Printing of rich data forms and reports in wxWidgets GUI toolkit

Michal Bližňák, Tomáš Dulík and Roman Jašek

Abstracts— Well-known cross-platform GUI toolkit called wxWidgets currently available in its 3rd major version is often used for cross-platform desktop applications development covering all major desktop OSes not only by wide open-source community but also by big commercial companies such are Intel [1], AVG, etc. Unfortunately, despite its maturity and wide range of built-in features, the well-known cross-platform GUI toolkit called wxWidgets still lacks support for easy printing of reports and forms. This paper introduces new library add-on called wxReportDocument which fills this gap and allows users to easily create rich reports and forms able to preview and/or print run-time application data.

Keywords—wxWidgets, cross-platform, reports, forms, printing, data, binding, C/C++

I. Introduction

WELL-KNOWN cross-platform GUI toolkit called wxWidgets offers wide range of features covering nearly all functionality of target operation systems including GUI definition, processes and threads control, file system access, sockets, streams and many other features [6][9]. Unfortunately, dedicated support for printing of forms or reports is still missing even in the latest version of the library (at the moment the version number is 3.0.2 released in October 2014). Although there exists general support for printing allowing users to "manually" draw onto the printer's canvas (including print preview functionality) and possibility to print HTML-based content [7], there is no easy way how to defined full-featured reports and forms able to publish run-time application data stored in scalar variables or arrays.

This article introduces new wxWidgets add-on library called *wxReportDocument* created at Tomas Bata University aimed to fill this gap which is together with wxShapeFramework [2] and wxXmlSerializer [3] our 3rd major contribution to this cross-platform GUI toolkit. The library offers a functionality needed for creation of print reports and forms. User-defined forms can be printed by using the library's internal printing back-end based on the wxWidget's printing framework or it can be saved to an output XML file (via any available output stream) for further usage. Also, it allows users to define unlimited number of printed pages consisting of various, highly customizable page items as shown later in this article.

The work was performed with the financial support of the research project NPU I No. MSMT-7778/2014 by the Ministry of Education of the Czech Republic and also by the European Regional Development Fund under the project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089.

Michal Bližňák is with the Tomas Bata University, Faculty of Applied Informatics, Department of Informatics and Artificial Intelligence, Nad Stranemi 4511, 76005 Zlin, Czech Republic (corresponding author to provide phone: 00420-576035187; e-mail: bliznak@fai.utb.cz).

Tomáš Dulík and Roman Jašek are with the Tomas Bata University, Faculty of Applied Informatics, Department of Informatics and Artificial Intelligence, Nad Stranemi 4511, 76005 Zlin, Czech Republic (e-mails: dulik@fai.utb.cz and jasek@fai.utb.cz).

- Text items
  - Various possibilities of the text formatting are provided: text font, foreground and background color, text indentation on the line and height of the line can be specified.
  - Texts can be wrapped into multi-line boxes with specified width and height.
  - Different formatting styles of the text are allowed within the one text box.
  - Text values can be read dynamically from run-time data sources (scalar variables and arrays).
- Images
  - All image formats supported by wxImage class can be used.
  - PPI value can be changed in order to scale loaded image.
- Tables
  - Table's rows or columns can be filled from arrays.
  - Dimension of the table is adjusted automatically accordingly to the sizes of the source arrays.
  - Automatic page break is used when the table exceeds the page extent.
- Static graphics shapes
  - Lines
  - Rectangles
  - Circles

Another supported features are:

- Settings of page size, margin, header and footer with elements.
- Automatic page numbering.
- Predefined styles of formatting for all page elements.
- Printing and previewing support.
- Serialization and deserialization of the reports to or from XML files.
- Pre-defined file handler class for additional/custom output file formats.

II. wxReportDocument library's internals and structure

The *wxReportDocument* library is created upon C++ implementation of wxWidgets GUI toolkit as its feature add-on. It uses built-in classes for graphics output, printing and previewing support and creates upper-level interface for definition of forms and reports. It can be used with both main wxWidgets 2.8 and 3.0 branches and can be easily obtained from wxCode add-ons repository [5].

The library consists of the following main components:

Report document is encapsulated by `wxReportDocument class` which can be regarded as a main library class responsible for management of defined document pages. It also defines API functions for serialization, printing and previewing of the document. Typically, the report document class instance should be created and initialized at the application start-up and should be available during all application's life-time.

Report page is a basic printing entity managed by the *report document* encapsulated by `wxReportPage` class. Each the document can contain one or more report pages. Moreover, one report page defined by the user can be divided into several printed pages automatically when needed (e.g. when a table placed onto specific report page exceeds its vertical dimension) so the overall number of printed pages can be higher then number of user-defined pages.

Report page item is a single graphic object defined by the user and placed onto the *report page* which is encapsulated by `wxReportPageItem` class or another inherited ones. It can be a static graphics object like bitmap image and vector shape (rectangle, cirle or line), static text object defined in the source code, dynamic text object displaying textual representation of values created at the run-time, statically or dynamically defined tables or another user-defined item with custom drawing.

The Figure 1 show inheritance diagram of all available report items supported by the RP library.

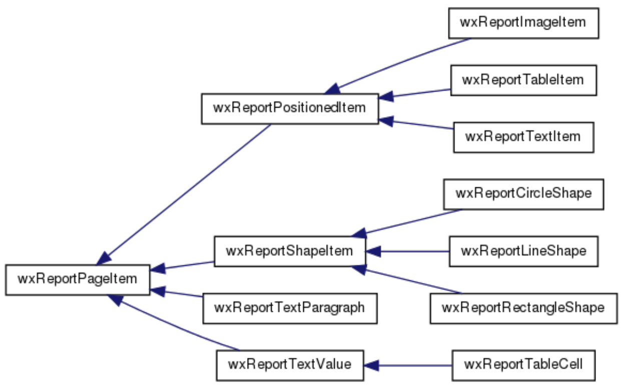


Fig. 1 Inheritance diagram of page item classes

Report style is a non visual object defining attributes of *report page* or *report page item*. It is encapsulated by `wxReportStyle` class whose objects can be assigned to report page items or directly to document page. It allows definition of foreground and background color, font, border style and other properties.

The Figure 2 show inheritance diagram of all available report styles supported by the RP library.

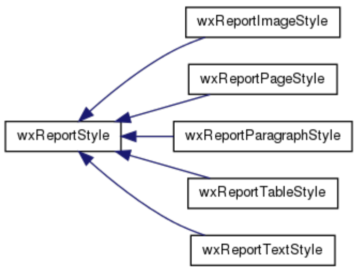


Fig. 2 inheritance diagram of report style classes

In addition to the mentioned components and classes the *wxReportDocument* library includes also other auxiliary classes encapsulating needed functionality. In the most cases, these classes are not intended to be used directly by the end-user so we will not discuss them here in details. For more information about them please refer to the library's project documentation available from the distribution package.

III. Definition of the reports and the forms

The following chapters deal with specific tasks needed for successful document creation.

The fist step needed for the proper report document creation is definition of a report document object and setting its properties. As mentioned above, the report document object should be persistent for whole the application's life-time due to possibility to ask for the refresh/drawing of the document anytime later. Of course, this functionality is highly dependent on specific usage scenarios so it is completely up to the user how the report document object will be treated.

