs. They are not used by report engine. On the right side (#4), as we mentioned above, there is parameters table. Parameters are variables with an easy access from an external application. Parameter-variable appears there automatically if it is mentioned somewhere in scripting expression within string property of report's objects.. To test this behavior switch to the "Script" tab and type there "\${test}" without brackets. We will review Script module later. For now we can switch back to the Report Editor module and see how new variable "test" appears in the parameter table. Now you can assign any value to the parameter. You are also able to change this value directly from your application. Usually, you need to assign any temporary value to all variables to be able to test report template separately from your application.

Next is Options. This frame shows all embedded objects of storage, printer or renderer type along with their parameters. You might want to use embedded objects in case your report template is used on some computers with different access or settings or to avoid user interaction. For example, you create report template for a special printer and you do not want to show printing dialog and provide ability to user to change printer settings. Use embedded printer object with all predefined parameters. Or you might want to integrate company logo directly into report template instead of loading it from disk every time report is rendered. In this case you can add Resource storage and save your logo in there. It can be helpful to distribute CuteReport templates along with all their resources in one file.

There are some buttons for each list type: add object, remove object, set object as default, clear defaults, rename selected object. As you can guess "add" and "remove" are responsible for adding and removing object to/from report template. Default object of any type means it will be used if you do not specify object name. For example, you can load object into your report using full URL like "file:/home/user/images/logo.png" or just using "/home/user/images/logo.png" if you have set default storage. Bear in mind that you will have an error if you do not specify object name and do not have created default object.

Should be mentioned that object type specified by module name which object was generated by and has representation as "SuiteName::ModuleName". Standard module suite is "Standard". Commercial version has suite name "Extended". You can have third party suites as well with their own names. For example, standard SQL storage has name "Standard::SQL". At the same time extended SQL storage could have name "Extended::SQL".

# Page Editor module



## Key to Page Editor Features:

1. module bar with Page Editor activated
2. page bar
3. page tool bar
4. drop-down list of bands
5. drop-down list of items
6. zooming buttons
7. magnets enabling/disabling
8. rise/lower item
9. font editor
10. alignment editor
11. border editor
12. object inspector
13. property editor
14. property description
15. workspace

Page Editor module is responsible for making report page template. It provides tools for managing page bands and items. To activate Page Editor press the "Pages" tab on the modules bar (#1). Page bar (#2) shows all pages in the report. You can use it to switch between pages



(mouse left-click) or to rename current selected page (mouse left double-click). There are some buttons on the page tool bar (#3) to provide some basic actions for a page: create new page, delete current page, and clone current page. Next go 2 buttons with drop-down lists. First one is for band selection and the second one is for item selection. After that you can see some buttons for zoom operations and the next 4 buttons for enabling/disabling page magnets. If magnets are enabled, the mouse pointer will stick to the other item's borders with the coordinates which are closest to the current cursor coordinates. Sticking range factor can be changed in the page property with name "**magnetRate**". Page Editor has some other tools to change item properties, such as font editor, alignment editor, border editor.

On the right side of the page widget you can see Object Inspector. All items belong to page are represented there as an item tree. You can switch between items using Object Inspector or by clicking item on the workspace area(#12). Property Editor is the next element to review. There are all editable properties that page has. By pressing on property name you can see a short property description (#11). The Workspace (#12) shows the entire page template. You can add a new band or an item to your page using drag-n-drop functionality dragging the selected item from the drop-down list(#4, #5) to the page on the workspace. Almost all items can be placed only on a band, and bands can be placed only on a page directly. Press "Delete" button to delete the band or the item with all their children items.

It is possible to select some items and do group operations. To select some items click first item and then hold CTRL and click on the other item. You can click direct on item on the page or click item name in the Object Inspector. To perform further group operation use editors on the tool bar. The changes you make will be applied to all selected items if item allow it.

On the application status bar you can see current mouse position and geometry of the current selected object.

Some item might have helper. Sometimes it is easier way to change object's property. To open object helper, double click on the object. For example, if you click on Memo object you will be able to use text editor to enter Memo text or expression. Some other editors can be available too if appropriate modules are installed. Commercial version has some additional modules that are not provided for community version.

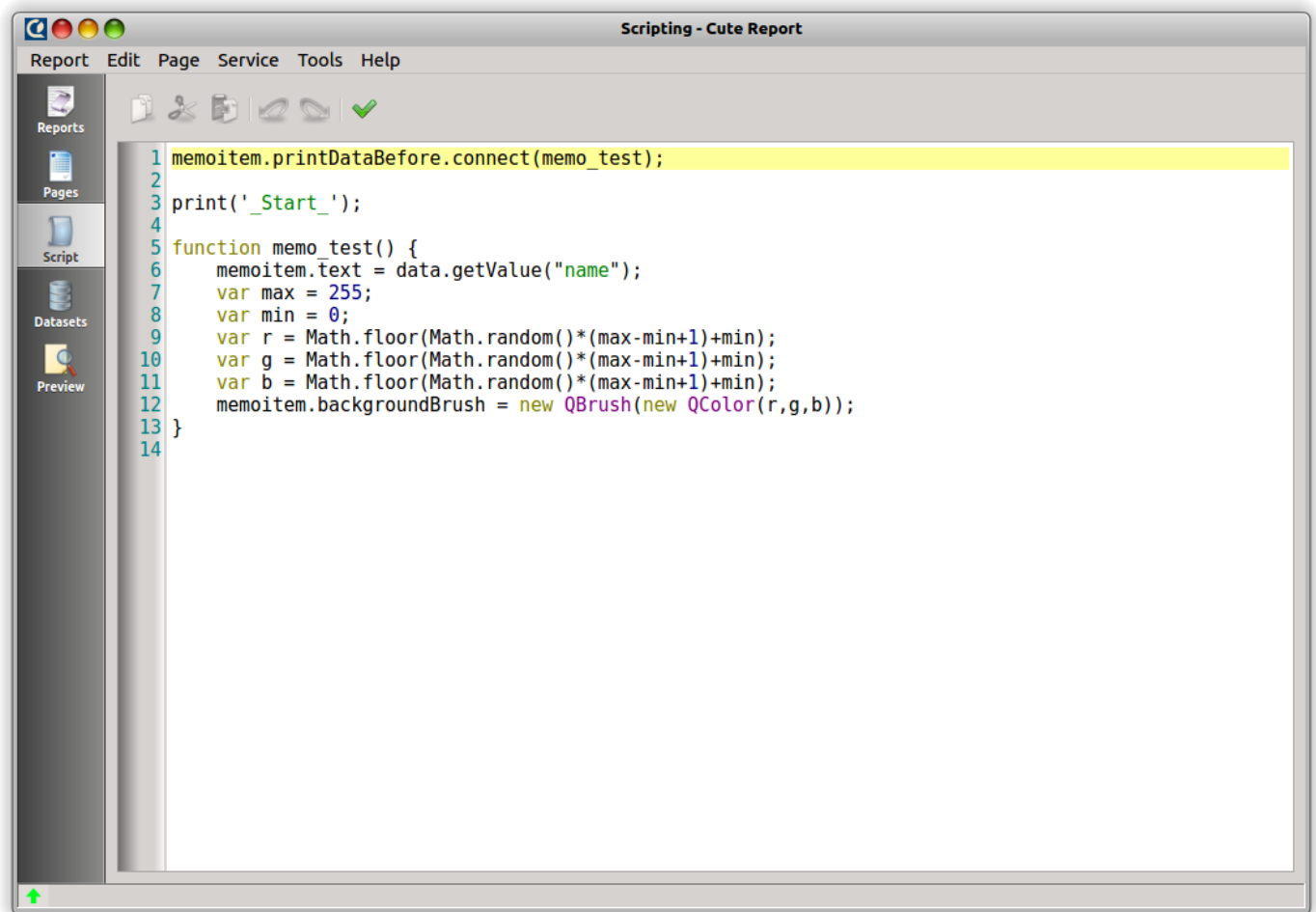
## Control Keys

Key	Action	Description
Ctrl+N	Report → New Report	Create new report template
Ctrl+O	Report → Open Report	Open report template
Ctrl+S	Report → Save Report	Save current report template
	Report → Save Report As...	Save current report template with another file name
Ctrl+W	Report → Close Report	Close current report template
Del		Delete current item

**Mouse controls**

Operation	Description
Left button	Select object; paste new object; move or resize selected object
Right button	Select object, paste new object; move or resize selected object with assigning new parent item at the mouse button leave position
Left double-click	Open object's helper
Mouse wheel	Scroll report page
Ctrl+left button	Add/remove object from selection group

## Script Editor module



You can switch to Script Editor by pressing the "Script" button on the module bar. Script editor is pretty simple module and contains an editor with syntax highlighting and "Validate" button. Validation checks only the syntax correctness of your script and does not actually run the script. So even if your script passed validation, it still can contain runtime errors. Usually you can see error list by pressing a green button in the status bar on the bottom of the Designer window. If there are errors exist in the script the button becomes red. Script Editor uses javascript as a scripting language.

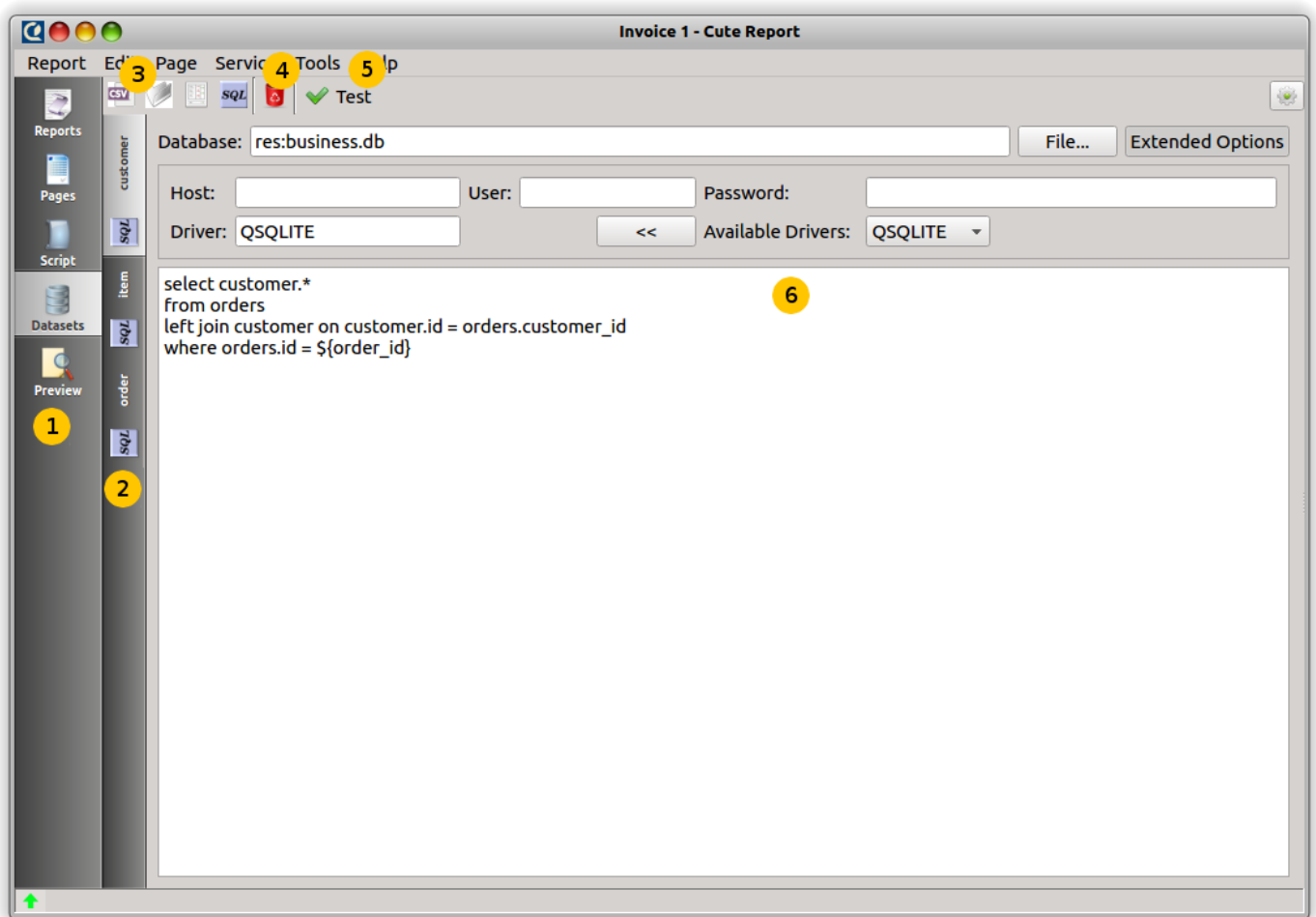
### List of controls:

Key	Description
Cursor arrows	Change cursor position
PageUp, PageDown	Go to Previous/Next page
Ctrl+PageUp	Got to text begin
Ctrl+PageDown	Go to text end
Home	Go to line begin
End	Go to line end
Enter	Go to Next line
Delete	Delete char in cursor position; delete selected text
Backspace	Delete char left to cursor position

Key	Description
Ctrl+A	Select all text

Since CuteReport uses standard javascript syntax, refer to JavaScript documentation if you experience difficulties with this language.

# Dataset Editor module



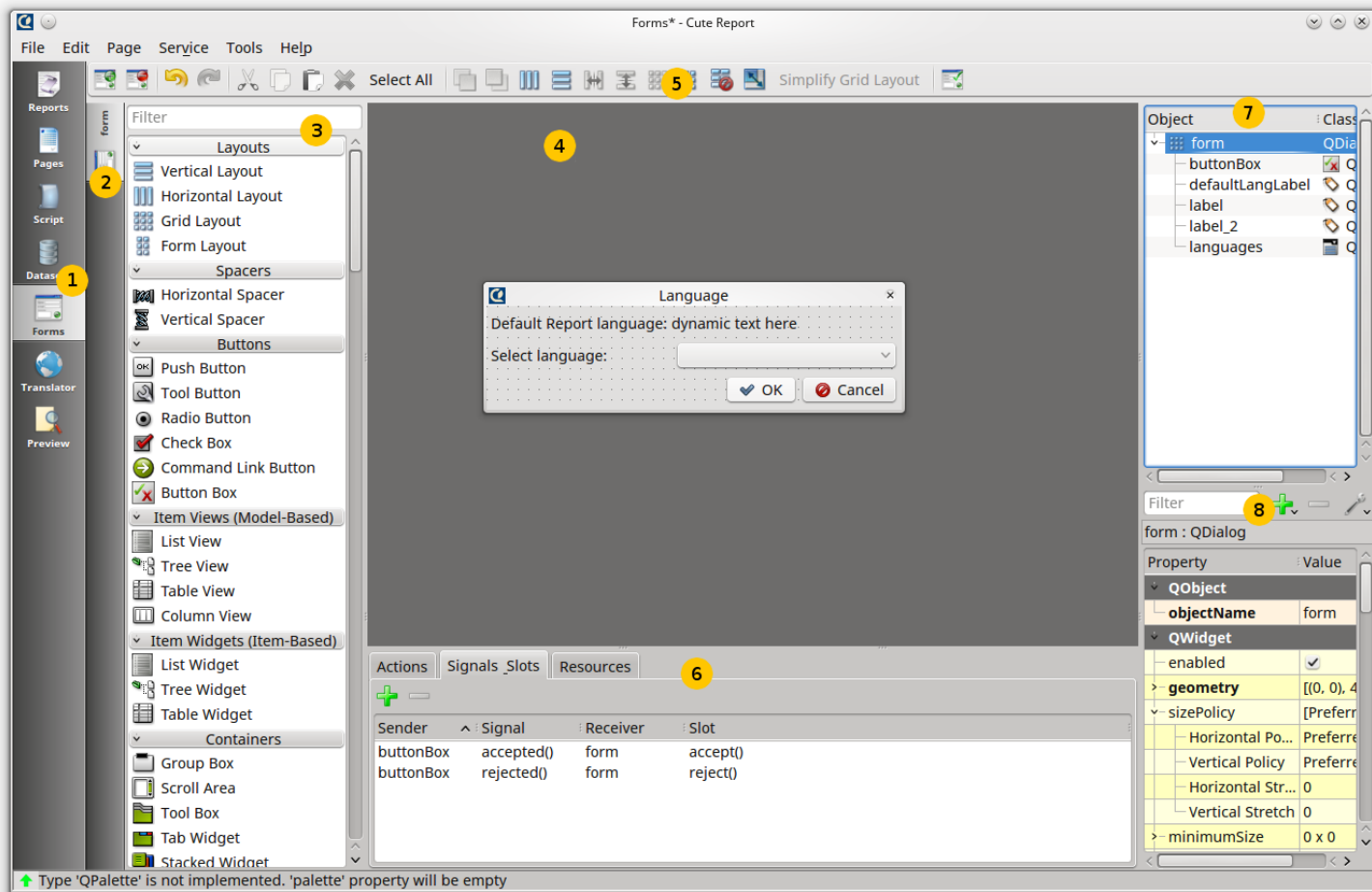
## Key to Dataset Editor features:

1. modules bar with Dataset Editor activated
2. datasets bar
3. create new dataset buttons
4. delete current dataset
5. test dataset
6. dataset helper

You can switch to Dataset Editor by pressing the "Dataset" button on the module bar (#1). All created datasets of the current report are shown on the dataset bar (#2). Using this bar you can switch between datasets (mouse click) or rename the current dataset (mouse double-click). For creating a new dataset press a button corresponding to required dataset type #3. Basic distribution provides 4 datasets: CSV dataset, SQL dataset, File System Dataset and Model dataset. These modules have names "Standard::CSV", "Standard::SQL", "Standard::Filesystem", "Standard::Model" accordingly. Read description for each dataset below. To delete current dataset, press button with trash bin icon #4. When you have set all the options for the created dataset, you can press the "Test it" button (#5) and check if everything correct. All datasets have a common interface and provide data as a table. Each dataset has its own configuration widget (#6).

