**QASreport ver.1.01**

**C++ QT library from ver.5.**

1. **Description**
2. **Features**
3. **Installation and Use**
4. **Description of class: QASReport**
5. **Report** **Designer**
   1. **Interface**
   2. **Keyboard, mouse**
   3. **Visual designer objects**
   4. **Bands**
6. **General Workflow**

**1. Description**

QASreport - a library that contains a set of classes for building reports. It is a mix of designer and report generator output means.

The author does not guarantee the correct operation of the component and is not responsible for any damages resulting from the use of this software. All comments and suggestions to the address below.

**2. Features**

* Report generator based on the band;
* Built-in designer, available in run-time, with the ability to work like a usual graphic editor;
* The conclusion of the report results in the Preview;
* Any number of pages in the report;
* Multiple reports, the possibility of drawing up a number of reports and forms in a single file;
* Save the report file format - XML;
* Master-detail reports;
* Full control over the printing process, support of any type of paper and printing devices;
* Easy to design a report using simple components: a text box, line, image, shape, and bar-code;
* The output data of any database, which the driver is installed;
* The output data from any SQL model, QSqlQueryModel inheritor;
* Data output from any of the standard model, the inheritable QAbstractItemModel;
* Export reports to PDF files;
* Support for Qt Script (in the future);
* others.

**3.** **Installation and Use**

To compile the library and demo program, run on a compilation of the group project file: asreport.pro.

After compiling the directory: build/debug-release will contain the shared library: asreportlib, file executable file demo program: asreportdemo, form files, and other files.

To use linking library file: asreportlib. To the project folder copy a dynamic link library file: asreportlib.dll (win).

By the project also need to connect the modules: sql xml printsupport widgets.

The base class for working with the library - QASReport.

Connecting the class file QASReport:

#include ".. /asreportlib/asreport.h "

//create an instance of the class QASReport.

m\_Report = new QASReport(this);

//Adding to the report data sources - for output model

m\_Report ->addModel("sqldatamodel ",m\_sqldatamodel);

m\_Report ->addModel("stringsmodel ", m\_stringsmodel);

m\_Report ->addModel("itemsmodel ", m\_itemsmodel);

//Load file report

QString filename = QDir::currentPath() + "/forms/list.xml";

if(!QFile::exists(filename)) return;

QFile file(filename);

if (!file.open(QIODevice::ReadOnly))

return;

bool res = m\_Report->load(&file);

//Output report on preview

m\_Report->reportView();

//Print

Report->reportPrint();//Designing

m\_Report->edit(filereport);

//If necessary, signal processing is initialized:

// Begin print

connect(m\_Report,SIGNAL(signalBeginReport()),this,SLOT(slotBeginPrint()));

//processing variables of custom reports

connect(m\_Report,SIGNAL(signalGetValue(QString,QVariant&)),this,

SLOT(slotGetValue(QString,QVariant&)));

// processing the output end of the print output

connect(m\_Report,SIGNAL(signalEndReport ()),this,SLOT(slotEndReport()));

// Process output in the transition to the next record output tabular data

//sending a pointer to the output model

connect(m\_Report,SIGNAL(signalModelOnNextRec(QObject \*),this,SLOT(slotModelOnNextRec (QObject \*) ));

// Process output when set to the first record of the table

// sending a pointer to the output model

connect(m\_Report,SIGNAL(signalModelOnFirstRec (QObject \*),this,SLOT(slotModelOnFirstRec (QObject \*) ));

// Process of forming a new page

// sending to the current page number

connect(m\_Report,SIGNAL(signalNewPage (QObject \*),this,SLOT(slotNewPage (QObject \*) ));

**4. Description of class: QASReport**

**QASReport** - is central to the main class for working with reports. Using it is loaded file report, call the report designer, builds the report, print it, and preview.

Public methods:

void **reportView**(); Displays a preview of the report.

void **reportPrint**(bool show = true) ; Print report, variable **show** – show print dialog.