Now let us focus to the report page creation process. It is supposed that the reader is familiar with basic aspects of programming with C++ language and with usage of wxWidgets library as well. Also assume that global instance of `wxReportDocument` class named `m_Report` already exists like shown in Listing 1.

Listing 1: Main document repor tobject

```
#include <wx/report/reportdocument.h>

wxReportDocument m_Report;
```

A. Basic page layout

After the creation of the document object basic properties including the page size and margins can be set. Also notice that one document page is created at the document object initialization by default so it is not needed to do that explicitly.

Now, let us to discus basic page layout as shown in Figure 3.

The report page consists of three main parts: *header*, *custom page area* and *footer*. The page width, height and margins surrounding the custom area can be set by using `wxReportPageStyle` class as can be seen from Listing 2. Remember that *wxReportDocument* library uses millimeters

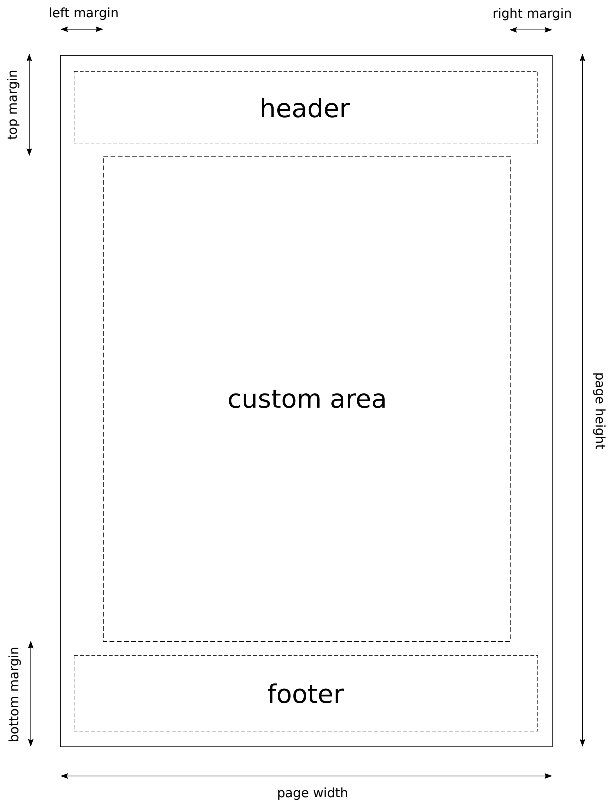


Fig. 3 basic page layout

[mm] as its native dimension units type for all sizes and positioning.

Now, let us to define dimensions of the page and its margins.

Listing 2: Page layout definition

```
wxReportPageStyle pgs;
pgs.SetWidth( 210 );
pgs.SetHeight( 297 );
pgs.SetMargins( 10, 10, 30, 30 );
pgs.SetBorder( wxRP_ALLBORDER );
pgs.SetBackgroundColor( wxColour( 220, 220, 220 ) );

m_Report.SetPageStyle( pgs );
```

Defined page can be previewed by using special API function called `ShowPrintPreview()` defined in `wxReportDocument` class like shown in Listing 3.

Listing 3: Show report in preview window

```
m_Report.ShowPrintPreview( this, wxSize( 640, 850 ) );
```

As a response the application shows print preview window displaying defined pages and allowing user to print the content by using available printers as can be seen from Figure 4. Also, the document can be printed immediately without showing the preview window by using `wxReportDocument::Print()` function.

Headers and Footers In contrast to page items placed into custom page area, items managed by the header or the footer are not restricted by specified page dimensions and margins. It means that given positions of header/footer items are absolute while standard page items coordinates are relative to origin of the custom page area. In general, the content of the header, the footer

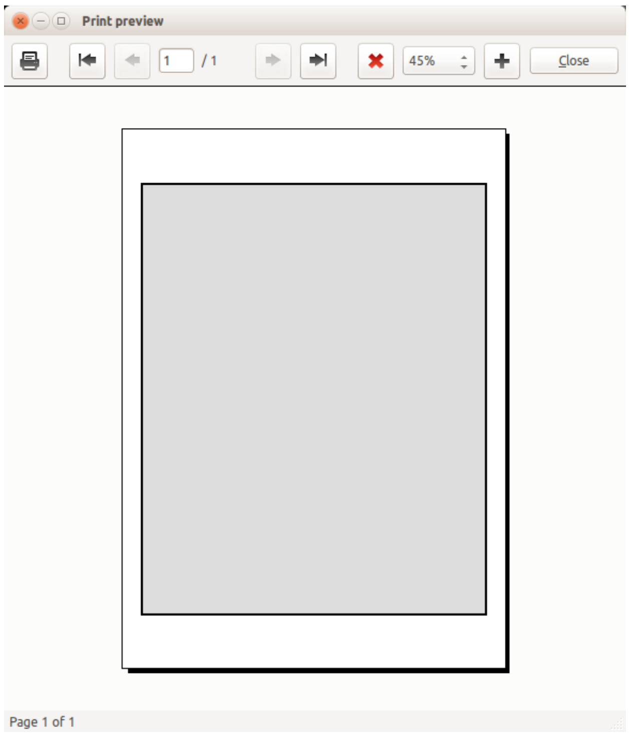


Fig. 4 print preview window

and the custom page area can be the same. The difference is that a content of the header and the footer is repeated on all document pages while content of the custom page area can be unique. Also, there exists dedicated function for placing the page items into the header/footer declared in `wxReportDocument` class.

Listing 4: API for definition of the header/footer content

```
void wxReportDocument::AddItemToHeader(const
        wxReportTextItem& textItem);
void wxReportDocument::AddItemToFooter(const
        wxReportTextItem& textItem);
```

The following code shown in Listing 5 illustrates how to add some content into the page header.

Listing 5: Definition of the header content

```
wxReportTextItem hi;

hi.SetSize( 200, 20 );
hi.SetTextAlign( wxRP_CENTERALIGN );
hi.SetPosition( wxRP_CENTER, 10 );
hi.AddText( "Lorem␣Ipsum␣..." );

m_Report.AddItemToHeader( hi );
```

Page numbers The main class encapsulating printed document includes also API function called `InsertPageNumbering()` suitable for specification of page numbers. Of course, it is also possible to do that by using user-defined text items placed into the headers/footers but the dedicated function does lot of needed work itself.

Listing 6: API for definition of page numbering

```
void wxReportDocument::InsertPageNumbering(const wxString&
        format, const wxReportTextStyle& textStyle, double
        posX, double posY, int atPlace = wxRP_FOOTER, int
        fromPage = 0);
```

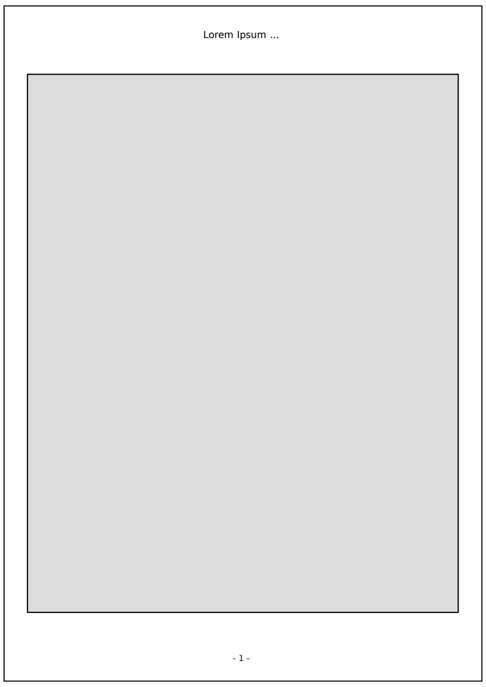


Fig. 5 numbered page with header

The `InsertPageNumbering` function allows users to specify:

- format of page number in `printf`-like way,
- text style used with the page number,
- X- and Y- coordinates of the page number (in absolute values),
- target container (i.e. `wxRP_HEADER`, `wxRP_FOOTER` or `wxRP_BODY` respectively) and
- starting page.