# Form editor

(professional version only)



## Key to Form Editor features::

1. modules bar with Form Editor activated
2. form bar
3. widget panel
4. workspace with active form template open
5. tool bar
6. tools: Actions editor, Signal-Slot editor, Resources editor
7. object inspector
8. property editor

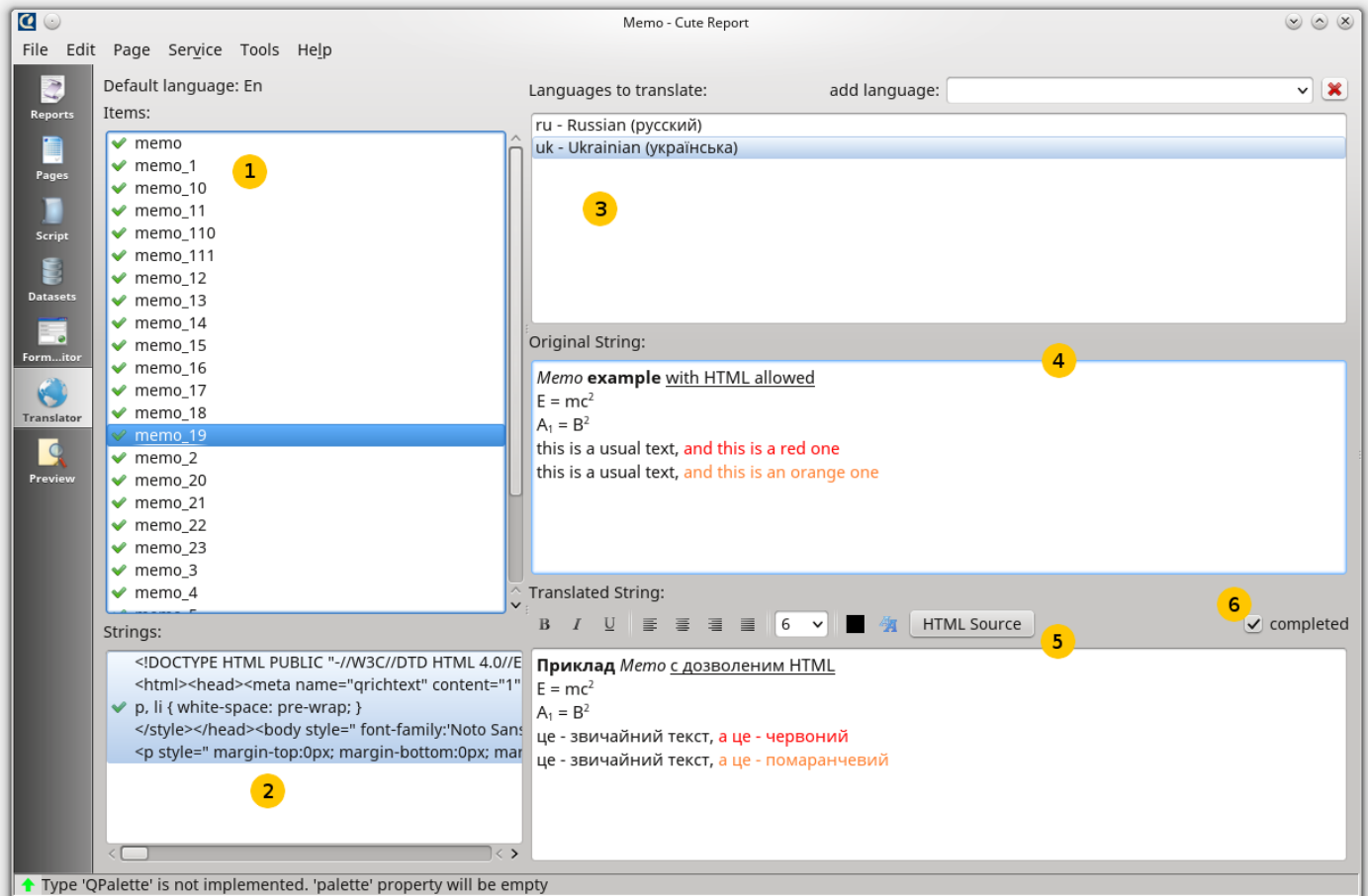
Form editor is a module to create a form templates to be used on rendering time. Form will be created to be used in user-report interaction. All form's widgets and their methods are accessible from the script engine. Using script to manage a form is the most natural and easy way. For more detailed information in this regard see chapter "Scripting engine → Forms".

To create a new form for your report click on the button "create new form" (leftmost on the toolbar #5). You will see a list with some preinstalled form templates. Choose one of them.

To set a correct form title use property editor (**#8**). The form should be an active element in the object inspector (**#7**). If it is not, click on the form name in the object inspector. A form is root element in the object inspector. All its child elements are represented as a branches of the root element. To place new widget to a form, find it in widget panel(**#3**) and drag to the form. To see a final look of your form press "Preview" button (rightmost on the toolbar).

# Translator module

(professional version only)



## Key to Page Editor features:

1. list of elements to be translated
2. list of text strings for selected element
3. list of report assigned languages
4. original text
5. translation editor
6. status of completeness

Translator is a yet another module in the Designer. This module makes possible to make translation for languages different from a report's default one. Suppose, original report language is English. In this case, you might want to add couple of other languages like German or Ukrainian. In the result your report will be available to read on 3 languages: English (default one), Ukrainian and German. Your default language will be present on a report page template and two other languages will be stored in the Translator. Original language is not visible in Translator.

In the list of elements to be translated (#1) contains all elements for which translation



can be performed. There is a translation completeness sign on the left of the element. It shows warning sign if element is not totally translated yet, i.e. has even one untranslated text string for even one assigned language. In the second window **(#2)** all the text strings for the current element are represented. If the an element has some translatable strings, you can select required one from this list. Often an element has only one translatable string. In this case this string will be selected automatically. In the list of assigned languages **(#3)** you can see all languages which report should be translated to. To add a new language to the list click on the “add language” drop-down list and select required language. In the field **#4** you will see an original string representation. In the field **#5** the current translation to selected language is shown. To mark current translation as completed check “completed” check box. If the original text string is changed the check box will be unchecked automatically. Translator automatically detects text formatting and if it is detected as HTML you will be able to see tools to edit HTML text.

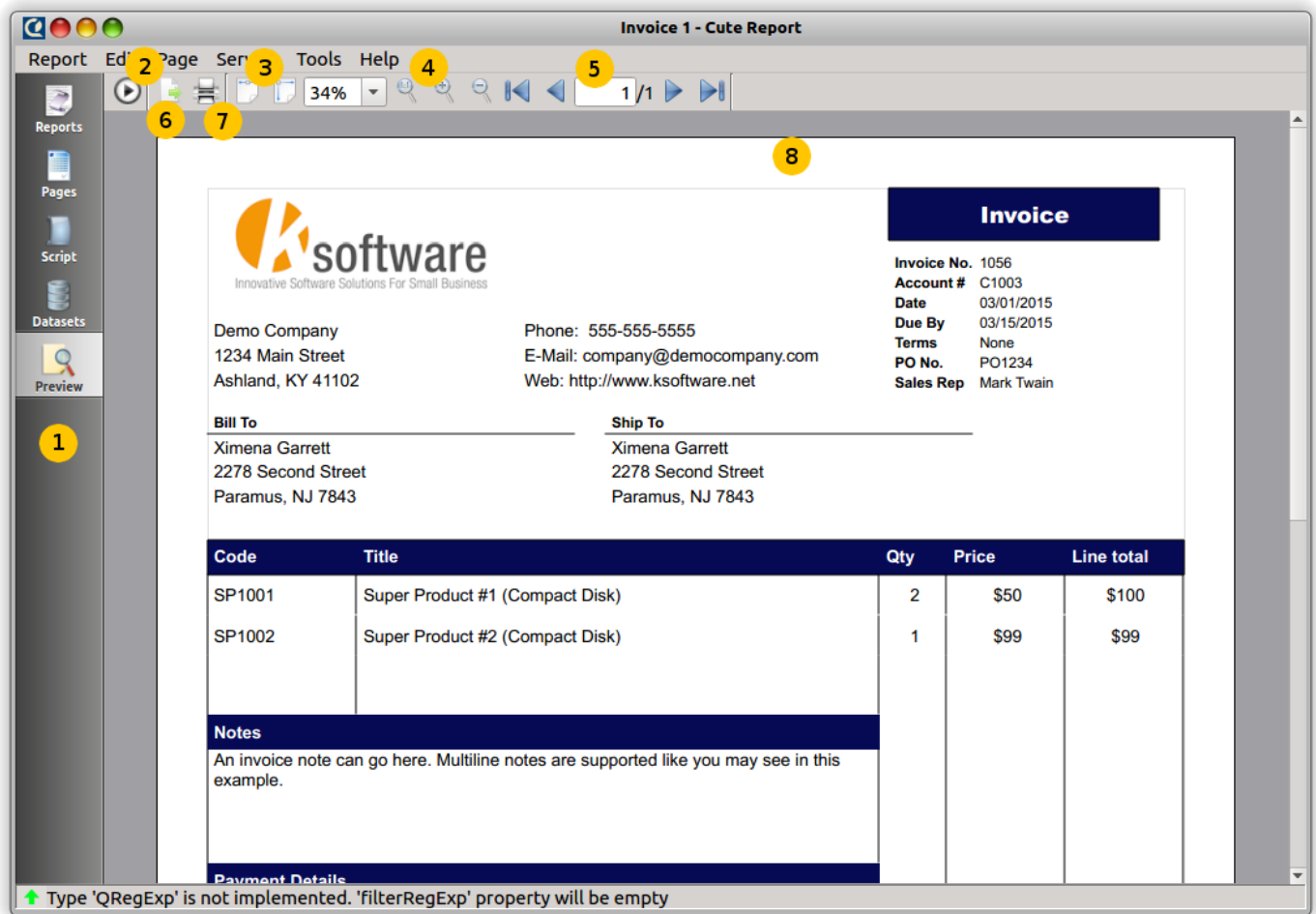
Need to notice, an original text of elements does not have to contain full text context to be printed. It can contain macroses i.e. short representation of a long text. The macroses will be replaced by full text representation on a report generation stage. That could be helpful to manage reports overloaded with long text fields. It might lead to page template with overlapping text fields and report could look messy. Macroses solve that problem. For example, for an element with a long text you could replace this long text with it short representation like “LONG TEXT HERE”. Then add your main report language to the list of languages in Translator and place your long text to the translation of the string. Thereby, your translation will contain a long text for the required string and the string itself will contain only short macros on a template.

To set required for rendering language switch to the Report editor module (look description of the module above). In the field “Variables” type two symbol language code for required language. On example below you can see how to set English language to render report.

Variables:

	Name	Type	Value
1	tr	String	en

## Preview module



### Key to Preview features:

1. module bar with Preview activated
2. rendering start/stop button
3. fit page to view button group
4. zooming button group
5. page navigation
6. export to file
7. print
8. rendered page

The job of the Preview module is to display rendered report pages. There are some useful button groups to help you. By first, there is the button to start rendering process of current report template (#2). Every time you need to render or re-render your report template this button will be helpful. As alternate variant you can use main menu action: Main Menu -> Service -> Render or just press F5 on your keyboard. If your report requires some time for rendering then process dialog will appear. To stop current rendering process press this button again (or press F5). When report is rendered you can change zoom by using buttons: fit to page, fit width, zoom in, zoom out (#3, #4) or you can set any zoom you want by entering percent value in percent representing widget. To change current page use buttons: First page, Previous Page, Next Page, Last Page or set page number directly in the page number widget. Finally, you

can print rendered report (#7) or export it to a file(#6).

# Designer Options

TODO

# Creating Report template

In this chapter we will review some general aspects of report designing. We will look close on some important items and theirs properties and will make some report examples. Make sure you have CuteReport installed and try to make these examples by yourself using pre-installed test databases. Since CuteReport is in active developing, some parts of this documentation can differ from your CuteReport installation.

## Report objects

CuteReport Designer's Page Editor module is designed to represent report as a set of schematic pages. All objects are placed somewhere on a report page and they are used to display any text or graphics information. Basic CuteReport objects are included to Community CuteReport edition package. Some extended objects are included to Commercial package.

Let us review object set.

### Bands:




- **PageHeader:** band located on the top of page
- **PageFooter:** band located on the bottom of page
- **Detail:** band that connected to dataset and processed with each dataset iteration
- **DetailHeader:** band that located on top of details group
- **DetailFooter:** band that located on bottom of details group
- **Title:** band that located before detail band(s)
- **Summary:** band that located after detail band(s)
- **Overlay:** band with the free accommodation, can be places anywhere on page without layouts

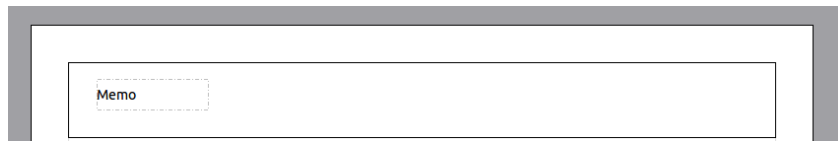
### Items:

- **Arc:** item that draws arc
- **Barcode:** item represents barcode
- **Chart:** item that draws any kind of charts
- **Chord:** items that draws chord
- **Ellipse:** item that draws ellipse
- **Image:** item that draws dynamic or static image in PNG, JPG, BMP and other formats
- **Line:** item that draws horizontal, vertical or diagonal line
- **Memo:** item that represents any text information, plain text and HTML formats are supported
- **Pie:** item that draws pie
- **Rectangle:** item that draws rectangle

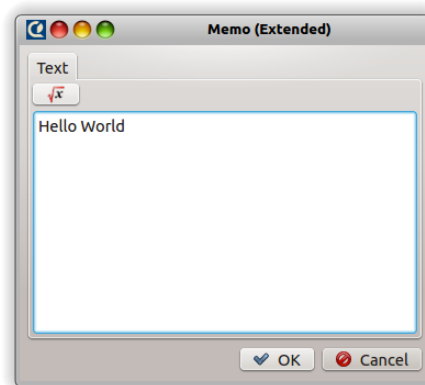
The basic objects most commonly used are the "Detail" band and "Memo" item. You will learn about their capabilities in detail later in this chapter.