QPrinter \* **printer**() ; returns the current object printer (QPrinter) report.

QList<QString> **models**(); Returns the list of the names of the models, used report for printing.

QObject \* **getModel**(QString name,bool addonabsent = false); Returns the pointer on object of the models of the inheritor from QAbstractListModel on name: name. In the absence of such a model name and addonabsent = true is added object QSqlTableModel with name: name.

void **addModel**(QString name,QObject\* object) ; Adding a model in the absence of a model with the name, name - model name, object - any type object inheritor from QAbstractListModel.

void **clearModels**(); Clearing the list of input models.

int **pageCount**() Returns the count of the pages of the active report.

bool **load**(QString xmltxt); Load report from string.

bool **load**(QIODevice \* dev); Load the report from file.

bool **load**(QByteArray & buffer); Load the report from byte aray QByteArray.

void **runDesigner**(); Run report designer without loaded report.

void **edit**(QString filename); Launching the report designer to edit the filename report. Passed the path and file name of the report.

void **edit**(QIODevice \* dev); Launching the report designer to edit the report, the loaded device dev.

void **edit**(QByteArray & buffer); Launching the report designer to edit the report, contained in a byte array buffer.

Signals:

void **signalGetValue**(QString paramName, QVariant &paramValue); Called to transmit the report of the custom variable values. Custom Variable is an arbitrary string enclosed in brackets []. For example: [company name].

void **signalBeginReport**(); Called before printing / viewing the report.

void **signalEndReport**(); Called at the end of the process the report output to print / preview.

void **signalSaveToStream**(QDomDocument \*); The signal caused by the report designer if the report file is transmitted to the editing as a byte array or stream from QIODevice device. To write custom handling of the report change. For example, write report file to the Blob field. A pointer to an object QDomDocument, containing the downloaded report file.

void **signalModelOnNextRec**(QObject \*); Called when the transition to the next record tabular data display. It is used, for example, to update the slave output data through the band: «Detail Data» reports with subordinate master-detail data. A reference to the model, the output of which is processed. Called when outputting through the bands output stream of tabular data: Master Data, Detail Data.

Example processing:

void FormDetailList1::slotModelOnNextRec(QObject \*object)

{

if(object == m\_customerliter){ //if the model is the parent //(customerliter) jumps to the next entry in the component a QTableView //its output and the corresponding data filtering subordinate table

QModelIndex index = m\_customerliter->*index*(m\_customerliter->query().at(),0);

if(index.isValid())

ui->tablemaster->setCurrentIndex(index);

}

}

void **signalModelOnFirstRec**(QObject \*); Called when setting the table output to the first record. A reference to the model, the output of which is processed. Called when outputting through the bands output stream of tabular data: Master Data, Detail Data.

void **signalNewPage**(int); The signal caused by the formation of a new page. Dispatched by the current number of processed pages.

**5. Report designer**

Report designer opens by QASReport::**runDesigner**() - the discovery of the empty designer. \

Or commands:

void **edit**(QString filename) with the transmission path and the name of the report file;

void **edit**(QIODevice \* dev); a pointer to the transfer of I/O device with the downloaded report file;

void **edit**(QByteArray & buffer); a pointer to the transfer of a byte array containing the downloaded report file;

* 1. **Interface**

Designer provides a convenient means to open, edit, save the report file, display it for viewing or printing. Editing the report form is made similarly as in a conventional graphics editor, using the same techniques for working with the mouse and keyboard.