Let us to define page number placed into the footer and show the result in Figure 5.

Listing 7: Definition of the page number

```
wxReportTextStyle ns( "ts_numbering", wxFont(10,
        wxFONTFAMILY_SWISS, wxFONTSTYLE_NORMAL,
        wxFONTWEIGHT_NORMAL) );
m_Report.InsertPageNumbering( "-␣%d␣-", ns, wxRP_CENTER,
        285 );
```

IV. Report styles

In the most cases, the library works in some sort of state-tracking mode which means that the defined styles are applied on all items added to the pages later after the style's definition and assignment. There are several classes encapsulating style objects as shown in Figure 2 which can be used for specific needs. For example, `wxReportStyle` is base common style object defining mainly page/items borders and background color, `wxReportPageStyle` class can be used for definition of overall document look, `wxReportTextStyle` class can be used for styling of page text items, `wxReportParagraphStyle` class can be used for styling of text paragraphs, etc.

The style objects can be assigned to relevant page items or to the document page itself and can be re-used freely. Note that some page items can use several style object at the same time. Let us explain the styling process on text item encapsulated by `wxReportTextItem` class.

Text item can contain several words or lines with user-defined line endings and can be also divided into one or more paragraphs. The text item uses two different style objects: the *text style* and the *paragraph style* which are inherited from base *report style*. Except the common properties such are border style, border and background color the text style allows users to defined used font and foreground color while the paragraph style allows definition of text alignment, indentation, line height and paragraph spacing. Both styles can be assigned to the text item concurrently and are valid until replaced by another style object.

Now, let us to examine how the style objects assigned to the text item can affect its look.

At the first a text item consisting of three paragraphs with standard *Lorem Ipsum* content will be defined and inserted into the document page as shown in Listing 8

Listing 8: Definition of text item

```
1   float width = pgs.GetSize().x - pgs.GetLeftMargin() - pgs.
        GetRightMargin() - 10;
2
3   wxReportTextItem ti;
4   ti.SetPosition( wxRP_CENTER, 5 );
5   ti.SetSize( width, 0 );
6
7   /* assume existing lorem_ipsum_par_1, lorem_ipsum_par_2
        and lorem_ipsum_par_3 string variables containing
        printed text */
8   ti.AddText( lorem_ipsum_par_1 );
9
10  ti.AddNewParagraph();
11  ti.AddText( lorem_ipsum_par_2 );
12
13  ti.AddNewParagraph();
14  ti.AddText( lorem_ipsum_par_3 );
15
16  m_Report.AddItem( ti );
```

The text item's dimensions are calculated and specified at lines 1 and 5. Notice that just the width must be set by the user here - the height will be (in this case) calculated by the library itself. After that, the text item is center on the page within previously defined margins. The vertical position of the text is 5 millimeters under the top page margin. The content is divided into three paragraphs (see lines 10 and 13) and inserted by dedicated function at lines 8, 11 and 14. Finally, the newly defined text item is added to the page at line 16. The printed result is shown in Figure 6.

Now let us to play with the styles little bit. In the following sample, three style objects are used to style paragraphs, text content and the text item itself.

Listing 9: Definition of styled text item

```
float width = pgs.GetSize().x - pgs.GetLeftMargin() - pgs.
    GetRightMargin() - 10;

// define common report style
wxReportStyle rs;
rs.SetBorder( wxRP_ALLBORDER, wxColour(245, 245, 245) );
rs.SetBackgroundColor( rs.GetBorderColor() );

// define text style
wxReportTextStyle ts;
ts.SetFont( wxFont(12, wxFONTFAMILY_ROMAN,
    wxFONTSTYLE_ITALIC, wxFONTWEIGHT_NORMAL) );

// define paragraph style
```

Lorem Ipsum ...

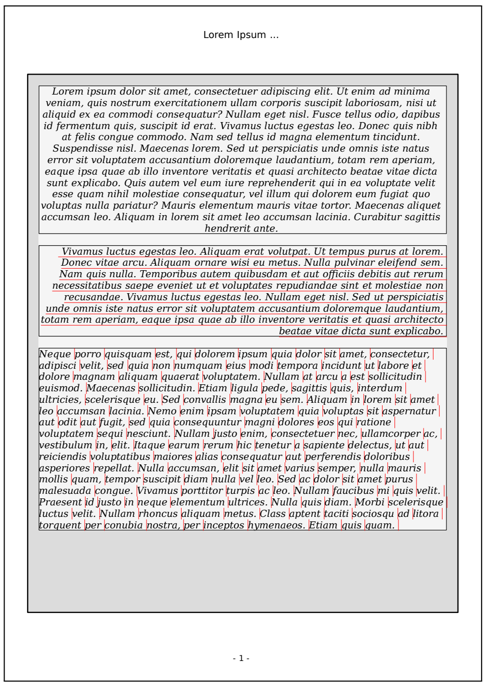
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Nullam eget nisl. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Vivamus luctus egestas leo. Donec quis nibh at felis congue commodo. Nam sed tellus id magna elementum tincidunt. Suspendisse nisl. Maecenas lorem. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur? Mauris elementum mauris vitae tortor. Maecenas aliquet accumsan leo. Aliquam in lorem sit amet leo accumsan lacinia. Curabitur sagittis hendrerit ante.

Vivamus luctus egestas leo. Aliquam erat volutpat. Ut tempus purus at lorem. Donec vitae arcu. Aliquam ornare wisi eu metus. Nulla pulvinar eleifend sem. Nam quis nulla. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Vivamus luctus egestas leo. Nullam eget nisl. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Nullam at arcu a est sollicitudin euismod. Maecenas sollicitudin. Etiam ligula pede, sagittis quis, interdum ultricies, scelerisque eu. Sed convallis magna eu sem. Aliquam in lorem sit amet leo accumsan lacinia. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Nullam justo enim, consectetuer nec, ullamcorper ac, vestibulum in, elit. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat. Nulla accumsan, elit sit amet varius semper, nulla mauris mollis quam, tempor suscipit diam nulla vel leo. Sed ac dolor sit amet purus malesuada congue. Vivamus porttitor turpis ac leo. Nullam faucibus mi quis velit. Praesent id justo in neque elementum ultrices. Nulla quis diam. Morbi scelerisque luctus velit. Nullam rhoncus aliquam metus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Etiam quis quam.

- 1 -

Fig. 6 Composed text item without styling



Lorem Ipsum ...

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Nullam eget nisl. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Vivamus luctus egestas leo. Donec quis nibh at felis congue commodo. Nam sed tellus id magna elementum tincidunt. Suspendisse nisl. Maecenas lorem. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur? Mauris elementum mauris vitae tortor. Maecenas aliquet accumsan leo. Aliquam in lorem sit amet leo accumsan lacinia. Curabitur sagittis hendrerit ante.