## "Hello World" report example

This simple report example contains just one piece of information: "Hello World!" text. Open CuteReport Designer, create new report template using Report → Create Report, go to the "Page editor" using left tab panel and create new page. Since any item can be placed only on carrier band, we must place any band first. Click on the button "Bands"  and select any simple band, for example Page Header. Click somewhere on the page to place this band. Then click on the button with the title "Items" or icon  and select "Memo" . There is also Extended Memo item exists in the professional version, that extends functionality of the basic Memo. Use any of them if you have both. After selecting Memo item, click somewhere inside Page Header to place selected Memo. The object will be placed at the mouse position.



Depending of your local settings Memo Helper dialog will appear immediately or you can double-click on the Memo to show this dialog. Type "Hello World!" and then click "Ok" button.

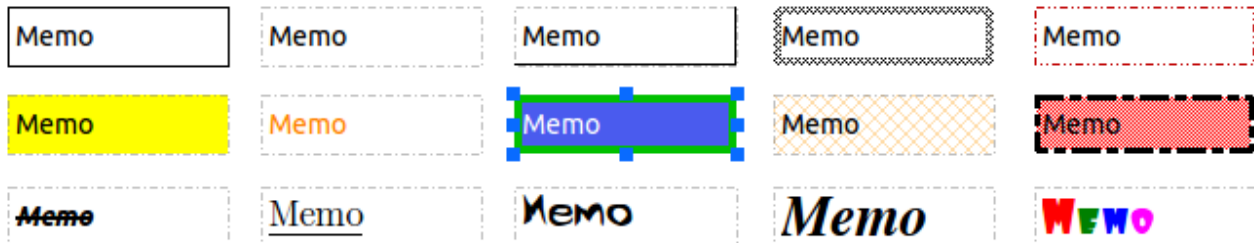


Now Report template is completed. To generate actual report select in main menu "Service → Run" or press "F5" on your keyboard. Designer will be switched to the "Preview" tab and rendered report page with "Hello World!" will appear. Rendered report can be printed or exported to one of the supported export formats.

## Memo object

The Memo object has many great features to draw text. It can draw text in a frame and can be filled with some color. The text can be displayed using any font with any size and style. All the properties can be set using Property Editor or visually with the help of the tool bar editors.

Here you can see some samples:

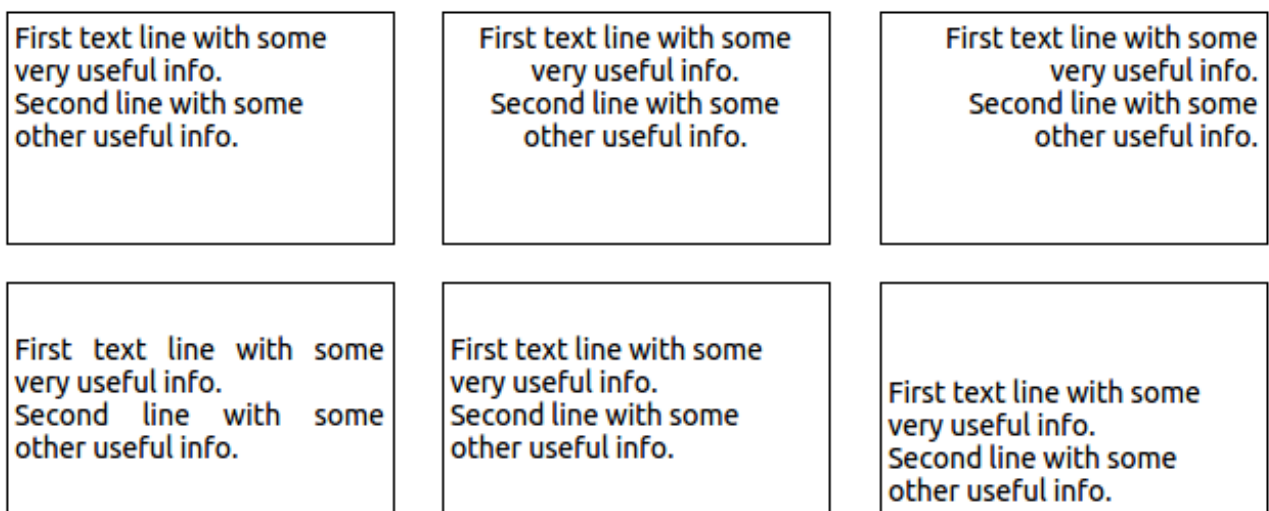


We will make a simple example of Memo with two lines of text:

*First text line with some very useful info.*

*Second line with some other useful info.*

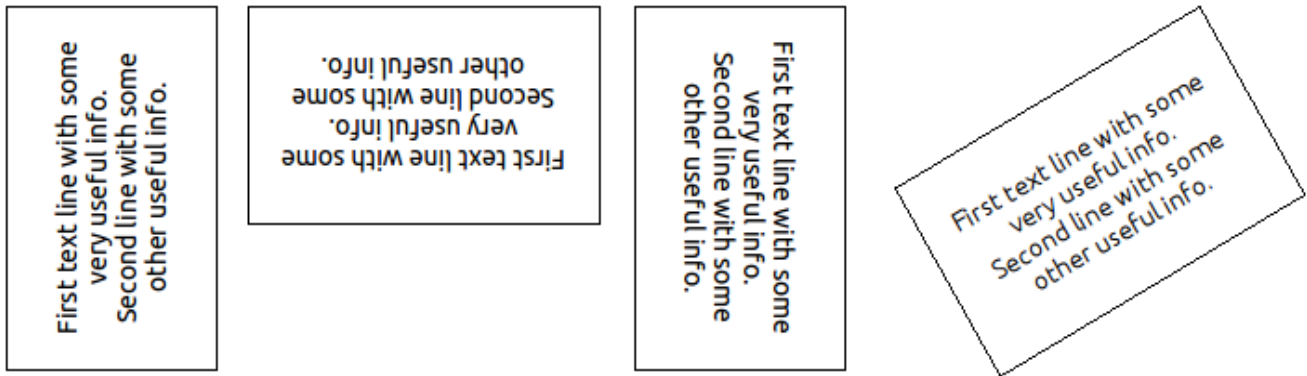
Enable Memo borders from Property Editor and resize item up to 90x30 mm using mouse or Property Editor. As you can see now Memo can display not only a single line but several lines of text as well. Try to reduce Memo width to 50 mm. Obviously, lines can not fit to the object's border and will be wrapped. This is controlled by **TextFlags::TextWordWrap** object property. If it is disabled any long line will be cut short. Lets play with other **TextFlags** and see what we can obtain.



## Rotating

Lets take a look at the other feature: rotation. Any object including Memo can be rotated to any angle in degree range 0..360. Set required angle in the Property Editor by changing property **"rotation"**. Memo borders will be aligned accordingly, so you don't need to care about the borders.





## HTML tags

Memo object allow most of HTML tags. Tags can be placed within Memo text. Tags are disabled by default. For HTML tags detection set property "allowHTML" to "true". There are some examples below.

```
<i>Memo</i> <b>example</b><br>
E = mc<sup>2</sup><br>
A<sub>1</sub> = B<sup>2</sup><br>
this is a usual text, <font color=red>and this is a red one
</font><br>
this is a usual text, <font color="#FF8030">and this is an orange one</font>
```

<i>Memo example</i>
E = mc <sup>2</sup>
A <sub>1</sub> = B <sup>2</sup>
this is a usual text, and this is a red one
this is a usual text, and this is an orange one

## Expressions

Expression is one of the most important feature of Memo object. It allows to display not only static text but runtime expression result as well. Expressions can be mixed with a static text. To learn how it works enter the text above to the Memo object:

Now is [QDateTime.currentDateTime()]

(or simplified variant with taking locale into account: "Now is [DATE]")

and render report by pressing **"F5"** on your keyboard. You'll see result of the rendering, something like that:

Now is Fri Jul 18 2014 00:44:22 GMT-0700 (PDT)

(or for DATE variable: 18/07/2014 or 07/18/2014 depending of your locale)

Why is that so? CuteReport renderer recognizes every expression instance, calculates it

and replaces expression with its result. Memo text can contain a number of expressions. Expression can use complex arithmetic, constants, variables, objects and their properties. But there are some possible issues may occur if our normal text contains square brackets that do not mean to be an expression. For example, we want to draw:

```
array[0] = 'Banana'
```

Entry [0] will be recognized as an expression, calculated with result 0 and will be placed to our text. As result we will see:

```
array0 = 'Banana'
```

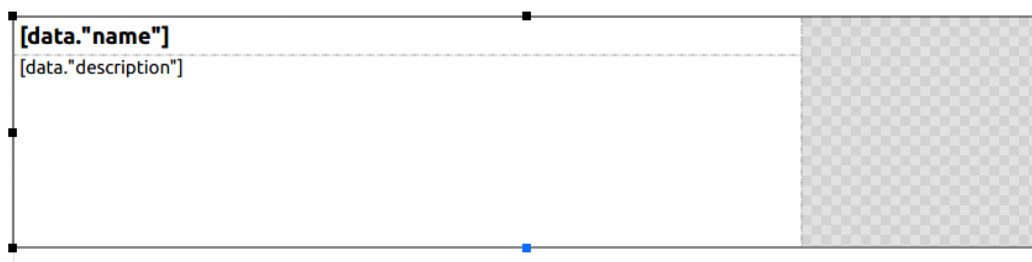
which is definitely not that we supposed to see. There are 2 solutions:

- set **"allowExpressions"** property to "false" to disable any expression into current Memo item
- change default square brackets to any symbols or symbol set you want in the **"expressionDelimiter"** property

In the first case entire text will be recognized as a regular text without expressions. In the second case expressions will be recognized using another "begin sign" and "end sign". You can use "<" as begin and ">" as the end, but only if you don't use HTML text. If you do use HTML you might use "<%" as begin and "%>" as the end. Expression detector works before text rendering, so any of your expression delimiters will be cut off from your text. Do not use the same symbol set as the "begin" and the "end" sign.

## Text flowing

Text flow feature available in the professional CuteReport version. It allows you to wrap text of objects. Let's make a simple example to demonstrate this feature. Here is our template to show a list of animals along with its descriptions (memo\_1):



When we render our example we will have such result:

### Capybara

The Capybara is a large, semi-aquatic rodent that is found inhabiting the water-logged regions of Central and South America. Closely related to other South American rodents such as Chinchillas and Guinea Pigs, the Capybara is the largest rodent in the world weighing up to 75kg and measuring nearly 1.4 meters long. Despite their enormous size though, these mammals have adapted well to life in the water and have a number of distinctive characteristics that aid their amphibious lifestyle, including the webbed skin between their toes which is particularly helpful when swimming. Interestingly enough, the common name of the Capybara is thought to mean "Master of the Grasses", whilst it's scientific name comes from the Greek word for water hog.



### Abyssinian

The Abyssinian Cat is thought to be one of the oldest breeds of



There is noticeable empty space below the image. It would be good to fill this space with a text. To do this we will add new Memo item under the current description Memo, disable stretching for first Memo. Instead we set property "flowTo" for our second Memo object (memo\_2) to "memo\_1". It should look like there:

[data."name"]

[data."description"]



Our rendered result:

### Capybara

The Capybara is a large, semi-aquatic rodent that is found inhabiting the water-logged regions of Central and South America. Closely related to other South American rodents such as Chinchillas and Guinea Pigs, the Capybara is the largest rodent in the world weighing up to 75kg and measuring nearly 1.4 meters long. Despite their enormous size though, these mammals have adapted well to life in the water and have a number of distinctive characteristics that aid their amphibious lifestyle, including the webbed skin between their toes which is particularly helpful when swimming. Interestingly enough, the common name of the Capybara is thought to mean "Master of the Grasses", whilst it's scientific name comes from the Greek word for water hog.



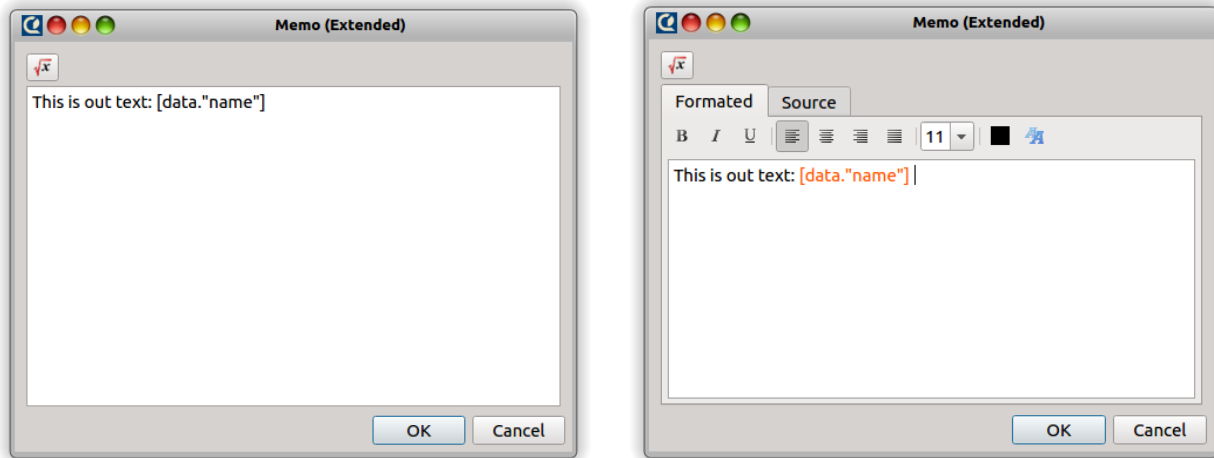
It is much more pretty. Isn't it?