With the following features:

* the properties of selected objects, you can simultaneously change both the toolbar and in the Property inspector;
* Some objects have editors that are triggered by double-clicking on it or by pressing the button - the Enter on the selected object, or by pressing the button in the Property inspector at the relevant property, either through the drop-down menu - Editor;
* Some properties the object "Text": font, size, and Mask, transfer and extension can be changed using the drop-down menu via the right mouse button;
* All designer items can be moved and positioned as anywhere. For some subjects, the location is not critical, eg for bands such as: Title, Page Hader, Page Footer, Column Title, respectively, they will always display the title at the beginning, the end of the page, regardless of their location on the page design. For most objects, their location is critical, and they print will be displayed right where you see them in the design of the page. Thus the appearance of the printed page is determined only by your imagination.
  1. **Main menu, keyboard commands, work with mouse**
* File operations, the main menu - File:
*  Ctr + N – the creation of a new report;
*  Ctr + O – open file report;
*  Ctr + S – save current report;
* Save as... – save writing renaming;
* The list of variables - opens an editor custom variable report. Custom variables - a strings created by the user to output data through them. Processing custom variables are creating a signal slot void QASreport::signalGetValue (QString paramName, QVariant & paramValue);
* Page options - to set the paper size, orientation, output, margins;
*  Ctr + P – call to preview the edited report;
* Exit - closing and the end of the editing of the report. In the case of changes to the report, the program will ask to be written;

Designer Editing Commands - Standard commands - Main Menu - Edit:

*  Ctr + Z – restore any previous editing operation (move, change properties, delete, etc.);
* return any undone edit operation;
*  Ctr – Y - return any undone edit operation;
* Ctr + X – Cut the selected objects;
* Ctr + C - copy the selected items to the clipboard;
* Ctr + V – past the selected items from the clipboard;
* Del – delete the selected items;
*  Ctr + A - select all items page design;
*  - add page design;
*  - delete current page design;
*  - move the selected objects in the display above the stack;
*  - move the selected objects in the display below the stack;

Designer editing commands that do not have links to the main menu:

* Shift - click on the object - the object to add to the selection list, repeated click deselects;
* Ctr + arrow keys - pixel by pixel movement of selected objects;
* Ctrl + Shift + arrow keys - the rapid movement of (5 pixels) of the selected objects;
* Shift + arrow keys - pixel by pixel resize selected objects;
* Click the mouse on the screen, the background page design - deselection;

Service commands - Main Menu - Service:

* Panel - switch - off panels: standard, text, frames, objects, alignment;
  + Standard toolbar - contains commands duplicate the menu - File and editing;
  + “Text” panel - contains commands for editing the text: font, size, thickness, color, vertical, horizontal alignment, text orientation;
  + “Frames" panel - provides a framework configuration commands around objects: the presence, thickness, border color, fill the background color, style;
  + “Objects" panel - includes design objects: text, Band, Image, Line, Figure, Barcode;
  + “Alignment" panel - command to align objects to the design page. With them, you can gently:

align the selected objects on the bottom, top, left, right edge. Either horizontally / vertically on the page. Evenly distribute the objects horizontally, vertically. To align the teams, a source of border leveling is the first selected object.

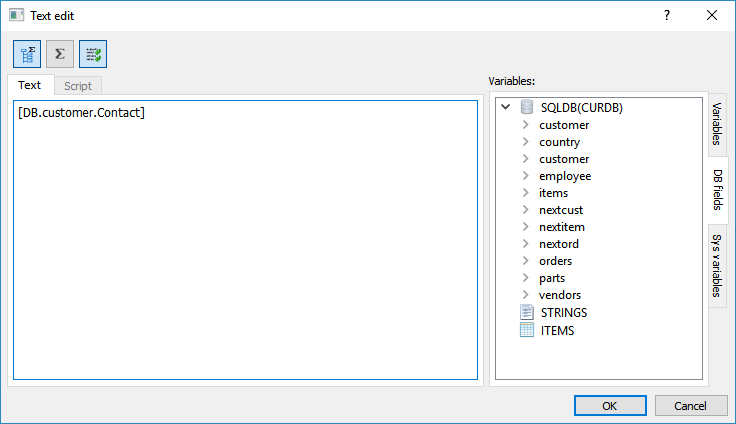
* designer settings:
  + Grid - map grid spacing design of the page, aligned to it;
  + page design line - display in millimeters or in inches;
  + Editing after insertion - start editor after creating a new object;
  + Display the header in the report's bands in the design;
* Help:
  + F1 – opening a help file;
  + Information about the program;
  1. **The visual designer objects**