Vivamus luctus egestas leo. Aliquam erat volutpat. Ut tempus purus at lorem. Donec vitae arcu. Aliquam ornare wisi eu metus. Nulla pulvinar eleifend sem. Nam quis nulla. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Vivamus luctus egestas leo. Nullam eget nisl. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Nullam at arcu a est sollicitudin euismod. Maecenas sollicitudin. Etiam ligula pede, sagittis quis, interdum ultricies, scelerisque eu. Sed convallis magna eu sem. Aliquam in lorem sit amet leo accumsan lacinia. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Nullam justo enim, consectetuer nec, ullamcorper ac, vestibulum in, elit. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat. Nulla accumsan, elit sit amet varius semper, nulla mauris mollis quam, tempor suscipit diam nulla vel leo. Sed ac dolor sit amet purus malesuada congue. Vivamus porttitor turpis ac leo. Nullam faucibus mi quis velit. Praesent id justo in neque elementum ultrices. Nulla quis diam. Morbi scelerisque luctus velit. Nullam rhoncus aliquam metus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Etiam quis quam.

- 1 -

Fig. 7 styled text item

```
13   wxReportParagraphStyle ps;
14   ps.SetTextAlign( wxRP_CENTERALIGN );
15   ps.SetParagraphsSpace( 5 );
16   ps.SetBorder( wxRP_ALLBORDER, *wxBLACK, 0.2 );
17
18   // define text item
19   wxReportTextItem ti;
20   ti.SetPosition( wxRP_CENTER, 5 );
21   ti.SetSize( width, 0 );
22
23   // set all styles
24   ti.SetStyle( rs );
25   ti.SetActiveTextStyle( ts );
26   ti.SetActiveParagraphStyle( ps );
27
28   ti.AddText( lorem_ipsum_par_1 );
29
30   ti.AddNewParagraph();
31
32   // modify style and apply them on next paragraph
33   ts.SetBorder( wxRP_BOTTOMBORDER, *wxRED, 0.1 );
34   ti.SetActiveTextStyle( ts );
35   ps.SetTextAlign( wxRP_RIGHTALIGN );
36   ti.SetActiveParagraphStyle( ps );
37
38   ti.AddText( lorem_ipsum_par_2 );
39
40   ti.AddNewParagraph();
41
42   // modify style and apply them on next paragraph
43   ts.SetBorder( wxRP_RIGHTBORDER, *wxRED, 0.1 );
44   ti.SetActiveTextStyle( ts );
45   ps.SetTextAlign( wxRP_LEFTALIGN );
46   ti.SetActiveParagraphStyle( ps );
47
48   ti.AddText( lorem_ipsum_par_3 );
49
50   m_Report.AddItem( ti );
```

At the first, a basic report style named `rs` is created at line 4. This object is used for drawing of light gray rectangle filling the text item's area. It is necessary to define both border and background color as can be seen at lines 5 a 6 because no page item can be filled without existing border. The borders can be specified by combination of binary flags `wxRP_LEFTBORDER`, `wxRP_RIGHTBORDER`, `wxRP_TOPBORDER`, `wxRP_BOTTOMBORDER` or simply `wxRP_ALLBORDER`. This general style object can be assigned to any type of page item by using `wxPageItem::SetStyle()` function.

Next, a text style object called `ts` is created at line 9. This style type can be applied just onto text items and allows definition of text font. This style can be assigned to the text item by using `wxReportTextItem::AddActiveTextStyle()` function as shown at line 25. From this point, any text value added to the text item uses this text style until changed to another one.

Finally, the paragraph style object is created at line 13 to specify various paragraph's properties like paragraph spacing or text alignment. Also, both text and paragraph style objects allow definition of their own borders. Borders defined by paragraph style are applied onto all following paragraphs added to parent text item while borders defined in text style object are applied onto single words. Paragraph style object can be assigned to parent text item by using `wxReportTextItem::AddActiveParagraphStyle()` function. Fully styled text item is resulting from the sample code is show in Figure 7.

V. Positioning of the Page Items

As mentioned in previous chapters, all page items can be positioned within custom page area or within the headers/footers areas by using absolute coordinates which are specified in *millimeters*. In addition to the absolute positioning, also predefined, automatically calculated position marks are available. These are:

- `wxRP_LEFT`,
- `wxRP_RIGHT`,
- `wxRP_TOP`,

- `wxRP_BOTTOM`,
- and `wxRP_CENTER`.

All position marks can be used for both vertical and horizontal positioning as shown at line 20 in Listing 9.

VI. Page items

In general, printed pages consist of set of page items with user-defined properties and styles. The following sub chapters describes all available item types in details.

A. Text items

Text items encapsulated by `wxReportTextItem` class can be used for inserting of single words or complex paragraphs as shown in Chapter IV.

There is also another important feature of the text item: it can display content of assigned variable updated at *run-time* automatically or on demand. It means that real displayed text do not has to be "hard-coded" in source code, but can be read from program variables. Assignment of source variable can be done via `wxReportTextItem::AddVariable()` function which can take reference to `short`, `int`, `long`, `float`, `double`, `char` and `wxString` variables as its argument. This feature is ideal for definition of printed forms filled with calculated values, text phrases, etc. Values from assigned variables are converted to its textual representation automatically before printing/previewing or when requested by user via `wxReportDocument::RefreshVariables()` function.

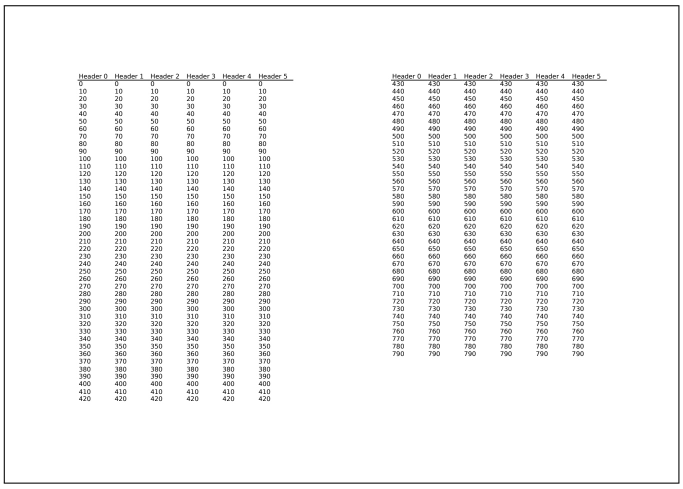
B. Tables

Another very useful page item provided by *wxReportDocument* library is *table item*. As expected, the *tables* can be used for printing of data in tabular form. The table item is encapsulated by `wxReportTableItem` class which defines rich API suitable for handling its data, dimensions and style. The class members allow user to customize the table in many ways: if required, each table cell can be styled in completely different way, data can be inserted into the table per cell, per row and also per column. The table cell can contain single textual value or assigned variable used for filling the cell with desired value at run-time similarly to the text item as mentioned above (actually, the table cell *is* the text item so it provides similar functionality). Moreover, the tables can contain row or cell headers and are divided into several blocks fitting parent document page automatically when needed.

Now, let us demonstrate the table item's API in two examples. The first one shows how to create simple table and how to fill it with values stored in array of integers. The sample code could be as follows:

Listing 10: Simple table item

```
1  // create array on integers
2  wxArrayInt columnValues;
3  for( size_t i = 0; i < 80; ++i ) columnValues.Add( i * 10
       );
4
5  // define page style
6  wxReportPageStyle pgs;
7  pgs.SetWidth( 210 );
8  pgs.SetHeight( 297 );
9  pgs.SetMargins( 10, 10, 30, 30 );
10
11 m_Report.SetPageStyle( pgs );
12
```



Header 0	Header 1	Header 2	Header 3	Header 4	Header 5
0	0	0	0	0	0
10	10	10	10	10	10
20	20	20	20	20	20
30	30	30	30	30	30
40	40	40	40	40	40
50	50	50	50	50	50
60	60	60	60	60	60
70	70	70	70	70	70
80	80	80	80	80	80
90	90	90	90	90	90
100	100	100	100	100	100
110	110	110	110	110	110
120	120	120	120	120	120
130	130	130	130	130	130
140	140	140	140	140	140
150	150	150	150	150	150
160	160	160	160	160	160
170	170	170	170	170	170
180	180	180	180	180	180
190	190	190	190	190	190
200	200	200	200	200	200
210	210	210	210	210	210
220	220	220	220	220	220
230	230	230	230	230	230
240	240	240	240	240	240
250	250	250	250	250	250
260	260	260	260	260	260
270	270	270	270	270	270
280	280	280	280	280	280
290	290	290	290	290	290
300	300	300	300	300	300
310	310	310	310	310	310
320	320	320	320	320	320
330	330	330	330	330	330
340	340	340	340	340	340
350	350	350	350	350	350
360	360	360	360	360	360
370	370	370	370	370	370
380	380	380	380	380	380
390	390	390	390	390	390
400	400	400	400	400	400
410	410	410	410	410	410
420	420	420	420	420	420

Header 0	Header 1	Header 2	Header 3	Header 4	Header 5
430	430	430	430	430	430
440	440	440	440	440	440
450	450	450	450	450	450
460	460	460	460	460	460
470	470	470	470	470	470
480	480	480	480	480	480
490	490	490	490	490	490
500	500	500	500	500	500
510	510	510	510	510	510
520	520	520	520	520	520
530	530	530	530	530	530
540	540	540	540	540	540
550	550	550	550	550	550
560	560	560	560	560	560
570	570	570	570	570	570
580	580	580	580	580	580
590	590	590	590	590	590
600	600	600	600	600	600
610	610	610	610	610	610
620	620	620	620	620	620
630	630	630	630	630	630
640	640	640	640	640	640
650	650	650	650	650	650
660	660	660	660	660	660
670	670	670	670	670	670
680	680	680	680	680	680
690	690	690	690	690	690
700	700	700	700	700	700
710	710	710	710	710	710
720	720	720	720	720	720
730	730	730	730	730	730
740	740	740	740	740	740
750	750	750	750	750	750
760	760	760	760	760	760
770	770	770	770	770	770
780	780	780	780	780	780
790	790	790	790	790	790

Fig. 8 simple table item spread over two document pages

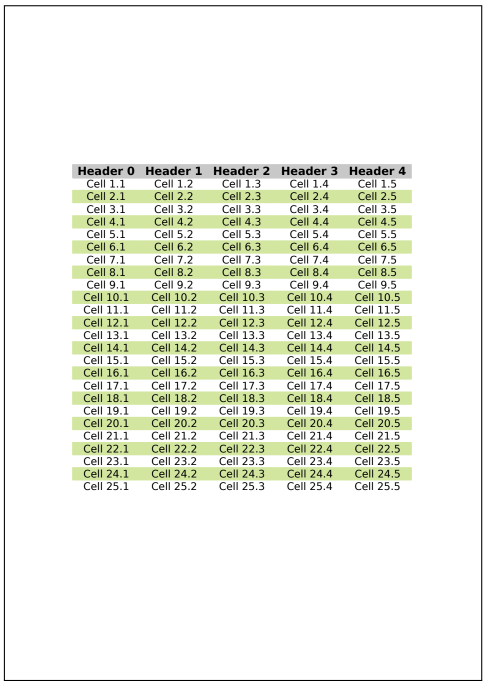
```
13 // create table item and center it within document page
14 wxReportTableItem table;
15 table.SetPosition( wxRP_CENTER, 10 );
16 table.SetTextAlign( wxRP_LEFTALIGN );
17
18 // define default style used also for table headers
19 wxReportTextStyle cellsStyle;
20 cellsStyle.SetBorder( wxRP_BOTTOMBORDER );
21 table.SetCellsStyle( cellsStyle );
22
23 // insert six columns with custom headers into the table
24 for( size_t c = 0; c < 6; ++c) table.AddColumn(
       columnValues, wxString::Format("Header %u", c) );
25
26 // style specific table rows
27 cellsStyle.SetBorder( wxRP_NOBORDER );
28 for( size_t i = 0; i < 80; ++i ) table.SetCellsStyleForRow
       ( cellsStyle, i );
29
30 // adjust table cells sizes automatically
31 table.SynchronizeCellsSizes();
32
33 // add the table to the document
34 m_Report.AddItem( table );
```

The most interesting aspect of this sample is the way how the table content is defined: the table columns are filled with values stored in array on integers as can be seen at line 24. Also, the column's header is defined there. In addition, two different styles are used with the table. The first one defined at line 18 is used for both table cells and the column headers while the second one defined at line 27 is assigned just to specific table rows. Notice, that the extent of the table exceeds document page dimensions so the table is divided into two pieces. The result of this sample code can be seen in Figure 8 which shows both printed pages.

The second example illustrates how to handle each table cell separately. In the source code several different text styles are used. Also, the content of the table cells is defined "manually" per-cell by using dedicated API function.

Listing 11: Styled table item

```
1  // define page style
2  wxReportPageStyle pgs;
3  pgs.SetWidth( 210 );
4  pgs.SetHeight( 297 );
5  pgs.SetMargins( 10, 10, 30, 30 );
6
7  m_Report.SetPageStyle( pgs );
8
9  // create table item and center it within document page
10 wxReportTableItem table;
11 table.SetPosition( wxRP_CENTER, 40 );
12 table.SetTextAlign( wxRP_CENTERALIGN );
```



Header 0	Header 1	Header 2	Header 3	Header 4
Cell 1.1	Cell 1.2	Cell 1.3	Cell 1.4	Cell 1.5
Cell 2.1	Cell 2.2	Cell 2.3	Cell 2.4	Cell 2.5
Cell 3.1	Cell 3.2	Cell 3.3	Cell 3.4	Cell 3.5
Cell 4.1	Cell 4.2	Cell 4.3	Cell 4.4	Cell 4.5
Cell 5.1	Cell 5.2	Cell 5.3	Cell 5.4	Cell 5.5
Cell 6.1	Cell 6.2	Cell 6.3	Cell 6.4	Cell 6.5
Cell 7.1	Cell 7.2	Cell 7.3	Cell 7.4	Cell 7.5
Cell 8.1	Cell 8.2	Cell 8.3	Cell 8.4	Cell 8.5
Cell 9.1	Cell 9.2	Cell 9.3	Cell 9.4	Cell 9.5
Cell 10.1	Cell 10.2	Cell 10.3	Cell 10.4	Cell 10.5
Cell 11.1	Cell 11.2	Cell 11.3	Cell 11.4	Cell 11.5
Cell 12.1	Cell 12.2	Cell 12.3	Cell 12.4	Cell 12.5
Cell 13.1	Cell 13.2	Cell 13.3	Cell 13.4	Cell 13.5
Cell 14.1	Cell 14.2	Cell 14.3	Cell 14.4	Cell 14.5
Cell 15.1	Cell 15.2	Cell 15.3	Cell 15.4	Cell 15.5
Cell 16.1	Cell 16.2	Cell 16.3	Cell 16.4	Cell 16.5
Cell 17.1	Cell 17.2	Cell 17.3	Cell 17.4	Cell 17.5
Cell 18.1	Cell 18.2	Cell 18.3	Cell 18.4	Cell 18.5
Cell 19.1	Cell 19.2	Cell 19.3	Cell 19.4	Cell 19.5
Cell 20.1	Cell 20.2	Cell 20.3	Cell 20.4	Cell 20.5
Cell 21.1	Cell 21.2	Cell 21.3	Cell 21.4	Cell 21.5
Cell 22.1	Cell 22.2	Cell 22.3	Cell 22.4	Cell 22.5
Cell 23.1	Cell 23.2	Cell 23.3	Cell 23.4	Cell 23.5
Cell 24.1	Cell 24.2	Cell 24.3	Cell 24.4	Cell 24.5
Cell 25.1	Cell 25.2	Cell 25.3	Cell 25.4	Cell 25.5

Fig. 9 styled table item

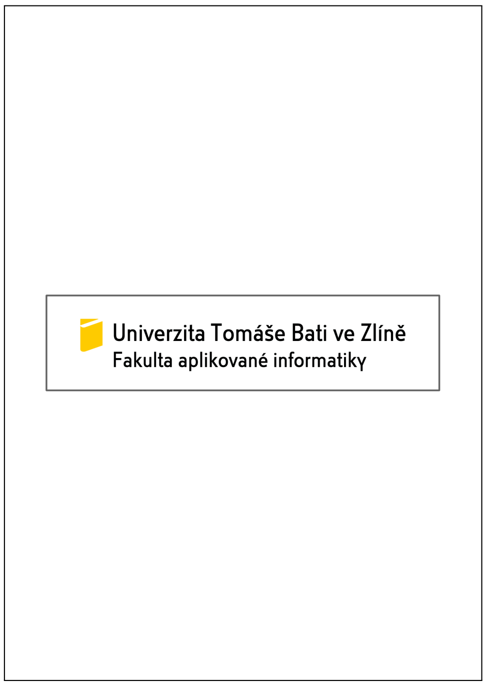

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Fig. 10 centered image item

```

// define style for table header
wxReportTextStyle cellsStyle;
cellsStyle.SetBorder( wxRP_BOTTOMBORDER );
cellsStyle.SetBorderColor( wxColour(200, 200, 200) );
cellsStyle.SetFont( wxFont(14, wxFONTFAMILY_DEFAULT,
    wxFONTSTYLE_NORMAL, wxFONTWEIGHT_BOLD) );
cellsStyle.SetTextColor( *wxBLACK );
cellsStyle.SetBackgroundColor( wxColour(200, 200, 200) );
table.SetCellsStyle(cellsStyle);

// insert an empty table row and fill it with custom
    values.
table.AddRow();
for(int c=0; c<5; ++c)
  table.AddCellToRow( wxString::Format(wxT("Header␣%d"), c
      ) );

// define style for table content
cellsStyle.SetBorder(0);
cellsStyle.SetBorderColor(wxColour(210, 230, 160));
cellsStyle.SetFont(wxFont(13, wxFONTFAMILY_DEFAULT,
    wxFONTSTYLE_NORMAL, wxFONTWEIGHT_NORMAL));
table.SetCellsStyle(cellsStyle);

// specify different styles for even and odd table rows
for(int r=0; r<25; ++r) {
  if(r % 2 != 0)
    cellsStyle.SetBackgroundColor(wxColour(210, 230, 160))
        ;
  else
    cellsStyle.SetBackgroundColor(wxNullColour);
  table.SetCellsStyle(cellsStyle);

    // fill the table rows with custom values
  table.AddRow();
  for(int c=0; c<5; ++c)
    table.AddCellToRow( wxString::Format(wxT("Cell␣%d.%d")
        , r+1, c+1) );
}

// set width of all columns
table.SetColumnWidth(30);

m_Report.AddItem( table );
```

The result of Listing 11 is shown in Figure 9.

C. *Images*

In addition to standard document elements like text item or table item mentioned in the previous chapters, the *wxReportDocument* library supports also set of static graphics items suitable for improvement of the document look.

The first item we are going to discuss here is *image item* encapsulated by `wxReportImageItem` class able to display images loaded from file system. The image item can be positioned similarly to text or table items within the custom page area od header/footer. It can be used in conjunction with dedicated style class called `wxReportImageStyle` allowing user to set image borders or margins. Also, PPI value can be customized via `wxReportImageItem::SetPPI()` API function so the image will be scaled in accordance to the set PPI value.

Let us to illustrate usage of *image item* on a simple example.

Listing 12: Simple image loaded from file system

```
1  // define page style
2  wxReportPageStyle pgs;
3  pgs.SetWidth( 210 );
4  pgs.SetHeight( 297 );
5  pgs.SetMargins( 10, 10, 30, 30 );
6
7  m_Report.SetPageStyle( pgs );
8
9  // define image style
10 wxReportImageStyle is;
11 is.SetBorder( wxRP_ALLBORDER, wxColour(100, 100, 100),
       0.75 );
12
13 // create image item with custom PPI and apply custom
       style on it.
14 wxReportImageItem img;
15 img.SetPath( "../utb.png" );
16 img.SetPPI( 200 );
17 img.SetPosition( wxRP_CENTER, wxRP_CENTER );
18 img.SetStyle( is );
19
20 m_Report.AddItem( img );
```

The Listing 12 shows how to use custom image style defined at lines 10 and 11 and how to scale the source image by altering its

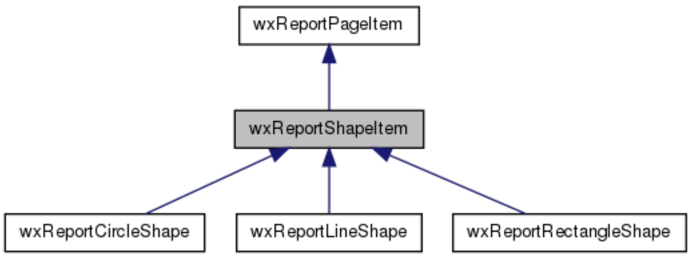


Fig. 11 hierarchy of shape items classes

PPI value as can be seen at line 16. Also, built-in position marks `wxRP_CENTER` are used for centering of the image within the main page area. Output of this source code can be seen in Figure 10.

D. *Static graphics*

Another way how to improve printed document's look is to use basic static graphic primitives like lines, rectangles and circles for highlighting of important parts of the document. For that purpose, the *wxReportDocument* library provides in addition to the *image item* also set of classes encapsulating these graphic objects which may be added to the document page. In contrast to the *image item* no position marks like `wxRP_LEFT` or `wxRP_CENTER` can be used for the placement - only absolute positioning is allowed there.

All static graphic primitives are inherited from common base class `wxReportShapeItem` defining API functions used for styling of the item. The shape base class allows definition on border/line color, line thickness and style and also definition of fill color if applicable.

Rectangles Probably the most common graphic primitive used in various printed documents and forms is *rectangle item*. This document item is encapsulated by `wxReportRectangleShape` and allows user to define its position, size, border and fill color. The usage of this class is straightforward and is illustrated in the following Listing 13.

Listing 13: Definition of rectangle shape

```
// define document page style
wxReportPageStyle pgs;
pgs.SetWidth( 210 );
pgs.SetHeight( 297 );
pgs.SetMargins( 10, 10, 30, 30 );

m_Report.SetPageStyle( pgs );

// create rectangle shae
wxReportRectangleShape rect;
rect.SetTopLeftCorner( 0, 0 );
rect.SetWidth( 190 );
rect.SetHeight( 237 );
rect.SetLineColor( *wxBLUE );
rect.SetFillColor( *wxGREEN );
rect.SetLineThickness( 4 );

// add the shape to the page
m_Report.AddItem( rect );
```

Circles Circles can be created and added to the document page by using `wxReportCircleShape` as shown in Listing 14. The *circle item* uses the same positioning and styling policy like the *rectangle item.*

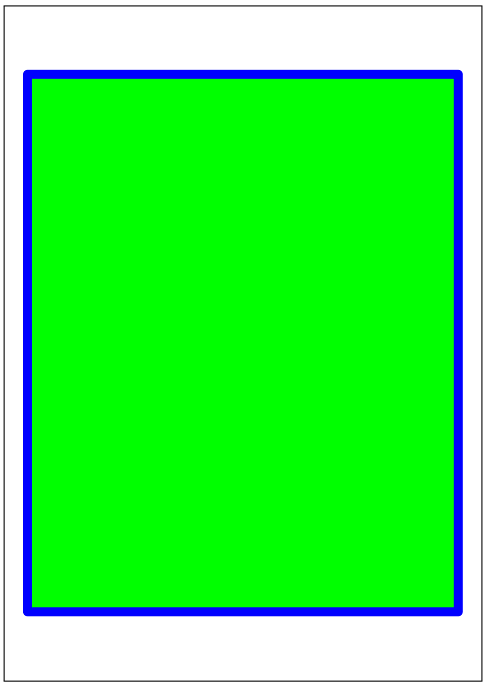


Fig. 12 Rectangle shape

Listing 14: Definition of circle shape

```
// define document page style
wxReportPageStyle pgs;
pgs.SetWidth( 210 );
pgs.SetHeight( 297 );
pgs.SetMargins( 10, 10, 30, 30 );

m_Report.SetPageStyle( pgs );

// create circle shape
wxReportCircleShape circle;
circle.SetCentreCoord( 95, 95 );
circle.SetRadius( 95 );
circle.SetLineColor( *wxBLUE );
circle.SetFillColor( *wxGREEN );
circle.SetLineThickness( 4 );

// add the shape to the page
m_Report.AddItem( circle );
```

Lines The last static graphic item supported by the library is *line shape*. As expected, the item allows users to define lines with specified staring and anding point, color and style. The class encapsulating this item is `wxReportLineShape` and can be used like demonstrated in Listing 15.

Listing 15: Definition of line shapes

```
// define page style
wxReportPageStyle pgs;
pgs.SetWidth( 210 );
pgs.SetHeight( 297 );
pgs.SetMargins( 10, 10, 30, 30 );

m_Report.SetPageStyle( pgs );

// create line shape and specify its style
wxReportLineShape line;
line.SetLineColor( *wxRED );
line.SetLineThickness( 2 );

// add a line leading from top-left to right-bottom page corner
line.SetPoints( wxRealPoint(0, 0),
    pgs.GetSize() - wxRealPoint( pgs.GetLeftMargin() + pgs.GetRightMargin(),
```

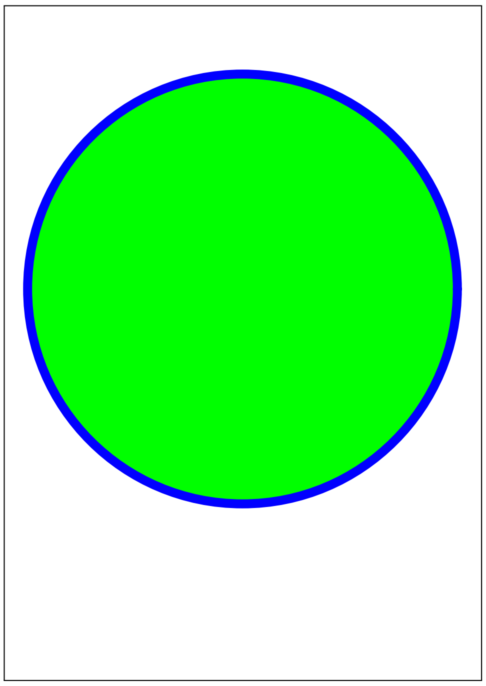


Fig. 13 Circle shape

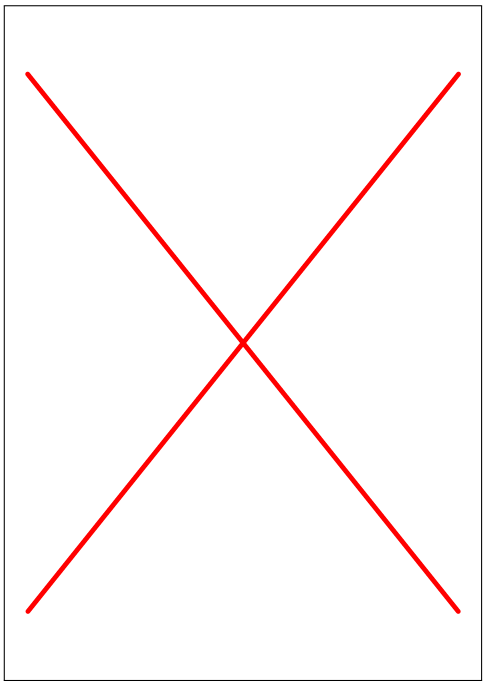


Fig. 14 Line shapes

```
                        pgs.GetTopMargin() + pgs.GetBottomMargin()
                                 ) );
    m_Report.AddItem( line );

    // add a line leading from top-right to left-bottom page
           corner
    line.SetPoints( wxRealPoint(pgs.GetSize().x - pgs.
           GetLeftMargin() - pgs.GetRightMargin(), 0 ),
                wxRealPoint(0, pgs.GetSize().y - pgs.GetTopMargin
                     () - pgs.GetBottomMargin() ) );

    m_Report.AddItem( line );
```

E. *Custom page items*

In some cases it is requested to print also user-defined, customized graphic output like various charts, diagrams, plots, etc. For that purpose a custom page item can be created by a user. Any of available page item classes provided by the library can be used as a base class for the custom page item. It is completely up to the user which one will be chose for his specific needs.

Let us to discuss the *custom page item* creation process on simple example allowing users to print custom graphic output. In this case `CustomReportPlot` class encapsulating the custom page item is derived from native *rectangle item* so it inherits all its public and protected members. Also, the custom class overrides virtual function called `DrawToDC()` responsible for drawing the custom output to output device context. Declaration of the class is shown in Listing 16.

Listing 16: Declaration of custom page item's class

```
#include <wx/report/reportdocument.h>

class CustomReportPlot : public wxReportRectangleShape
{
public:
   // set this item type to CUSTOM
   CustomReportPlot() { m_iType = wxRP_CUSTOM; }
   // override virtual function drawing into output
          canvas
   virtual void DrawToDC(wxDC* dc, bool toScreen, const
          wxReportPageStyle& pageStyle);
};
```

It is important to set the item's type to `wxRP_CUSTOM` because custom items are handled by the library in different way as compared with native page items. For example, the library takes ownership of the custom object so the user cannot delete it explicitly by using `delete` operator. Also, custom page items cannot be serialized to XML files.

Next Listing 17 shows how the user can use inherited members in conjunction with his own implementation code to perform desired graphic output.

Listing 17: Implementation of custom graphic output

```
void CustomReportPlot::DrawToDC(wxDC* dc, bool toScreen,
       const wxReportPageStyle& pageStyle)
{
   // calculate needed dimensions and coordinates
   int lineWidth = MM2PX( GetLineThickness(), dc, toScreen)
       ;
   int x = MM2PX( GetTopLeftCorner().x + pageStyle.
        GetLeftMargin(), dc, toScreen );
   int y = MM2PX( GetTopLeftCorner().y + pageStyle.
        GetTopMargin(), dc, toScreen );
   int w = MM2PX( GetWidth(), dc, toScreen );
   int h = MM2PX( GetHeight(), dc, toScreen );
   int ws = w / 15;
   int hs = h / 15;

   // call original drawing function
   wxReportRectangleShape::DrawToDC( dc, toScreen,
        pageStyle );

   // draw custom content over the original
   for( int i = 0; i < 15; ++i )
     dc->DrawLine( x, y + i * hs, x + i * ws, y + h -
          lineWidth );
}
```

Finally, the new custom page item can be created and added to the document page in the following way:

Listing 18: Creation of custom page item

```
// define page style
wxReportPageStyle pgs;
pgs.SetWidth( 210 );
pgs.SetHeight( 297 );
```

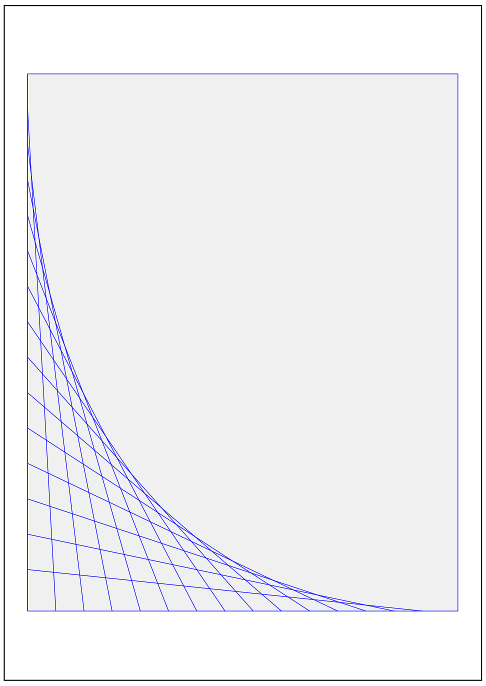


Fig. 15 Custom page item

```
pgs.SetMargins( 10, 10, 30, 30 );

m_Report.SetPageStyle( pgs );

/* create custom page item inherited from native rectangle
   item
   and set its properties */
CustomReportPlot *plot = new CustomReportPlot();
plot->SetTopLeftCorner( 0, 0 );
plot->SetWidth( 190 );
plot->SetHeight( 237 );
plot->SetLineColor( *wxBLUE );
plot->SetFillColor( wxColour( 240, 240, 240 ) );
plot->SetLineThickness( 0.3 );

/* add the custom item to the document page which
   takes its ownership (do not delete it
   explicitly) */
m_Report.AddItem( plot );
```

After printing, the custom output looks like shown in Figure 15.

VII. Persistent reports and forms

Another useful functionality provided by *wxReportDocument* library in an ability to serialize defined document into XML files and load it back anytime later. For that, a small set of API functions is declared in `wxReportDocument` class.

For serialization, `SaveLayoutToXML()` function can be used. By using this function all document pages including their layout, data and styling can be serialize to XML file stored on a file system or send to any suitable output stream supported by wxWidgets library.

Serialized data can be loaded back by using `LoadLayoutFromXML()` function which can read data directly from file system or from any available input stream.

When needed, current document content can be cleared completely by using `DeleteAllPages()` function.

Typical usage of I/O API functions mentioned above is shown in Listing 19

Listing 19: Document persistence API

```
// save document to XML file with "rpt" extension
wxString path = wxSaveFileSelector( "report", "rpt" );
if( path != "" ) {
  m_Report.SaveLayoutToXML( path );
}

// clear curent document
m_Report.DeleteAllPages();

// load saved data back to the document
wxString path = wxLoadFileSelector( "report", "rpt" );
if( path != "" ) {
  m_Report.LoadLayoutFromXML( path );
}
```

VIII. Conclusion

As shown in the paper, *wxReportDocument* add-on to wxWidgets GUI toolkit can be regarded as fully functional, production-ready library helping users to define, print and even store various documents and forms easily. The library is published unde wxWidgets License which means there are no legal restrictions for using the library so it can be used for both open-source and closed-source projects freely. Thanks to that the library has been already used in two commercial project developed at our university which proved its maturity.

At the moment, an active development of wxReportDocument library continues by implementation of visual forms designer based on CodeDesigner RAD tool [4] allowing users to define structure of printed forms in easy, interactive, non-programming way.

Source code of the library can be obtained from wxCode source repository [5] and can be build against wxWidget 2.8.x and wxWidgets 3.0.x GUI toolking by using any of supported compiler (e.g. GCC, MinGW, MS VC++, ...). In addition to the sources also Doxygen-based documentation [8] is available at the same location.

References

[1] *Did you know that Intel VTune used wxWidgets?* , wxBlock Website. (2012). [Online]. Available: http://docs.wxwidgets.org/3.0/classwx_html_easy_printing.html/af788066cba7ec33fe89bb66475729500

[2] M. Bližňák, T. Dulík, V. Vašek, "wxShapeFramework: An easy way for diagrams manipulation in C++ applications", *WSEAS Transactions on Computers* , vol. 9, issue 3, March 2010, pp. 268-277

[3] M. Bližňák, T. Dulík, V. Vašek, "A persistent cross-platform class objects container for C++ and wxWidgets", *WSEAS Transactions on Computers* , vol. 8, issue 5, 2009, pp. 778-787

[4] M. Bližňák, T. Dulík, R. Jašek, "Production-Ready Source Code Round-trip Engineering", *NAUN INTERNATIONAL JOURNAL OF COMPUTERS* , issue 3, vol. 6, 2012, pp. 158-169

[5] *wxCode*. (2015). wxReportDocument Component. [Online]. Available: http://wxcode.sourceforge.net/

[6] *wxWidgets Website*. (2015). [Online]. Available: http://wxwidgets.org/

[7] *wxHtmlEasyPrinting Class Reference*, wxWidgets Website. (2015). [Online]. Available: http://docs.wxwidgets.org/3.0/classwx_html_easy_printing.html/af788066cba7ec33fe89bb66475729500

[8] *Doxygen Website*. (2015). [Online]. Available: http://www.stack.nl/ dimitri/doxygen/

[9] J. Smart, "Cross-platform GUI programming with wxWidgets", Upper Saddle River: Prentice-Hall, 2006.