One of the next chapter will be specially devoted to the text flowing functionality

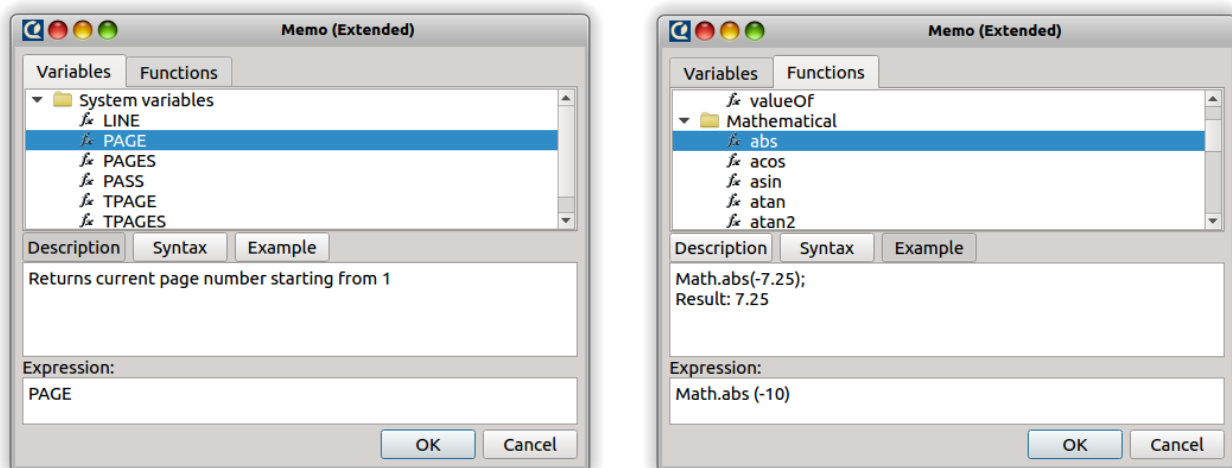
## Memo Helper

Using Property Editor to change Memo text or expression is not the only way. You can

use Memo helper. It is useful to enter large text or compose expression or HTML formatting. To show Memo helper double click on your Memo item. Depending of Memo property "allowHTML", appeared dialog will be having or not having HTML tool bar. There are 2 variants for these both cases:



Use HTML controls like in any other HTML editor. Also use can use direct HTML tags to compose required HTML code in the tab "Source". All changes made there will be immediately reflected in the "Formatted" tab. You can mix HTML code with expressions without any limitation, except the limitation to use correct expression delimiters that do not interfere with HTML tags. If your installation has installed additional modules that provide additional functionality, you will see appropriate additional buttons on the top. In our example we have Expression Editor installed. If you press the button  $\sqrt{x}$  Expression Editor form will appear. It has some tabs and represent variables, functions, methods that can be used to compose expression:



As you can see on the pictures there are available full description, syntax description, one or more examples for each function. By double clicking on the function name you will add it to your text expression. Pay attention to sigh "~" in some of the added functions. This means it is required to type correct value of the function parameter.

## Formatting

[TODO]

Detailed description of formatting string see in Chapter “Script Engine” → “Formatting”.

#### Other Memo properties:

- **stretchMode:** object stretchability to fit text
- **showStretchability:** do stretching not only on rendered page, but on Designer's Page Editor as well
- **expressionDelimiter:** two strings separated by comma which means the begin and the end of scripting block.
- **stretchFont:** set automatic font size to fit Memo text within Memo width

## Bands

Bands are designed for dynamic or static positioning of items on a page. Each band has its own position and functionality. CuteReport has some special bands designer to represent data from a dataset. Dataset contains structured data organized into rows(lines) which have one or more columns(fields). To print data from datasets CuteReport uses these special bands named "Detail...". To make it work, add one or more such bands to the page, connect them to the dataset and place Memo items on them. Once band is connected to the dataset, Memo will have button on its right side with the drop-down list of dataset's field names (commercial version only). At the rendering time these bands will be printed on the rendered page. "Detail" band usually will be printed once per dataset row, DetailHeader and DetailFooter will be printed accordingly to theirs "condition" property. If there is no free space to print new band on the current page, new rendered page will be automatically created to continue. New page will print all page headers and footers before continue to print Detail bands. This process is called rendering.

There are some bands included into the standard package and theirs short description:

- **PageHeader:** displays all nested items on the top of the page
- **PageFooter:** displays all nested items on the bottom of the page
- **Title:** displays all nested items on the report begin
- **Summary:** displays all nested items on the report end
- **Detail:** must be joined to a dataset and it displays all nested items on every dataset iteration
- **DetailHeader:** must be joined to a dataset and can be shown on every dataset iteration or when "condition" property calculated as "true". It can be used to show header for group of lines.
- **DetailFooter:** must be joined to a dataset and can be shown on every dataset iteration or when "condition" property calculated as "true". It can be used by the same way as detailHeader but at the bottom of the group.
- **Overlay:** can be located anywhere on a page without respecting any layouts. Can be used as carrier band for foreground, background, watermarks.

## Storages

Now we review another class of report elements. Storage is a structure that keeps all data used in a report such as images, templates, databases, etc. CuteReport provides some standard types of storage: File System, GIT, resource, Database. Certainly, you are familiar with some or all of them. Further we will review each storage detailed, but for now let's review common aspects of storage class. In the CuteReport you can create any number of instances of each type. It is useful if you have some remote systems that store reports or one system with different settings. Like one GIT server with read only access and one GIT server with full access. In this case you can create 2 storage objects of type "GIT" and set appropriate parameters to each of them. All storages in the CuteReport have their own unique name, based on a storage schema. So 2 created objects with type "GIT" will have names "git\_1" and "git\_2". Any name can be changed to any desired string, but it still should be unique. CuteReport core will not allow you to assign already used name. To get access to the object on the storage you will use path in URL style, like: "file\_1:/file\_path/image\_file.jpg", where "file\_1" is the storage name and "/file\_path/image\_file.jpg" path to the file located on the storage. It is possible to assign default storage in the Designer Option dialog using "Tools → Options → Storage", so any file name with unspecified storage name will pass to the default storage.

There are 2 kind of storage with different priorities: global storages and report's internal storages. When CuteReport engine meets storage url by first it checks internal report storages, then global storages and then default storage. So if you have well specified storage destination and options for your report, add it to report internally in the module tab "Reports". Your report will always use this storage not regarding of the global storage settings.

### File System storage (Standard::Filesystem)

FileSystem is the most commonly used storage. It has only few options: "root folder" and "ask for rewrite". Root folder is the upper directory accessible to user. "Ask for rewrite" option is used to detect if it is needed to show dialog for overwrite file.

### GIT storage (Standard::GIT)

This type of storage can be used to keep all reports and their objects in local or remote GIT version control system. It has such options:

- remote url: git repository url
- login, password: credentials to access git repository
- local path: local directory where git repository will be cloned to
- git binary: git console binary. CuteReport uses external git binary to operate git repository, so it has to be defined
- "sync now" button: button for cloning or pulling data from a remote repository

### Resource storage (Standard::Resource)

All objects stored in this storage will be included to report's template file.

### SQL Storage (Standard::SQL)

This storage is designed to help you easily save and load report templates and report objects from SQL databases without writing any code in your application. Just provide info about the database and correct data table and field where the reports or object are stored.





# Datasets

As it was mentioned before Dataset is an object that contains structured data organized into rows(lines) with one or more columns(fields). Dataset can fetch data from any source and it provides common interface to the data. There are some datasets provided in the basic CuteReport edition: SQL Dataset, CSV Dataset, FileSystem Dataset, Model Dataset. All of them fetch data from different sources.

## Datasets:

- **SQL Dataset (Standard::SQL):** It provides an interface to fetch data from an SQL database. It can work with any database supported by Qt itself, ie that has Qt database driver. There are some settings to connect to a remote database (like mysql, postgresql) or to use embedded database file (like sqlite)
- **CSV Dataset (Standard::CSV):** It provides an interface to a data stored in a file and separated by comma or other predefined symbol. It can load data from an external file every time when populated or load and cache the data internally
- **Filesystem Dataset (Standard::Filesystem):** It provides an interface to fetch information from a file system and show disks, files and directories information as a row structured data. There are some options: filtering, recursion level, max number of exposed records. This dataset can be used to make goods list, photo catalogs, etc.
- **ModelDataset (Standard::Model):** It is used to export prepared data from a custom application to a report object. If you have widget like QTableView or QTableWidget or your custom filtered/sorted model you can easily print data stored in there using this dataset.

Try to play with each of the dataset to understand how they work. You can fetch data and see it in table by pressing "Test it" button. Data from all dataset is exposed to scripting engine and can be used by any report object. To get data from dataset where it's allowed you can use [datasetname."fieldname".lineNumber] or [datasetname.getValue("fieldname", lineNumber)] expression. First form is just shortening of the second one. Every short form replaced by full form internally before script execution automatically. Line number can be skipped in any form. In this case, current dataset line will be used.

For more detailed information regarding each dataset refer to chapter Datasets.

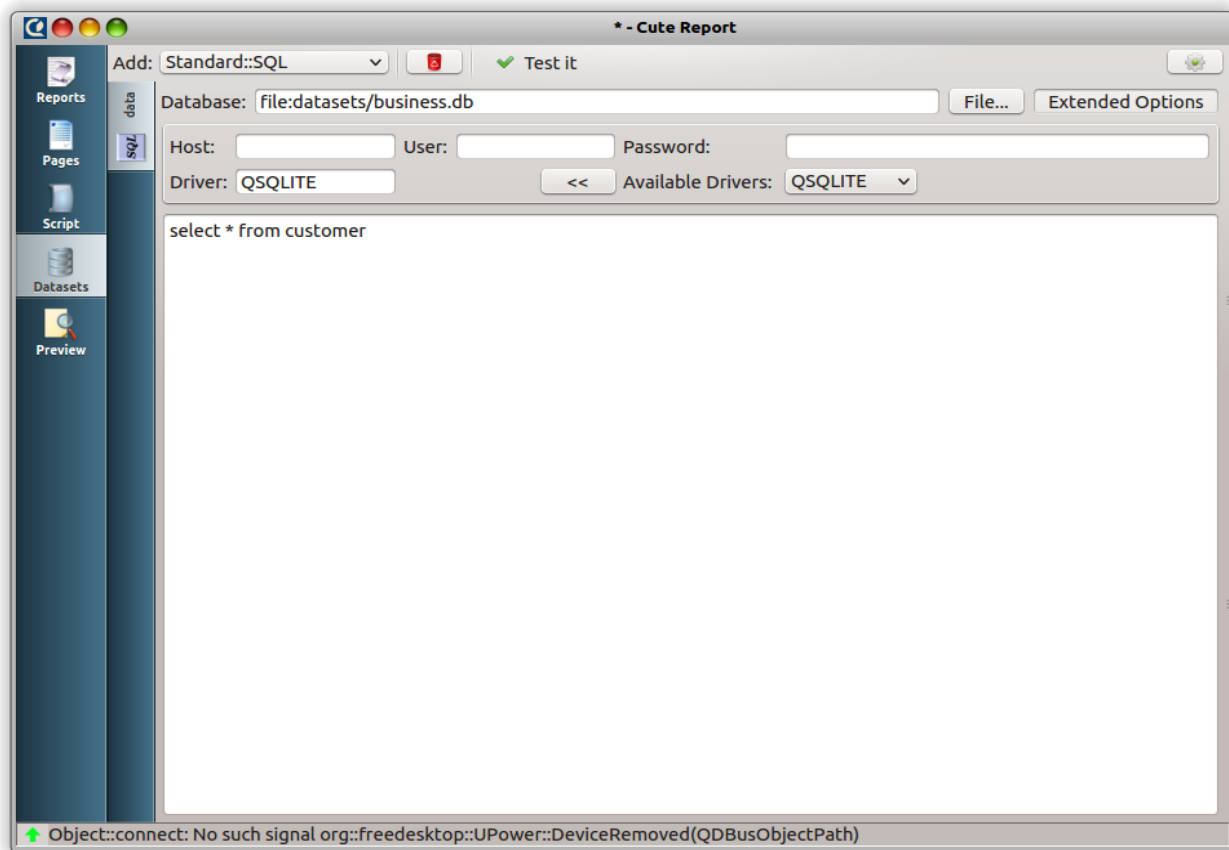
Further we will see how to connect dataset to a band and how to use dataset's data.

## "Customer List" example

Now we will create our second report and will learn how to use datasets. For that we will use test sqlite database named "business.db". By first, create new empty report by pressing "main menu -> Report -> New Report". Then go to the "Datasets" tab, click on the dataset names combobox and select SQL. Now you have one SQL dataset in your report. Lets set correct parameters for the dataset. Click "File...". When "Open database file" dialog appears, make sure you have correct storage name selected in the "Storage" combobox, locate database file named "business.db" and choose it. Now you should have field "Database:" filled with something like "file:dataset.db". Since we have no host, user and password for this database we will skip these fields. Choose "SQLITE" driver in the "Available Drivers:" list and press "<<" to copy driver name to the field "Driver". Now we can connect to our test database. Lets write a simple SQL query to test it:

```
select * from customer
```

As a result you will have something like on the picture below:



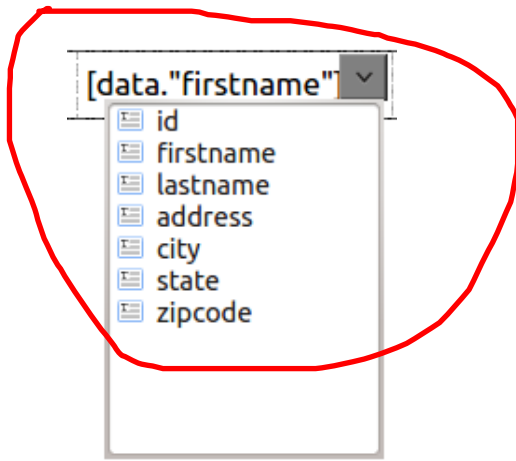
Click "Test it" button and table with the fetched from the database data will appear. If there is an error exists you will see it in the status bar bottom frame. Well done! Now you have dataset completed.

	id	firstname	lastname	address	city	state	zipcode
1	1	Ximena	Garrett	2278 Secon...	Paramus	NJ	7843
2	2	Jaunita	Johnson	5505 Park ...	Drummond	OK	73124
3	3	Calista	Baskin	2017 Seven...	Boca Raton	FL	94765
4	4	Tyra	Marcotte	6745 Fifth ...	Springboro	PA	54322
5	5	Steve	Neeley	793 Main St...	Edward	NC	76348
6	6	Vida	Brock	395 Sixth Ci...	Wrightsboro	TX	89372
7	7	Carolyn	London	990 Hill Str...	Metaline Falls	WA	67901
8	8	Tia	Bergman	4327 Cedar...	Jacksonville	NC	76320
9	9	Brady	James	875 Lake Ci...	Stockholm	WI	98347
10	10	Alia	Bohannon	5306 maple...	Seiling	OK	75349
11	11	Andeana	Pratt	1096 Eight...	Des Moines	IA	10345
12	12	Ellens	Vanmeter	403 Elm Cir...	Orchard	CO	80632
13	13	Waylon	Hardison	1078 Fourt...	Harmon	IL	86432
14	14	Ankti	Vanwinkie	3956 Eight...	Byron	CA	94321

Object::connect: No such signal org::freedesktop::UPower::DeviceRemoved(QDBusObjectPath)

Go to the tab "Page" and create new page if it not exists. There are some possible types of page. For now we will use "Standard Page" or "Extended page"(for commercial version). Click on the combobox, select appropriate page type and then click on the button "Add new page" . Add "Title" band to the page, and set correct size by mouse dragging on a blue handle or by setting a correct size in the Property Editor. Now place "Memo" item to the center of the band and set a correct geometry as well. Now double click on the Memo and type "Customer List" and then press "Ok". To make this text centered, click on the Memo's "TextFlags" property in the Property Editor and enable "AlignHCenter" flag. Next step is adding "Detail" band to the page and joining it to our customer dataset. To do that activate the Band by clicking on it and type "data" in the band's property "dataset". Entered text "data" is the name of our dataset. You can change the name of any dataset by double-clicking on the dataset tab in the Dataset Editor. Now it's time to add some "Memo" items to the band: number, first name, second name, address, city, zip code. To make item arrangement more easy, you can enable magnets by pressing magnet buttons on the top of the Page Editor. If you want to change name of the object and make it more understandable in the Object Inspector, go to Property Editor and change "objectName" to something like: memoFirstName, memoLastName, memoAddress and go on. We do not use these names in this example, but it can be useful for you later. Next is adding instruction to our Memos what to display. Go to the first one, double click on it and type "[LINE]" in the Helper's editor. As it was explained before, "[ ]" means expression borders and the "LINE" text is an internal variable that keeps current dataset row number. Press "F5" and you will see how it works.

Now return back to the Page Editor by clicking on "Pages" on module tab bar and fill the memo with the customer's first name. Add text "[data.firstname]" to this Memo. Users of the CuteReport commercial version can simply click on the gray button that appears on the right side of the Memo as you can see here:



This drop-down list contains all the fields in the dataset Memo related to. It works only for Memos located on bands that have property "dataset" filled and the filled dataset exists and properly initiated.

Do the same for all other items and set correct dataset field to draw. Press "F5" and you should see something like this:

A screenshot of a report titled "Customer list". The report displays a table with 14 rows of customer data. The table has 6 columns: an index column, first name, last name, address, city, and zip code. A red circle is drawn around the title "Customer list".

	First Name	Last Name	Address	City	Zip Code
1	Ximena	Garrett	2278 Second Street	Paramus	7843
2	Jaunita	Johnson	5505 Park Boulevard	Drummond	73124
3	Calista	Baskin	2017 Seventh Street	Boca Raton	94765
4	Tyra	Marcotte	6745 Fifth Circle	Springboro	54322
5	Steve	Neeley	793 Main Street	Edward	76348
6	Vida	Brock	395 Sixth Circle	Wrightsboro	89372
7	Carolyn	London	990 Hill Street	Metoline Falls	67901
8	Tia	Bergman	4327 Cedar Avenue	Jacksonville	76320
9	Brady	James	875 Lake Circle	Stockholm	98347
10	Alia	Bohannon	5306 maple Avenue	Seiling	75349
11	Andeana	Pratt	1096 Eighth Boulevard	Des Moines	10345
12	Ellens	Vanmeter	403 Elm Circle	Orchard	80632
13	Waylon	Hardison	1078 Fourth Boulevard	Harmon	86432
14	Ankti	Vanwinkie	3956 Eighth Circle	Byron	94321

If you don't have such result, you can load report CustomerList.qtrp from the CuteReport package and figure out what you did wrong.

# Image object

Image object is designed to represent any images in the supported graphical formats. Currently supported formats: BMP (Windows Bitmap), GIF (Graphic Interchange Format), JPG (Joint Photographic Experts Group), JPEG (Joint Photographic Experts Group), PNG (Portable Network Graphics), PBM (Portable Bitmap), PGM (Portable Graymap), PPM (Portable Pixmap), XBM X11 (Bitmap), XPM (X11 Pixmap).

Lets look closer at this object. Create a new report, add a new page, add an Image item. There are some data sources for the Image available: Static, Storage, Dataset. Type of source is defined in **sourceType** property.

Let's review each option:

- **Static:** allows you to load Image from a file and keep loaded data inside the object. File must be loaded manually by pressing "image" property in the Property Editor.
- **Storage:** allows you to load file in runtime Image property "source" must contain the path to the required file. If path does not contain storage name default storage will be used. Property "source" can contain expressions. For example, if source is defined as "file:[data."filename"]" then report engine it will try to take file name from the database "data" and then will try to load this file from the storage with name "file".
- **Dataset:** allows you to load file from a dataset blob. Image property "source" should point to the dataset field that contains image blob. For example: [data."image"]

A list of some other important properties:

- **keepAspectRatio:** If it is set to "true", image will always keep original aspect ratio while scaling.
- **Center:** If it is set to true, image will always be centered in its frame.

## Report with Images

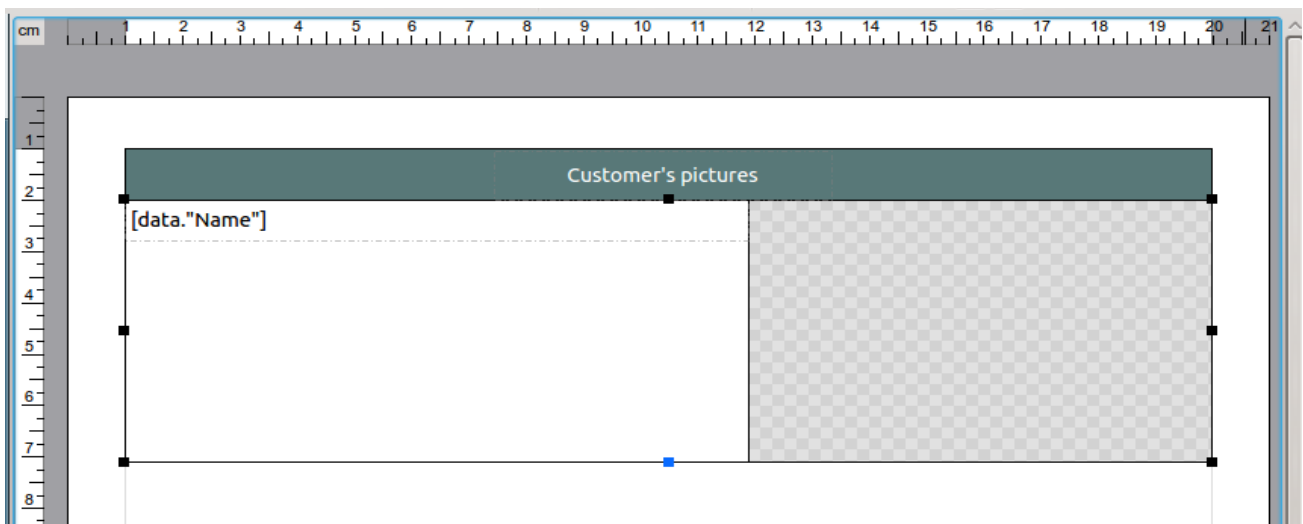
In this chapter we will learn how to use an Image object and a File System storage. Let's create a new report, add a new page, add Title and Detail band. Next is creating File System dataset. Go to "Datasets" tab, choose Standard::FileSystem and click it. Choose any directory that contains some pictures by pressing "Select dir..." button. Set "maxNumber" to 6. This is maximum records that dataset will fetch. Disable flags: Directories, All Directories. Add filter: ".jpg; .png" or any other graphic formats you want. Set "Path appearance" to "AbsolutePath", so we will be able to load picture using its absolute path. Now press "Test it". As a result you should see the list contains 6 files or less with the file format you have set in your filter.

Now go to the Page Editor. Add Memo object to the Title band, type there "Customer's pictures" and make text centered. For the better appearance change "backgroundBrush" for the Title. Set "backgroundBrush::style" to "SolidPattern" and "backgroundBrush::color" to #688482. Now change text color. Click on the Memo and change property "textColor" to white. Well done!

Now we are starting to create image frame. Set height of the Detail to 30mm. Add Image object to the right side of the Detail and change its size to fit to the detail. Set "sourceType" as "Dataset" and "source" as "file://[data."name"]". As you remember everything inside the square brackets will be identified as an expression and will be replaced by the expression result. So our expression finally will be looking like "file://picture\_path/picture.jpg". Well, there is another very important thing: **if report loads any data in runtime, it MUST have storage assigned!**. This is done for security. In some cases you may not allow user to use any storage CuteReport has. So go to the "Reports" tab, and add "Standard::FileSystem" storage in the "Storage" tab. We have images path as absolute path, so clean up "Root folder" for this storage.

And the last step is creating Memo item that contains file's path. Add a new Memo to the left side of the Detail band and type there "[data."Name"]". Users of the Commercial version can simply press the button that appears when you hover mouse over the Memo.

Finally you report template should look like this:



Now it's time to press "F5" and enjoy result:

Customer's pictures

/home/alex/temp/cutereport/git\_storage/objects/116521662132



/home/alex/temp/cutereport/git\_storage/objects/1218095387\_c



/home/alex/temp/cutereport/git\_storage/objects/57.GIF



/home/alex/temp/cutereport/git\_storage/objects/67.GIF



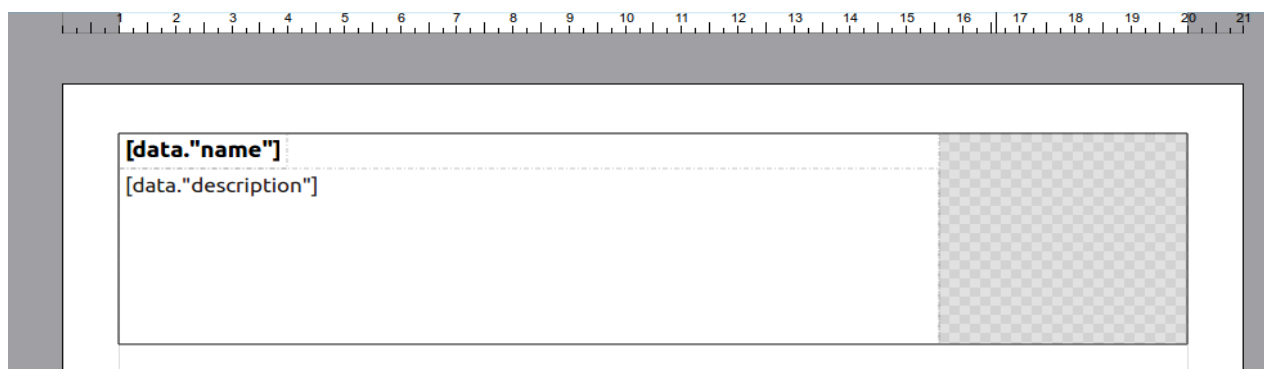
/home/alex/temp/cutereport/git\_storage/objects/914813250.jpg



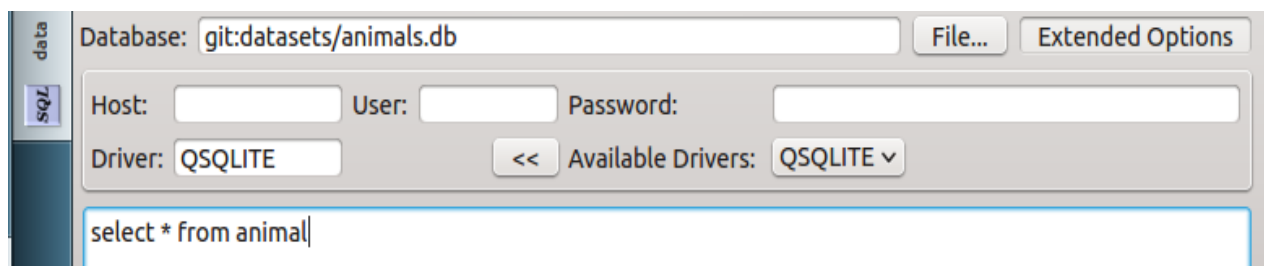
## Multi-lined text display

Let's proceed further and learn how to manage multi-line text. In the previous chapter we have learned how to make new report, create dataset and connect dataset to a band. So let's do it:

- create new empty report
- add a new SQL dataset and use test sqlite dataset "animals.db" which you can find along with the CuteReport distribution or take it there:  
[https://github.com/AlFoX/CuteReport\\_examples/tree/master/datasets](https://github.com/AlFoX/CuteReport_examples/tree/master/datasets).
- add Detail band
- put 2 Memos on the band: first is the animal name, second one is the animal description
- put Image item to the right side of the band



Now let's look closer how to create Dataset. When you have added SQL dataset, point it to your "animals.db" file, set Driver - "SQLITE" and add the sql query - "select \* from animal". Press "Test it" and check if all fine. Below you can see how it should finally look.



When this part is completed, go back to the Page Editor and set correct fields to display for the Memo's. The first topmost Memo is an animal name. So double-click it and type: [data."name"]. Users of the commercial version, can simply click on the appeared button on the right side of the Memo and choose field from the drop-down list. If it doesn't appear check if your band is connected to the correct dataset, i.e. dataset under the Memo has filled field "dataset". Set Bold text for the animal name by clicking on the "font" property in the Property Editor or by using font editor in the tool bar. Now go to the second Memo. It is animal description. So type or select there: [data."description"]. Simple, huh? Let's look the result and render our report (press "F5"):



### Capybara

The Capybara is a large, semi-aquatic rodent that is found inhabiting the water-logged regions of Central and South America. Closely related to other South American rodents such as Chinchillas and Guinea Pigs, the Capybara is the largest rodent in the world weighing up to 75kg and measuring nearly 1.4 meters long. Despite their enormous size though, these mammals have adapted well to life in the water and have a number of distinctive characteristics that aid their amphibious lifestyle, including the webbed skin between their toes which is



### Abyssinian

The Abyssinian Cat is thought to be one of the oldest breeds of domestic Cat in the world, as the first domestication of the Abyssinian Cat occurred in Ancient Egyptian times. It is thought that Abyssinian Cats were bought and sold on the banks of the River Nile by traders, where the African Wild Cats (the ancestors of all domestic Cats) lived in their native habitats. Abyssinian Cats are most easily identified by their "ticked" fur which gives their coat a mottled appearance.



### Adelie Penguin

The Adelie Penguin is the smallest and most widely distributed species of Penguin in the Southern Ocean and is one of only two species of Penguin found on the Antarctic mainland (the other being the much larger Emperor Penguin). The Adelie Penguin was named in 1840 by French explorer Jules Dumont d'Urville who named the Penguin for his wife, Adelie. Adelie Penguins have adapted well to life in the Antarctic as these migratory Birds winter in the northern pack-ice before returning south to the Antarctic coast for the warmer summer months.



### Baboon

The Baboon is a medium to large sized species of Old World Monkey that is found in a variety of different habitats throughout Africa and in parts of Arabia. There are five different species of Baboon which are the Olive Baboon, the Guinea Baboon, the Chacma Baboon, the Yellow Baboon and the Hamadryas Baboon which differs most from the others wide it's bright red face and cliff-dwelling lifestyle (the other four species are collectively known as Savanna Baboons). However, there is some debate over the classification of the different species



### Camel

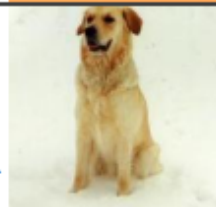
The Camel (also known as the Dromedary Camel, the Arabian Camel and the One-Humped Camel) is a large hoofed animal that is most commonly found in the hot deserts of Northern Africa and the Middle East. Thought to have been first domesticated by native people more than 5,000 years ago, these hardy animals have proved vital to the survival of humans in these areas as they are not just used for transporting both people and goods, but also provide a good source of milk, meat and wool. The Camel is one the most unique mammals on the planet.



### Dog

Dogs are thought to have been first domesticated in East Asia thousands of years ago. People primarily used dogs for guarding the hunters and areas of land.

Today's domestic dog is actually a subspecies of the gray wolf, a type of dog that is feared by most humans. Many people today, in all countries around the world, keep dogs as household pets and many even regard their dog as a family member.



Doesn't look good, right? Some of the descriptions were cut-off. Sure we can simply change the description Memo's height to fit largest text, but there are some disadvantages:

- paper wasting, since we will not use entire Memo's room for small text
- you newer know how long text will be or it can be changed in future
- it just doesn't look pretty :)

Go ahead and fix it: set Memo property "stretchMode" to "ActualHeight", set Detail property "stretchable" to "true" and generate report again.

### Capybara

The Capybara is a large, semi-aquatic rodent that is found inhabiting the water-logged regions of Central and South America. Closely related to other South American rodents such as Chinchillas and Guinea Pigs, the Capybara is the largest rodent in the world weighing up to 75kg and measuring nearly 1.4 meters long. Despite their enormous size though, these mammals have adapted well to life in the water and have a number of distinctive characteristics that aid their amphibious lifestyle, including the webbed skin between their toes which is particularly helpful when swimming. Interestingly enough, the common name of the Capybara is thought to mean "Master of the Grasses", whilst it's scientific name comes from the Greek word for water hog.



### Abyssinian

The Abyssinian Cat is thought to be one of the oldest breeds of domestic Cat in the world, as the first domestication of the Abyssinian Cat occurred in Ancient Egyptian times. It is thought that Abyssinian Cats were bought and sold on the banks of the River Nile by traders, where the African Wild Cats (the ancestors of all domestic Cats) lived in their native habitats. Abyssinian Cats are most easily identified by their "ticked" fur which gives their coat a mottled appearance.



### Adelie Penguin

The Adelie Penguin is the smallest and most widely distributed species of Penguin in the Southern Ocean and is one of only two species of Penguin found on the Antarctic mainland (the other being the much larger Emperor Penguin). The Adelie Penguin was named in 1840 by French explorer Jules Dumont d'Urville who named the Penguin for his wife, Adelie. Adelie Penguins have adapted well to life in the Antarctic as these migratory Birds winter in the northern pack-ice before returning south to the Antarctic coast for the warmer summer months.



### Baboon

The Baboon is a medium to large sized species of Old World Monkey that is found in a variety of different habitats throughout Africa and in parts of Arabia. There are five different species of Baboon which are the Olive Baboon, the Guinea Baboon, the Chacma Baboon, the Yellow Baboon and the Hamadryas Baboon which differs most from the others wide it's bright red face and cliff-dwelling lifestyle (the other four species are collectively known as Savanna Baboons). However, there is some debate over the classification of the different species due to the fact that some have been known to interbreed, indicating that they could be sub-species instead. Baboons are incredibly sociable and intelligent animals that are known to form close bonds with other members of the troop that often last for life. They are also incredibly adaptable animals but their population numbers are declining throughout their natural range primarily due to hunting and habitat loss.



### Camel

The Camel (also known as the Dromedary Camel, the Arabian Camel and the One-Humped Camel) is a large hoofed animal that is most commonly found in the hot deserts of Northern Africa and the Middle East. Thought to have been first domesticated by native people more than 5,000 years ago, these hardy animals have proved vital to the survival of humans in these areas as they are not just used for transporting both people and goods, but also provide a good source of milk, meat and wool. The Camel is one the most unique mammals on the planet and has adapted perfectly to life in the desert where food and water can often be scarce, and the temperature changes rapidly from the scorching-hot days to the cooler nights. However, although they would have once been found freely roaming the Arabian deserts, they are today extinct from the wild but the domestic population is widespread and numerous.



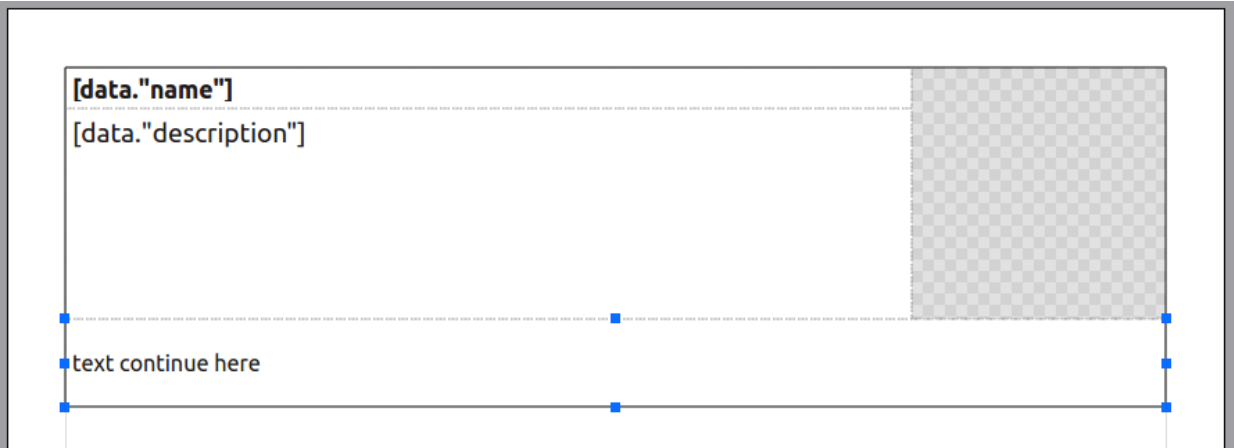
As you can see "stretchMode" do the work. There are all options:

- DontStretch - do not stretch the object
- ActualHeight - stretch the height of the object to fit all assigned text
- MaxHeight - stretch the height of the object to reach bottom of the band

# Text wrap of objects

*(commercial version only)*

Sometimes you might want to develop report design that requires text wrapping around other objects. It can be Image or table. This is simple challenge with CuteReport. Simply add one new Memos where text should flow to. For the second Memo set property "flowTo" with the name of the first Memo where is text begin. For example if first Memo has name "memo\_1", set second Memo property "flowTo" to "memo\_1". Property "StretchMode" for the first Memo should be set to "DontStretch" and for the second one to "ActualHeight". That's it.



Now lets render this template and see how it looks:

### Capybara

The Capybara is a large, semi-aquatic rodent that is found inhabiting the water-logged regions of Central and South America. Closely related to other South American rodents such as Chinchillas and Guinea Pigs, the Capybara is the largest rodent in the world weighing up to 75kg and measuring nearly 1.4 meters long. Despite their enormous size though, these mammals have adapted well to life in the water and have a number of distinctive characteristics that aid their amphibious lifestyle, including the webbed skin between their toes which is particularly helpful when swimming. Interestingly enough, the common name of the Capybara is thought to mean "Master of the Grasses", whilst it's scientific name comes from the Greek word for water hog.



### Abyssinian

The Abyssinian Cat is thought to be one of the oldest breeds of domestic Cat in the world, as the first domestication of the Abyssinian Cat occurred in Ancient Egyptian times. It is thought that Abyssinian Cats were bought and sold on the banks of the River Nile by traders, where the African Wild Cats (the ancestors of all domestic Cats) lived in their native habitats. Abyssinian Cats are most easily identified by their "ticked" fur which gives their coat a mottled appearance.



### Adelie Penguin

The Adelie Penguin is the smallest and most widely distributed species of Penguin in the Southern Ocean and is one of only two species of Penguin found on the Antarctic mainland (the other being the much larger Emperor Penguin). The Adelie Penguin was named in 1840 by French explorer Jules Dumont d'Urville who named the Penguin for his wife, Adelie. Adelie Penguins have adapted well to life in the Antarctic as these migratory Birds winter in the northern pack-ice before returning south to the Antarctic coast for the warmer summer months.



### Baboon

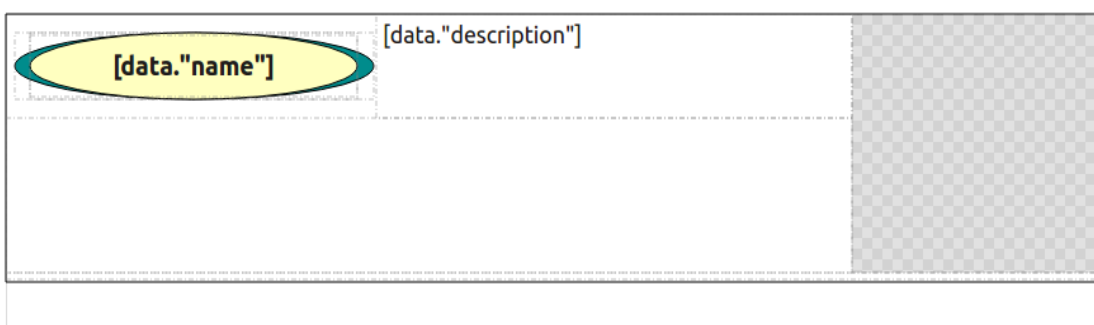
The Baboon is a medium to large sized species of Old World Monkey that is found in a variety of different habitats throughout Africa and in parts of Arabia. There are five different species of Baboon which are the Olive Baboon, the Guinea Baboon, the Chacma Baboon, the Yellow Baboon and the Hamadryas Baboon which differs most from the others wide it's bright red face and cliff-dwelling lifestyle (the other four species are collectively known as Savanna Baboons). However, there is some debate over the classification of the different species due to the fact that some have been known to interbreed, indicating that they could be sub-species instead. Baboons are incredibly sociable and intelligent animals that are known to form close bonds with other members of the troop that often last for life. They are also incredibly adaptable animals but their population numbers are declining throughout their natural range primarily due to hunting and habitat loss.



For this example we have set **textFlag** property to **AlignJusify**. It is not necessary to set text flags to all Memos, so set them only for the first one. Every subsequent Memo inherits this setting.






## Complex wrapping

Making complex wrapping is not really much complicated. Take a look at the next example:





As you may notice there are 3 Memo objects that used to fit text: first in the middle top contains text "[data."description"]". Second one is laying under the Title memo and covers left and central part. Third one is located on the bottom under all other objects. Its height is set to minimal, since for some short texts it will not be used. So we do not need space wasting. Every next Memo joined to the previous one by setting property **"flowTo"**. The property **stretchMode** of the last item is set to "ActualHeight". First two items do no need to stretch, so theirs property is set to "DontStretch". Press **"F5"** to render and voilà:

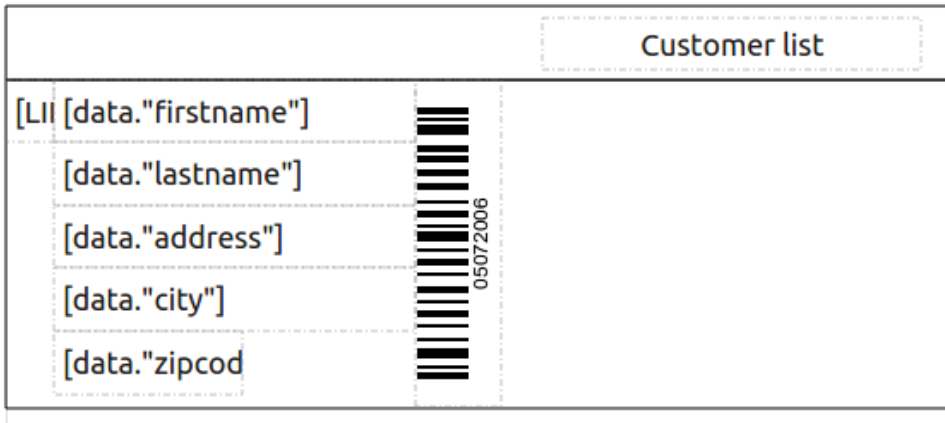
<p><b>Capybara</b></p>	<p>The Capybara is a large, semi-aquatic rodent that is found inhabiting the water-logged regions of Central and South America. Closely related to other South American rodents such as Chinchillas and Guinea Pigs, the Capybara is the largest rodent in the world weighing up to 75kg and measuring nearly 1.4 meters long. Despite their enormous size though, these mammals have adapted well to life in the water and have a number of distinctive characteristics that aid their amphibious lifestyle, including the webbed skin between their toes which is particularly helpful when swimming. Interestingly enough, the common name of the Capybara is thought to mean "Master of the Grasses", whilst it's scientific name comes from the Greek word for water hog.</p>	
<p><b>Abyssinian</b></p>	<p>The Abyssinian Cat is thought to be one of the oldest breeds of domestic Cat in the world, as the first domestication of the Abyssinian Cat occurred in Ancient Egyptian times. It is thought that Abyssinian Cats were bought and sold on the banks of the River Nile by traders, where the African Wild Cats (the ancestors of all domestic Cats) lived in their native habitats. Abyssinian Cats are most easily identified by their "ticked" fur which gives their coat a mottled appearance.</p>	
<p><b>Adelie Penguin</b></p>	<p>The Adelie Penguin is the smallest and most widely distributed species of Penguin in the Southern Ocean and is one of only two species of Penguin found on the Antarctic mainland (the other being the much larger Emperor Penguin). The Adelie Penguin was named in 1840 by French explorer Jules Dumont d'Urville who named the Penguin for his wife, Adelie. Adelie Penguins have adapted well to life in the Antarctic as these migratory Birds winter in the northern pack-ice before returning south to the Antarctic coast for the warmer summer months.</p>	
<p><b>Baboon</b></p>	<p>The Baboon is a medium to large sized species of Old World Monkey that is found in a variety of different habitats throughout Africa and in parts of Arabia. There are five different species of Baboon which are the Olive Baboon, the Guinea Baboon, the Chacma Baboon, the Yellow Baboon and the Hamadryas Baboon which differs most from the others wide it's bright red face and cliff-dwelling lifestyle (the other four species are collectively known as Savanna Baboons). However, there is some debate over the classification of the different species due to the fact that some have been known to interbreed, indicating that they could be sub-species instead. Baboons are incredibly sociable and intelligent animals that are known to form close bonds with other members of the troop that often last for life. They are also incredibly adaptable animals but their population numbers are declining throughout their natural range primarily due to hunting and habitat loss.</p>	
<p><b>Camel</b></p>	<p>The Camel (also known as the Dromedary Camel, the Arabian Camel and the One-Humped Camel) is a large hoofed animal that is most commonly found in the hot deserts of Northern Africa and the Middle East. Thought to have been first domesticated by native people more than 5,000 years ago, these hardy animals have proved vital to the survival of humans in these areas as they are not just used for transporting both people and goods, but also provide a good source of milk, meat and wool. The Camel is one the most unique mammals on the planet and has adapted perfectly to life in the desert where food and water can often be scarce, and the temperature changes rapidly from the scorching-hot days to the cooler nights. However, although they would have once been found freely roaming the Arabian deserts, they are today extinct from the wild but the domestic population is widespread and numerous.</p>	

You do not have to care about object insertion order, you can add Memo objects to report page in arbitrary order. Once you set correct "flowTo" name all is done. CuteReport is smart enough to understand what do you want and it hides all routine work from your sight. Enjoy!

# Label printing

(commercial version only)

In this chapter we will see how to create report with columns using CuteReport. That can be useful to print labels or so. Let's create a simple report containing customer labels to print out on customer case folders. Below you can see this example:



And after rendering we have the following:



As you can see there is a lot of wasted space on the right side, therefore a lot of wasted paper. To optimize the space we will set a number of columns that will fit all our labels. It can be done using **"columns" page** property. Set it to "3" in the Property Editor on the right side of the screen. Then press **"F5"** to generate report.

Customer list		
1 Ximena Garrett 2278 Second Street Paramus 7843	7 Carolyn London 990 Hill Street Metaline Falls 67901	13 Waylon Hardison 1078 Fourth South Harmon 86432
2 Jaunita Johnson 5505 Park Boulevard Drummond 73124	8 Tia Bergman 4327 Cedar Avenue Jacksonville 76320	14 Ankti Vanwinkie 3956 Eighth Circle Byron 94321
3 Calista Baskin 2017 Seventh Street Boca Raton 94765	9 Brady James 875 Lake Circle Stockholm 98347	
4 Tyra	10 Alix	

There are 2 types of column filling: **"Vertical"** and **"Horizontal"** that can be set via **"fillDirection"** Page property. On the picture above you can see "Vertical" type that means any next label will be printed under the previous one and so on while there is empty space exist in the column. When there is no space, report will create a new column and start from the top. "Horizontal" type means every next label will be printed on the right of the previous one while there is enough space on the right side of the page. If there is no space, report will print next label on the next row as you can see below:



Customer list		
1 Ximena Garrett 2278 Second Street Paramus 7843	2 Jaunita Johnson 5505 Park Boulevard Drummond 73124	3 Calista Baskin 2017 Seventh Street Boca Raton 94765
4 Tyra Marcotte 6745 Fifth Circle Springboro 54322	5 Steve Neeley 793 Main Street Edward 76348	6 Vida Brock 395 Sixth Circle Wrightsboro 89372
7 Carolyn London 990 Hill Street Metaline Falls 67901	8 Tia Bergman 4327 Cedar Avenue Jacksonville 76320	9 Brady James 875 Lake Circle Stockholm 98347
10 Alix	11 Andreas	12 Elise

You can set any type depending of your needs. Not all bands respect column setting. Some ignore it, like PageHeader, PageFooter. Some other have special property to adjust this behavior, like DetailHeader or DetailFooter. Using this option you can design complex columned reports. One of the samples of columned report with grouping you can see below:

My Music Collection					
Title			lenght		
Chris Norman					
3	Sarah (You Take My Breath Away)	342	4	I Want To Be Needed (Duet With)	633
5	Never Make My Angel Cry	345	6	Hunters Of The Night	345
7	As Good As It Gets	532	8	Every Little Thing	432
9	Red Hot Screaming Love	342	10	Still In Love With You	243
11	Jealous Heart	342	12	Stupid Girl	342
Chris Norman tracks: 12 min: 243 max: 633 avg: 399.00 Total length: 4794					
Richard Clayderman					
1	Gimme! Gimme! Gimme!	356	2	The Winner Takes It All	489
3	Chiquitita	683	4	Fernando	386
5	Mamma Mia	496	6	Piano Concerto No.1 In B-Flat	453
7	Elvira Madigan- Piano Concerto	533	8	Cornish Rhapsody	522
9	Liebestraum (Reves D'Amour)	754	10	La Pathetique	321
11	Fur Elise	643	12	Arabesque	643
13	Aquarela	254	14	Ballade pour Adeline	456
15	Para Elisa	564	16	Yesterday	564
17	Just the way you are	335	18	Just the way you are	467
19	Ne me quitte pas	624			
Richard Clayderman tracks: 19 min: 254 max: 754 avg: 502.68 Total length: 9543					
Roxette					
1	Spending My Time	342	2	Crash! Boom! Bang!	234
3	Run To You	234	4	It Must Have Been Love	533
5	I'm Sorry	264	6	Almighty 7 Mix	245
7	Almighty 12 Definitive Mix	269	8	Almighty Alternate 12	439
9	X-Treme Extended Mix	298	10	Opportunity Nox	659
11	The Look	342	12	Dressed For Success	432
13	Dangerous	532	14	Joyride	432
my music collection			page 2 of 0		



## Multi-page report

It is possible to create several design pages in CuteReport. This feature is useful if report should contain different pages with different sizes, orientations, etc. In this case report engine will fully render first page and then second and so on. Total number of template pages is not limited. Let us look at the simple example with 2 pages where the first one is title page and the second one is report itself. We will use our previous example "Customer List". To add new page click on the button  on the Page Editor's toolbar. If there are some types of page you will see drop-down menu. Then choose page type you want by clicking it. New page will be added to the report. Move it to the first place by clicking . Since item cannot be placed direct on page, we will add **Overlay** band to the middle of the page. Now add Memo item to the band and enter text "Customer Report" in there. Render report and now it should look like:

Customer report					
Customer list					
1	Ximena	Garrett	2278 Second Street	Paramus	7843
2	Jaunita	Johnson	5505 Park Boulevard	Drummond	73124
3	Callista	Baskin	2017 Seventh Street	Boca Raton	94765
4	Tyra	Marcotte	6745 Fifth Circle	Springboro	54322
5	Steve	Neeley	793 Main Street	Edward	76348
6	Vida	Brock	395 Sixth Circle	Wrightsboro	89372
7	Carolyn	London	990 Hill Street	Metairie Falls	67901
8	Tia	Bergman	4327 Cedar Avenue	Jacksonville	76320
9	Brady	James	875 Lake Circle	Stockholm	98347
10	Alla	Bohannon	5306 maple Avenue	Selling	75349
11	Andeana	Pratt	1096 Eighth Boulevard	Des Moines	10345
12	Ellens	Vanmeter	403 Elm Circle	Orchard	80632
13	Waylon	Hardison	1078 Fourth South Boulevard	Harmon	86432
14	Ankti	Vanwinkle	3956 Eighth Circle	Byron	94321

# Script Engine

In this chapter we will learn how to work with CuteReport script engine. Script is a program written on a high level programming language and this program is executed in report rendering time. Scripting feature brings an extremely high level of flexibility. Using script user can control almost every rendering step and design really complex reports. There is a main script in any report that controls everything from report starting till it's rendered. Some items like Memo or Barcode support in-line script in their text properties. Usually in-line scripting expression have to be framed by [], so the scripting engine knows that this is a script and not a regular text. But to some fields where only script expressions is allowed it can be written without []. Some items can reimplement "[" to use something else instead of square brackets or provide additional field to define in-line script expression borders, like Memo item does by its "expDelimiter" property.

Script is applicable in a wide range cases. Using script you can do following:

- perform a data processing with a complex logic that cannot be accomplished by standard functionality of CuteReport engine;
- control the properties of pages, bands, items changing their position, size, appearance and data.

To see report script, switch to the "Script" tab on the Modules bar on the left side of CuteReport Designer. Refer to the chapter "Designer → Script Editor" for more information about Script Editor.

## Script objects

All report objects are accessible from script by unique object name. For example, if you have Memo object with name memo\_1 and you want to change its color, you can do it by the following way:

```
memo_1.backgroundBrush = new QBrush(new QColor("#665544"));  
memo_1.color = new QColor(Qt.red);
```

All object properties you see in Property Editor can be managed via script. Some objects may have two or more properties with the same name, like property "stretchMode" of Memo element. There are 2 types: enum and string.

You can use any of them like:

```
memo_1.stretchMode = "DownStretch";  
memo_2.stretchMode = Memo.ActualHeight;
```

## Script variables

Any object or value is stored in script as variable. You can create your own custom variable on the start of report processing and then change its value while report processing. Script Engine may have its own internal variables that user can access to.

### Local variables

Variables can be declared and used in script locally. Once variable declared it can acquire any arbitrary value. Here is an example:

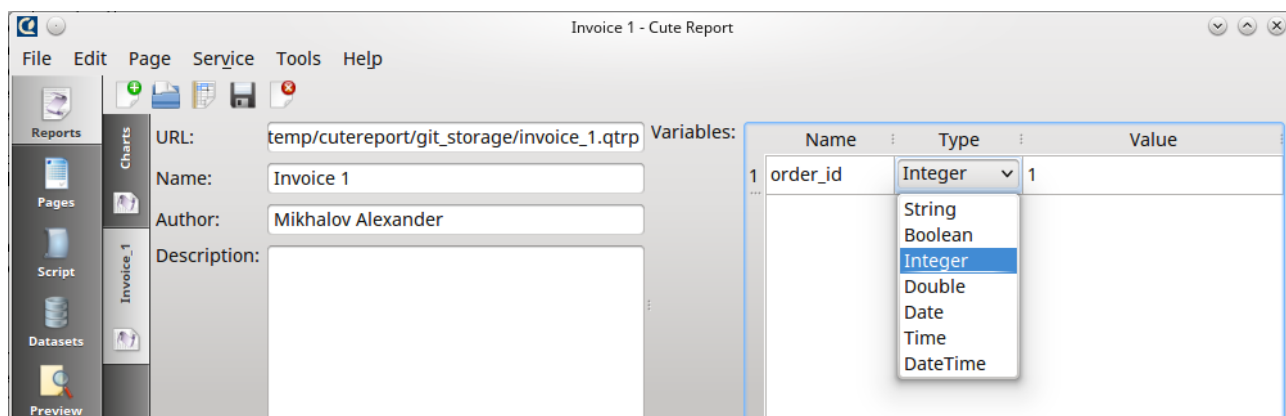
```
var myVar = "Hello World!";
```

When you have declared variable you can use it in any report's object, for example, Mome, typing "[myVar]" in property **"text"**. For more detailed information regarding JavaScript variables refer to JavaScript documentation.

### Global variables

Any global variables declared in report can be accessible from script expression. Other name of global variables is parameters. Variable name should be declared as: `${my_variable}` without spaces in variable name within brackets. It is recommended to use two variants of variable naming if the name contains some words, like "my super duper variable". First variant is to replace spaces with underscore sign "\_", like "my\_super\_duper\_variable". Second variant is to remove spaces and use first capital letter on any word after the first one like "mySuperDuperVariable". Internally Script Engine uses special representation of the global variables. So using "\_" in the begin of variable name highly not recommended.

Once variable declared it will be automatically added to the list of global variables. Let us demonstrate it. Open new report, go to "Script" tab and type "\${test}" in there. Now switch to the "Reports" tab. You will see new variable appears in the list of reports global. Now you test your report by assigning any value to this variable.



Some variable types available for external variables (parameters): String, Boolean, Integer, Double, Date, Time, DateTime. In case variable is passed from external program, its type will be automatically casted to one of the specified type.

## Renderer variables

Renderer engine has its own variables and full variable list depends only of a renderer module itself.

There are some of the variable available for the standard renderer:

- **LINE** — current dataset line number starting from 1
- **LINES** — total lines
- **PAGE** — current page number starting from 1

Professional version of render has some additional variables:

- **PAGES** — total pages (requires 2 passes report processing)
- **PASS** — current report pass number
- **PASSES** — total report passes
- **DATE** — string representing current date formatted regarding application locale settings (professional version only). Function `QDate.currentDate()` can be used instead of this variable
- **TIME** — current time (professional version only). Function `QTime.currentTime()` can be used instead of this variable)
- **DATETIME** — datetime object that contains current date and time and can be formatted to any representation using formatting tag

# Forms

*(professional version only)*

In this chapter we will review one example how to manipulate a form by the script. Working example you could be able to find in CuteReport Demo application in “General → forms” report.

Here is the full script content:

```
var reportDefaultLang = engine.reportVariableValue('tr');
var translator_codes = engine.customData('translator_codes');
var translator_languages = engine.customData('translator_languages');

print('def lang: ' + reportDefaultLang);
print('codes: ' + translator_codes);
print('languages: ' + translator_languages);

form.defaultLangLabel.setText(reportDefaultLang.length == 0 ? 'Not Defined' :
reportDefaultLang);

form.languages.addItem('Default');
form.languages.addItems(translator_languages);

if (form.exec()) {
    var index = form.languages.currentIndex;
    var newLang = translator_codes[index - 1];
    print("newLang = " + newLang);
    engine.setReportVariableValue('tr', newLang);
    engine.run();
}
```

In the first 3 lines we get default language set in report template if it is set, language codes and full languages names.

```
var reportDefaultLang = engine.reportVariableValue('tr');
var translator_codes = engine.customData('translator_codes');
var translator_languages = engine.customData('translator_languages');
```

All these 3 variables belong to Translator module and will not be available if you don't have the commercial version installed. Next 3 lines is here to print these values to script debug output for debugging purposes.

```
print('def lang: ' + reportDefaultLang);
print('codes: ' + translator_codes);
print('languages: ' + translator_languages);
```

You can see these values if press red/green arrow in the Designer status bar at the right bottom corner.

Next line sets required value to the label that is placed on the form and named “defaultLangLabel”. It is general Qt widget QLabel. So all methods that available for it is accessible within the script. All available method for a QLabel widget you can find on <http://doc.qt.io/qt-4.8/qlabel.html>. On that web site you can find a documentation for all Qt widgets and theirs methods.

Next 2 lines is a filling language combobox with the data fetched from Translator internal data.

```
form.languages.addItem('Default');  
form.languages.addItems(translator_languages);
```

In the first line we add “Default” language. This is required because Translator contains only translated data, but not the original report item's text. Generally if report variable is set to empty, i.e. “” then no translation will be performed. For our example it means that report will be generated using default report data on English.

Next code performs form appearing:

```
if (form.exec()) {
```

Keep in mind that form that is QDialog widget has another method “show”, but it doesn't block further script processing while form is displayed. This is not behaviour you want to see. You want form to accept user input and then continue. So always use “exec” method, but not “show”.

Next code block will be processed if user pressed “Ok” button. If user pressed “Cancel” then this block will be skipped and report generation not be performed.

```
var index = form.languages.currentIndex;  
var newLang = translator_codes[index - 1];  
print("newLang = " + newLang);  
engine.setReportVariableValue('tr', newLang);  
engine.run();
```

Here we check combobox with name “languages” for its current selected row number (aka index). Next we take language code from previously saved list of all language codes used in report. We save this code to the “tr” variable and then run report by calling “*engine.run()*”. Should be noted that if script does not have any code then “*engine.run()*” will be added automatically. Once you have any code in the script you must call this method by yourself. Setting “tr” variable instruct the translator to translate all translatable strings to the language selected. Bear in mind that changing translation doesn't change locale of the report. Generally it means you will have date representation in your default computer locale format. So change report locale as well, use “*engine.setLocaleName()*” method along with “*engine.setReportVariableValue()*”.

# Functions

## Spelling out

*(professional version only)*

To use spelling functions you need to have “ScriptSpellout” module on place. This module contains all required functionality to spell out numbers, currencies and money values. Unlike other reporting solutions CuteReport is not limited by a couple or even few processing languages, but rather it has functionality to spell out on the most of the world languages. The module can correctly process values taking into account their plurality (6 forms) and gender (masculine, feminine, neuter) on a selected language. Although gender is not important to translate values for English, it is very important for a lot of other languages where spelling for different genders differs.

Before having a good look at these functions, let us consider the basic concepts that will help to understand them.

*Locale.* The locale is a string of the form «en\_US», where the first value is a two character code of the language, and the second one is a country code. So for American English it looks like «en\_US», for British English it looks like «en\_EN». The country code can be omitted along with preceding underscore symbol.

*Plurality.* There are 6 basic plural forms: zero, one, two, few, many, other. Only few Asian languages support all 6 forms. Most of the languages support only 2 or 3 forms. For example, English has only 2 plural forms: one (dollar), other(dollars). Ukrainian has 3 plural forms: one, few, other.

*Gender.* There are 3 forms of gender: male, female, neuter. On some languages the same object can be by different gender. Therefore, some functions require explicitly passed parameter to specify the gender of the spelled value. For some other languages spelling is the same for all genders. For example, in English: “*fifty-one* dollar” or “*fifty-one* women”. In first case gender is masculine, in the second case it is feminine. But number spelling still the same “fifty-one”. For Ukrainian language spelling in these cases will be different “двадцять *один* доллар”, “двадцять *одна* жінка” or “двадцять *одне* значення”. As you can see knowing gender is very important sometimes. If gender is not specified, the default masculine is used. As a parameter indicating the masculine, following values can be passed: “male”, “masculine”, “m”, “he”. As a parameter indicating the feminine, following values can be passed: “female”, “feminine”, “f”, “she”. For neuter: “neuter”, “n”, “it”.

**numberSpell( *number*, *locale*, *gender* )** — function to spell out of numbers. As function parameters it takes 3 values: number to be converted, locale, gender. Gender can be omitted. Masculine is used by default in this case. If locale parameter passes is empty then default report locale will be used.

Examples:

numberSpell(123002, 'en') → one hundred twenty-tree thousand two

numberSpell(123001, 'uk', 'f') → сто двадцять три тисячі одна



numberSpell(123002, 'es') → ciento veintitrés mil dos

numberSpell(2341, 'pl') → dwa tysiące trzysta czterdzieści dwa

numberSpell(2341, 'gm', 'n') → zweitausenddreihundertzweiundvierzi

**currencySpell( *currencyISOcode*, *locale*, *plural/number*, *cutWord* )** - function to spell full currency name using the currency ISO code. As function parameters it takes 4 values: currency ISO code, locale, plurality indicator, word cutting index, number to be converted, locale, gender. ISO code of currency is a three character code. For example, for USA currency it is "USD", for Canada currency it is "CAD", for Ukraine it is "UAH", for Georgia it is "GEK". You can find all the codes on the WiKiPedia page: [https://en.wikipedia.org/wiki/ISO\\_4217](https://en.wikipedia.org/wiki/ISO_4217). As a plurality indicator a plurality key word or a number can be passed. The keyword can be one of the following: zero, one, two, few, many, other. Plurality will be detected automatically if a number is passed as a plurality indicator. The last parameter "cutWord" determines which word in the full currency ISO name will be printed out. For example, full currency name for "USD" is "US dollar", for "UAH" it is "Ukrainian hryvnia". But very often in local documents you might want to use local shortened currency name instead of full ISO name. For example, in USA, perhaps, you will use just "dollar" instead of "US dollar", in Ukraine "hryvnia" instead of "Ukrainian hryvnia" and so on. To print local currency representation you need to cut out unnecessary words and leave only local currency representation. For example, if you set cutWord to 2 for USD, the only second word will be printed out - "dollar". For Ukraine if you pass 2 you will get "hryvnia". Simple, right?

Examples:

currencySpell('USD', 'en', 2342) → US dollars

currencySpell('USD', 'en', 2342, 2) → dollars

currencySpell('UAH', 'en', 1, 2) → hryvnia

currencySpell('USD', 'en', 'many', 2) → dollars

currencySpell('USD', 'en', 'one', 2) → dollar

**moneySpell( *number*, *currencyISOcode*, *locale*, *gender*, *cutWord* )** - function to spell amount of money. Mean of all parameters like in functions above.

Examples:

moneySpell(2342, 'USD', 'en', '', 2) → two thousand three hundred forty-two dollars

moneySpell(1, 'USD', 'en') → one US dollar

moneySpell(1, 'USD', 'en', '', 2) → one dollar

**pluralSpell( *number*, *pluralityArrayString* )** — function to spell out a specified word in a required plurality. As a plurality form sign the number is passed in the function parameters. To choose a correct word for the passed number there is another parameter. It passes a string that contains pairs. Each pair represents plurality form and required word in this plurality form.

The string is passed in the following format: *'one{apple} other{apples}'*. Each language has its own set of plurality forms. English has only 2: one and other. Other languages has more forms, up to 6. If a required plurality form is not found in the array passed, then “other” form will be used. You can have a plurality array per each report languages. Use translator module to perform this.

Examples:

`pluralSpell(13, 'one{apple} other{apples}')` → apples

`pluralSpell(1, 'one{apple} other{apples}')` → apple

`pluralSpell(0, 'one{apple} other{apples}')` → apples

# Formatting

*(professional version only)*

Any expression calculated by Script Engine can be formatted for better appearance. To do that followed format is used:

[expression #tag]

where “expression” is scripting expression, and “tag” is formatting tag. Tag should be located after the expression and it is separated from the expression by space followed by # sign. No other symbol after the tag is acceptable.

Formatting tag contains of two parts separated by percent sign “%”: expression type and formatting options. Expression types and theirs codes can be:

- text — **s**
- numeric — **n**
- date — **d**
- boolean — **b**

Formatting options depend of the type.

## Formatting options for text type:

- (empty) — no formatting required;
- **l** — (low case) all text should be printed using low case letters. Example: original text “Text” → formatted text “text”;
- **u** — (upper case) all text should be printed using upper case letters. Example: original text “Text” → formatted text “TEXT”;
- **fu** — print first letter of any sentence in upper case. Example: original text “just a text” → formatted text “Just a text”;
- **s** — print text using sentence formatting, i.e. first letter is in upper case and period sign at the end. Example: original text “just a text. Additional text” → formatted text “Just a text. Additional text.”;

## Formatting options for numeric type in general look like:

[flag][number of integer digits][decimal separator][number of decimal digits][additional char formatting option]

Any component in numeric formatting can be skipped. “Flag” has only one effective value “+” and it directs to print positive sign for positive values besides negative sign with negative values.

Number of integer digits shows minimal number of digits in integer part of value. All

missing digits will be filled by zero.

Decimal separator shows which symbol will be used as decimal separator. The most often used symbol is dot "." and comma ",". To print standard separator for the current application locale setting use sign "?". Any other non numeric symbols can be used as well, for example "-".

Number of decimal digits determines minimal number of decimal digits after the decimal dot. If this number contains less decimal digits then missed digits will be filled by zero.

Additional char formatting option can be followed:

- f — fixed number of decimal digits. Any extra digits will be discarded.
- n — like above but with adding thousand group separator
- m — money formatting used for current application locale setting

Examples:

[12345. 6 #n%7] → «0012345.6»

[12345. 6 #n%+7] → «+0012345.6»

[12345. 6 #n%4.2] → «12345.60»

[12345. 6 #n%4-2] → «12345-60»

[12345. 6 #n%m] → «12345.60 грн.»

[12345. 6032 #n%4.2f] → «12345.60»

[12345. 6032 #n%4.2n] → «12,345.60»

### **Date and time formatting options:**

- text — includes day and month name, number of day in month and full year. Day and month names are printed in shortened form defined in current locale. It is equal to "ddd MMM d yyyy"
- short — print short date form
- long — print long date form
- iso — print data in ISO format. Either YYYY-MM-DD for dates or YYYY-MM-DDTHH:mm:ss, YYYY-MM-DDTHH:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00) for combined dates and times
- dd.mm.yyyy — date in format '23.12.2015'
- dd mmm yyyy — date in format '23 Nov 2015'
- dd mmmm yyyy — date in format '23 November 2015'
- hh:mm — time in format '23:12'
- hh:mm:ss — time in format '23:12:01'
- dd mmmm yyyy, hh:mm — date and time in format '23 November 2015, 23:12'



## Script Signals

When you write a script it means you write its main function, which is processed on the report generator start. In this function user can create some new variables, initialize them or do some other preparations. You still might want more control over report processing. To make it possible almost every single report object has signals and you can assign your custom slot to any of these signals. For example, you can assign your custom filter to Detail band and hide some bands while pass another. In such way you can filter out unnecessary records. Lets review some signals and later will see how we can use them.

### Common item signals

Signal Name	Description
printInit	emitted when all items are preparing to be printed
printReset	emitted when all items are cleaned up after printing
printCreatingBefore	emitted before item printing. All property changes affects original template item
printDataBefore	emitted when initial data for printed item is prepared. All property changes affects only current printed item and will be reset
printBefore	emitted after all item's data is processed, but before actual printing
printAfter	emitted after item is printed on a page

Also any item can have its own signals. You can see full signal list along with signal description in a Property Editor.

### Renderer signals

Signal Name	Description
reportStart	emitted after report started
templatePageBefore (CuteReport::PageInterface * page);	emitted before template page processing
templatePageAfter (CuteReport::PageInterface * page);	emitted after template page [processing
pageBefore (CuteReport::RenderedPageInterface *)	emitted before rendered page processing
pageDataAfter (CuteReport::RenderedPageInterface *)	emitted after page data is processed
pageDesignAfter (CuteReport::RenderedPageInterface *)	emitted after page design elements is places. Design elements is such elements like: page headers and footers, background and foreground, watermarks ans so on.
pageAfter (CuteReport::RenderedPageInterface *)	emitted after rendered page processing
bandBefore (CuteReport::BandInterface *)	emitted before band rendering
bandDataAfter (CuteReport::BandInterface *)	emitted after band data is processed
bandLayoutBefore (CuteReport::BandInterface *)	emitted before placing child elements to the container and possible geometry changing

Signal Name	Description
<b>bandLayoutAfter</b> (CuteReport::BandInterface *)	emitted after placing child elements to the container. Band geometry will not be changed after that signal
<b>bandAfter</b> (CuteReport::BandInterface *)	emitted after band is rendered
<b>itemBefore</b> (CuteReport::BaseltemInterface *)	emitted before item rendering
<b>ItemDataAfter</b> (CuteReport::BaseltemInterface *)	emitted after item data is processed
<b>itemLayoutBefore</b> (CuteReport::BaseltemInterface *)	emitted before placing child elements to the item and placing the item on its parent element or band
<b>itemLayoutAfter</b> (CuteReport::BaseltemInterface *)	emitted after placing child elements to the item and placing the item on its parent element or band
<b>itemAfter</b> (CuteReport::BaseltemInterface *)	emitted after item is rendered
<b>datasetBefore</b> (CuteReport::DatasetInterface *)	emitted before dataset processing
<b>datasetAfter</b> (CuteReport::DatasetInterface *)	emitted after dataset processed
<b>datasetIteration</b> (CuteReport::DatasetInterface *)	emitted on every dataset iteration
<b>formBefore</b> (CuteReport::FormInterface *)	emitted before form is shown
<b>formAfter</b> (CuteReport::FormInterface *)	emitted after form is closed
<b>reportDone</b>	emitted after report rendering is done

Lets review a simple scenario of report building. Report contains only one page, container and placed on it item. After start rendering process a signal "reportStart" of object "engine" will be emitted. Before first page processing engine will emit signal "templatePageBefore". This signal is emitted only once before any template page processing. Should be mentioned that you need differentiate template report page and rendered report page. For any single template page some rendered report page and be produced.

After that data of the page template will be processed.

Further processing goes in following sequence:

1. signal "requestNewPage" is emitted that gives and ability to do some actions before rendered page is produced
2. signal "pageBefore" is emitted. On this stage preparing of the page data is completed, but bands and items are not precessed yet
3. signal "bandBefore" is emitted just before band processing
4. signal "itemBefore" is emitted for the item placed on the band. If item has its children items, then for each of them this signal will be emitted
5. signal "itemDataBefore" is emitted for all items and bands

6. data of all band and elements is processing
7. signal "itemDataAfter" is emitted
8. signal "layoutBefore" is emitted for all elements
9. objects are placed on band
10. signal "layoutAfter" is emitted for all elements
11. signal "itemAfter" is emitted for all items located on the band precessed
12. signal "bandAfter" is emitted for the container

It was mentioned before that all items and bands have theirs own signals. You might want to use them in most cases.

To handle signal you need to connect your custom slot function like it is shown below:

```
memoitem.printDataBefore.connect(memo_test);
print('_Start_');

function memo_test() {
    memoitem.text = data.getValue("test name");
    var max = 255;
    var min = 0;
    var r = Math.floor(Math.random()*(max-min+1)+min);
    var g = Math.floor(Math.random()*(max-min+1)+min);
    var b = Math.floor(Math.random()*(max-min+1)+min);
    memoitem.backgroundBrush = new QBrush(new QColor(r,g,b));
}
```

In this example we are assigning random color for every new item with name "memoitem". You can see syntax the signal-slot connecting on the first line of the example.

Syntax in general is:

`object_name.signal_name.connect(slot_name)`

First two rows in the example above belong to the main script function, so they will be processed just after report processing start. Function "memo\_test" will be processed just after signal "printDataBefore" of the object "memoitem" emitting.



# Using in custom application

In this chapter you will learn how to use CuteReport in your custom application.

## Project setting up

There are 2 possible ways to use CuteReport with your custom application: as a standalone framework or as an embedded framework. Let's look close to the both ways.

### Embedded library

There are some important steps to use CuteReport as embedded library in the custom application:

- add all necessary data to your project file (.pro);
- add header files of CuteReport to your cpp file;
- create and initialize report core;

Add next lines to your .pro file:

```
INCLUDEPATH += path_to_CuteReport_headers
DEPENDPATH += $$INCLUDEPATH

LIBS += -Lpath_to_cutereport_shared_files -lCuteReport -lCuteReportWidgets
```

And add above headers to your code:

```
#include "reportcore.h"
#include "reportinterface.h"
```

### Standalone framework

To use CuteReport as standalone framework you can simply install CuteReport using installer provided. There are some advantages using it in this way:

- if you have some application installed that use CuteReport you should not update it for every application. Update CuteReport and all application will use new version.
- you can use official CuteReport repositories and keep CuteReport up to date automatically for Linux distributions.

To connect CuteReport to your application add to your pro file something like that:

```
!include( path_to_cutereport_include_directory/CuteReport.pri ) {
    error( Cannot find the CuteReport.pri file! )
}
```

Path to CuteReport.pri is dependent of your distribution.

There is default path for most installation:

Linux: /usr/include/cutereport/CuteReport.pri

Windows: c:/Program Files/CuteReport/dev/include/CuteReport.pri

Second step is to include header file to your C++ file:

```
#include <CuteReport>
```

## Simple example

Next is creating `CuteReport::ReportCore` instance and initializing it:

```
CuteReport::ReportCore * reportCore = new CuteReport::ReportCore();
```

There are some parameters you can pass to the constructor:

- *parent*: sets parent object to the `CuteReport` instance. If it is set you should not care about `CuteReport` instance deletion;
- *settings*: pointer to a `QSettings` object. You can use your project's settings to allow `CuteReport` to save its settings and states to the file using `[CuteReport]` group. The settings in this file can provide some initial info to configure `CuteReport`. Instead of writing a lot of code to add instances like storage objects, renderer objects, printer objects to the report, you can use custom settings. We will review these settings later. If `QSettings` pointer is not specified `CuteReport` will create its own ini file.
- *interactive*: used to specify if report objects are static or can be changed. If you develop some console application that process report templates without changing report objects you can free some resources by using "false". It is true by default;
- *initLogSystem*: determines if you need or no to see `CuteReport`'s logs. It is True by default. It should be mentioned that log destinations and log levels can be configured separately.

Additionally you can connect some signals to detect report exporting and rendering:

```
connect(reportCore, SIGNAL(exportDone(QString,bool)), this, SLOT(slotExportDone(QString,bool)));  
connect(reportCore, SIGNAL(printingDone(QString,bool)), this, SLOT(slotPrintDone(QString,bool)));
```

Now when we have core created we can load a report template:

```
CuteReport::ReportInterface * report = reportCore->loadReport("file:/path/myreport.qtrp");
```

In most cases you might want to see report preview, so lets create preview widget:

```
CuteReport::ReportPreview * preview = new CuteReport::ReportPreview(reportCore);  
preview->connectReport(report);
```

By passing report pointer to the preview widget you specify what report object preview should be represented.

Now we can start report rendering. There are some ways to do this. First way is pressing button "Run" in the preview widget. Second way is to invoke `CuteReport` Core method *render(report)*.

```
reportCore->render(report);
```

Third way is to invoke Preview method *run()*. You can choose any way.

```
preview->run();
```

Choose any way you like.

## Custom application example

Now let's do some coding. To add CuteReport library to your application you can do something like this:

```
#include "reportcore.h"

/* create report core instance */
CuteReport::ReportCore * reportCore = new CuteReport::ReportCore(0,0, false);

/* create report preview widget */
CuteReport::ReportPreview * preview = new CuteReport::ReportPreview(parentWidget);

/* assign report core to our preview */


```
preview->setReportCore(reportCore);
```



/* loading report template from file and creating of report object */
CuteReport::ReportInterface * reportObject = reportCore->loadReport("git:report.qtrp");

/* connect created report object to the preview */
preview->connectReport(reportObject);

/* show preview widget */
preview->show();

/* start report rendering */
preview->run();
```

Usually you need only one ReportCore instance in your application. Any number of Preview widget can be assigned to the core.

**Datasets**

## Model Dataset

Model Dataset is designed to print model's data (inherited from QAbstractItemModel) from an application. For printing data some steps are necessary:

- Create Model Dataset in your report;
- In the field "Model name" type in any name for your model. If there are some, they should be different;
- For the report testing you can fill test model with the data using any number of columns and rows;

All set. For printing your report you have to pass your model address (as longlong) to the report using report parameters.

For example:

```
CuteReport::ReportCore * cuteReport = new CuteReport::ReportCore();

// load report
QString err;
CuteReport::ReportInterface * reportObject = cuteReport->loadReport("file:test.qtrp", &err);

// if error, exit with message
if (!reportObject) {
    QMessageBox::critical(this, "loadReport", err);
    return;
}

// making of the test model
QStringList list;
list << "11111" << "2222" << "333" << "44" << "5";

model = new QStringListModel();
model->setStringList(list);

// Warning!!! Link to model passed as quint64
// Set model name the same you have set in the ModelDataset before
reportObject->setVariableValue("model1", quint64(model));

// making report preview window
CuteReport::ReportPreview * preview = new CuteReport::ReportPreview(cuteReport);
if (reportObject) {
    // set core and set preloaded report
    preview->setReportCore(cuteReport);
    preview->connectReport(reportObject);
    // report processing
    preview->run();

    //Preview window show
    preview->show();
}
```



While rendering the data model will be cloned, since `QAbstractItemModel` is not thread safe.

# Setting of required translation

TODO