All visual objects of the report presented six types: text, band, image, line, shape, bar code. Common properties of all objects of the designer are:  
Name - the name;  
Width - width;  
Height - height;  
Left - left indent;  
  Top - top margin;  
Border - the type of framing the frame;  
BorderStyle - style line framing frame;

If a component is the editor, highlight, make double-click on it or press the «Enter», or call the editor of the object through the Property inspector, or via a drop-down menu.

The object "Text" - is the main component of the design, the basic building block of the report. It can display ordinary strings or the contents of the fields of database tables or mathematical formulas, expressions of official functions and constants. Aside from the above had the following properties:

• vertical, horizontal alignment;  
• font;  
• word wrap;  
• output direction horizontal, vertical;  
• stretching (Strechable) - When setting this property, band, on which the facility is located, printing will change the height, according to the contents of the object " Text";  
• string format, which is used in the derivation of the object through the numbers, date / time, logical values. Format Editor, you can call either through the Property inspector, or via a drop-down menu;



Editor object "Text" by double-clicking on it or by pressing the button - Enter, calling through the pop menu or the Property inspector. Editor object " Text" is a simple text editor, toolbar and browser variables for automatic input variables. The browser has three tabs: custom variables, field data sources, system variables. Custom Variables (Variables) - arbitrary strings divided into groups, via which the output signal by means treating an QASReport::signalGetValue (QString paramName, QVariant & paramValue). In paramName, respectively, are transmitted user variables through paramValue returned values ​​assigned to them. Entering custom variables made through the main menu - file - a list of variables.

DB FIelds - variable field data sources of any type. Tree QSQLDB (ConnecttionName) groups attached table contains a list of all SQL databases, and table fields. STRINGS Group – strings models. ITEMS Group - all other types of models, from the heirs QAbstractItemModel. Fields SQL data source table contains names of fields, the fields of all other types of data - numeric fields indexes.

Sys variables - tab system with built-in variables:  
page# - page number;  
date# - the current date;  
time# - current time;  
linethrough# - number of processed recording streaming data globally, taking into account all the streaming data-bands;  
current# - number of processed recording streaming data of the current data band;  
totalpages# - the number of pages in the report;

Input variants text object "Text."

* **usual string** - a simple string is printed in the form in which you entered it;
* **[string variable], [requisites - name]** - string bracketed perceived as a variable. In the derivation, it is determined whether it is a custom, service variable or a field of the data source;
* **[DB.customer.Contact]** - variable - field data source table. ID Data Source: DB, then after a point, the name of tables: customer, and the next substring after the point: Contact - Name field.  
  For data sources that are not SQL database tables (QStringListModel, QStandardItemModel, any heirs of the QAbstractItemModel, except QSqlQueryModel), the field is indicated by a numerical index: [DB.itemsmodel.0], [DB.itemsmodel.1], etc.
* [[**amount of document] + [tax]], [sum ([amount of document]) / 100 \* 20]** - expression, any mathematical operations with variables must be taken in square brackets. Otherwise, when you print it will be perceived as a regular string;
* **Name of the company: [Company name] Company Address: [Company address]** - mixed input plain text, variables, expressions in a single object "Text";

**User variables** are entered via the main menu - List of variables.

|  |  |
| --- | --- |
|  | The form displays a list of report variables. Forms panel commands allow you to add, edit, delete variables and their group. For a variable, you must have a group. If the list of variables is empty, first click the Add button  - group.  Name of the variable may contain any set of characters and spaces, besides the square brackets: [].The number of variables and groups is not limited. After adding they will be available in the object editors "Text", "Picture", "Barcode". |

Example processing custom report variables in the program:

//--------------------------

void FormList::initPrint()

{

QObject::connect(MainForm->report(),SIGNAL(signalGetValue(QString,QVariant&)),this,

SLOT(slotGetValue(QString,QVariant&))); //init read m\_Report custom variables

}

void FormList::slotGetValue(QString paramName, QVariant &paramValue)

{

if(!paramName.compare("Company name"))

paramValue = m\_customer->query().value("Company").toString();

if(!paramName.compare("Company address"))

paramValue = m\_customer->query().value("Addr1").toString();

}

//--------------------------

**5.4.Bands**

QASreport is a band-oriented library. It is thanks to him and output data look the way you want. All objects are displayed within the boundaries of the band in which they are located. Some bands do not depend on where you position them. For example, Report Title, Report Summar, Page Heade, Page Footer, Column Header.  
For all other types of bands, their location on the page design is critical.

Brief description of the bands:

* **• Report Title -** displays at the beginning of the report. With this band displays the header data of the report. The report can be only one Report Title;
* **Report Summary -** output at the end of the report. Used to display the summation calculations;
* **Page Header –** appears at the beginning of each page;
* **Page Footer –** output at the end of each page;
* **Master Header –** It appears at the beginning of the print stream data from the master data band. Therefore, respectively should be placed on the page above the Master Data band, for which he is heading data. This is required because the number of bands in the Master Data Report can be anything. Convenient to use to display the table header.
* **Master Data –** base band to output data from any data source. It has the editor by double-clicking on it or by using the Enter key. In the editor, from the dropdown list select the names of the sources for your model.
* **Master Footer –** It appears at the end of O Master Data streaming. Convenient to use to display the total payments for the fields output Master Data;
* **Detail Header -** It appears at the beginning of the print stream data from the slave Detail Data band. Therefore, respectively should be placed on the page above the Detail Data band, for which he is heading data. Master Data Number of bands in the report can be anything. Convenient to use to display the table header.
* **Detail Data -**  band for the slave output data stream. The conclusion is made that after the withdrawal of the main data through Master Data band. It has the editor of the same Master Data band. Detail Data processing is carried out at each transition to a new entry in the main model Master Data parent. Parent for Detail Data is the closest upstream Master Data. So here is important processing signalModelOnNextRec(QObject\* model). The signal passes a reference to the model. After receiving a signal from the parent model to take appropriate steps to filter child data, go to the next record, etc. Number of Detail Data can be anything.
* **Detail Footer -** It appears at the end of output Detail Data streaming. Convenient to use to display the total calculations for output fields Detail Data;
* **Column Header -** It appears at the beginning of the output table data through the first Master Data band on the page. When transferring data to the next page, respectively, is displayed in the top of the page. There can only be one page;
* **Column Footer –** It appears at the end of the output of the first Master Data band on the page designer;
* **Group Header** – Band to display grouped data. The editor of this band indicates the source model and the field over which the data group output from the Data band (Master or Detail). To correctly display these outputted data, respectively to be grouped. On the design of the page Group Header band should be located above the date of the band, which he grouped data displays. Number of Group Header bands may be any appropriate number of grouping fields in the parent table.
* **Group Footer -** It appears at the end of the output grouped data. On the design of the page is placed after the Data band, in the same manner as his Group Header, for which it serves as final;

|  |  |
| --- | --- |
| The processing sequence when printing bands. | In the same way bands should be placed on the design of the page to print the correct output. |

Aggregate functions: SUM, COUNT, AVG, MAX, MIN variable can be used only on summation Band: Report Summary, Master Footer, Detail Footer, Group Footer. Accordingly, the count will be the level of data covered by the data bands, which Footer data are final. In a band, Report Summary - units produced settlement of all data sources report.

1. **General Workflow**

Create a new report.

To be continued…

autor: Zgukov Serg

Contact details:

email: [serg\_@ukr.net](mailto:serg_@ukr.net)

WWW: