ReportLib

By Adiuvaris

PHP library to create PDF reports

https://www.adiuvaris.ch/reportlib

Table of content

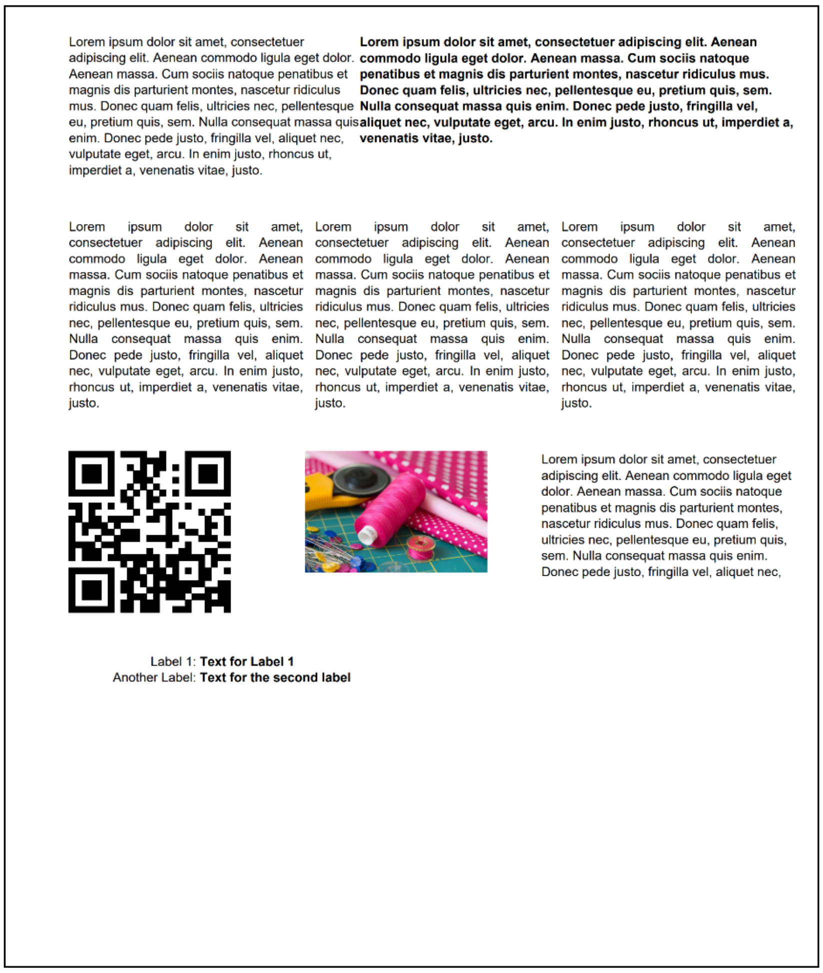
1 Overview

ReportLib is a PHP library to generate dynamic PDF reports using the *TCPDF* library to create a PDF document.

The library works with nested rectangular regions on the paper where you can define the sizes in millimeters or in percent of the surrounding rectangle or the library can calculate them based on the content. Each rectangle can have a different kind of content (e.g., some text, an image, a barcode).

1.1 A first Example

The following example shows the result of a dynamic report. I have defined the widths of the parts in percent of the surrounding rectangles, which in this case is the printable width of the paper. The first two text blocks have a width of 40% and 60% with no padding. Then follows a row with three text blocks with 33.3% width each and a padding of 4mm. Then follows some other content types: *https://www.adiuvaris.ch/example-26*.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec,

Label 1: **Text for Label 1**
Another Label: **Text for the second label**

Figure 1 Output of example 26

1.2 Basic Concept

The *ReportLib* offers classes and functions to create reports in a straightforward way. You do not have to take care about line and page breaks or about positions of content.

The calculation of the sizes and positions of text and other elements on the report is part of the *ReportLib*. As a user of the classes, you can control the output by using the different frame types and their attributes and functions.

1.2.1 Frames

Everything on a report is inside a frame i.e., a rectangular region on the paper. A frame has an upper left corner and a lower right corner. The library will calculate these values based on the definitions of the frame and/or the content. You can define the width in millimeters or in percent of the surrounding frame or you leave it to the *ReportLib* to calculate them.

Some frames (container frames) may contain other frames. The sizes of the inner frames define the size of the surrounding frame, whereas the outer frame may limit the size of the inner frames.

The whole report has internally a tree structure of different nested frames. During the output, the library will process the tree recursively.

As you can see the basic idea is quite simple but because the structure is recursive it can get complex.

1.2.2 Example of report structure

The following figure shows the structure of a simple report. Firstly, there is the page layout (page format and page margins) which defines the size of the pager and the printable area of the report body. The body is a vertically organized container and contains two horizontal containers. Each of them contains then two text frames.

The border lines serve only for a visualization of the structure. If you do not define any margins or paddings the frames have no distance between them as the image may imply.

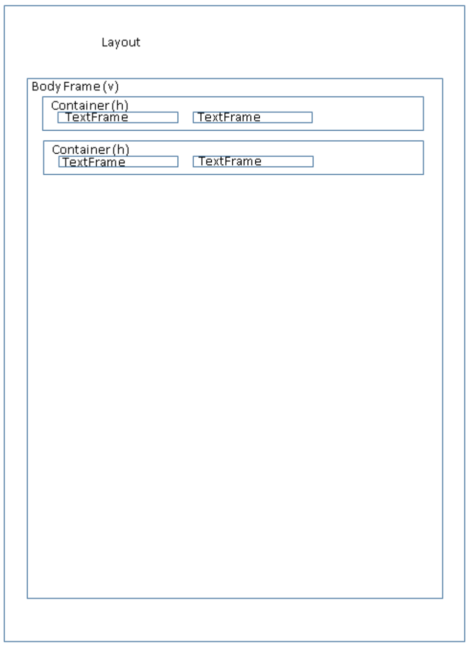


Figure 2 Report frame structure

1.2.3 Simple frame types

These frame types have content (e.g., some text, an image et cetera) and their size and position can be calculated directly in one step.

- *LineFrame*: A simple line on the report. It can have different length, extents, colors, and formats
- *TextFrame*: A frame with text and a text style.
- *ImageFrame*: A picture
- *TableFrame*: A feature rich table frame with columns and rows of data
- *BreakFrame*: A manual page break which can change the page format from the next page on.
- *BarcodeFrame*: A barcode (e.g., a QR code)

1.2.4 Container frame types

A container frame is a special kind of frame which can hold a list of other frames. The library processes the elements of a container frame as a unit to calculate the size and position of the container frame. A container can be organized horizontally or vertically.

- *SerialFrame*: A container frame to hold a list of other frames. They can be organized horizontally or vertically.
- *BoxFrame*: A rectangular region on the report which can hold one other frame. It can have a border, a background color, and paddings to format its content
- *PageFrame*: A container frame that the library prints on every page or only on certain pages.
- *FixposFrame*: A frame that the library prints on an exact position on the paper. It can overlay other frames.

I will describe all frame types in detail later in this document.

1.2.5 General attributes for frame types

Every frame type has a list of attributes which defines its individual character. The following list shows a few attributes which may be uses by multiple frame types.

- Margins (left, top, right and bottom).
- Flags if the frame may use the full width or height of the surrounding frame.
- Alignment (horizontal and vertical).
- Maximal sizes
- Flag if a frame should keep all content on one page.
- Background color
- Lines around a box
- Directions (horizontal or vertical)

All these attributes I will describe in detail later in this document when discussing the different frame types.

1.3 Text styles

The library uses text styles to define the font and the appearance of text in a report. There is a list of predefined text styles which you can use in a report. A text style consists of a font family (e.g., Helvetica, Courier et cetera), some font styles (bold, italic, underline) and a size in points. A text style also defines a text color and a background color.

Text styles have a reference to a default style which acts as a basis for the text style from where the text style inherits attributes. If you do not change any attributes a text style will look exactly the same as its base text style.

The default basic text style (NORMAL) uses the font family *Helvetica* and has a size of nine points. The text color is black, and the background is white.

The class *TextStyles* contains the following text styles which all are based on the text style *NORMAL*. The size attribute can be defined relative to the base style as a delta value in points (positive or negative).

- *SmallNormal*: one point smaller
- *Heading1*: nine points taller and bold
- *Heading2*: six points taller and bold
- *Heading3*: three points taller and bold and italic
- *Heading4*: one point taller and bold and italic
- *Bold*
- *SmallBold*: one point smaller
- *Italic*
- *Underline*
- *Footer*: one point smaller
- *Header*: like NORMAL
- *TableHeader*: one point smaller and bold
- *TableRow*: one point smaller
- *TableSubTotal*: one point smaller and italic
- *TableTotal*: one point smaller and bold

As said text styles uses a base style and can overwrite some attributes. Therefore, a change on the base style can change all other styles. That means if you change the font family in the base style, all the other styles inherit this setting and will use the other font (as long you do not overwrite it).

You can define new text styles which are based on any other text style. To access these text styles, you will use their name. I will describe the usage of text styles attributes in detail later in this document.

1.4 Example programs

The installation of the ReportLib contains about 30 example programs which shows the usage of the classes of the library. The programs are in the folder examples of the installation.

You can find the examples also on the site *https://www.adiuvaris.ch/reportlib*. You will find references to examples when they illustrate the current topic in this document.

2 Classes in the ReportLib

The class *ContainerFrame* offers a series of convenience functions (see 2.18) which it allows you to create a report without any *new* operation in the code. This although the *ReportLib* works highly dynamic. You can of course use all base classes and functions to build the report, or you can mix the base classes and the convenience functions.

In the following sections you find the description of the classes in the *ReportLib*. At first there are all the classes which you will need to build the report structure and some abstract base classes from which the frame classes inherit functionality. At the end of the chapter, you can find a description of some internally used classes.

Most of the classes that you need to create a report are set up so that you create an instance of the class with some parameters in the constructor and then you can set more attributes with setter functions. You have to add the created objects to the hierarchical structure, normally via the function *addFrame*.

2.1 Units of measurement

All sizes, extents and positions in the library functions are expected in millimeters as float values. Some frame types allow it to define the width in percent of the surrounding frame.

In these cases, the widths are expected as strings e.g., “35.0%”. The percent sign is not necessary but makes your code more readable, and it is clearer what you mean. The parameters for these functions are defined as mixed and if a string arrives it will be managed as percent value.

2.2 Namespace

All classes of the *ReportLib* are in the namespace *Adi\ReportLib*. You can add something like the following code at the top of your program files to access the *ReportLib*. You have to adjust the path to the library.

```
include_once "../src/Report.php";

use Adi\ReportLib as ReportLib;
```

After that you can access the classes by *ReportLib\classname.*

2.3 TCPDF

As mentioned earlier the *ReportLib* library uses the *TCPDF* library to create PDF documents. Therefore, some of the parameters and values in this library will be inherited from the *TCPDF* library. All code which accesses the *TCPDF* library is encapsulated in the class *Renderer*.

2.3.1 Fonts

This library can use the font families that are available for *TCPDF*. There are three core font families *Helvetica*, *Courier,* and *Times* that you can use in the *ReportLib*. It is possible to add more fonts to the *TCPDF* library which you may use in the *ReportLib*. You can find articles in the web how to do that.

2.3.2 Page sizes

You can use all defined page sizes in *TCPDF* for reports that you generate with the *ReportLib* (e.g., 'A4', 'B4', 'Letter' et cetera). The list of the possible page formats can be found in the static array *$page_formats* in the module *tcpdf_static.php* in the include folder of the *TCPDF* installation.

2.3.3 Colors

Colors in this library are strings containing the RGB hex values as in HTML colors. To make it more visible that a string is a color you should add an '#' at the first position of the string but it is not necessary. The characters in the string can be uppercase or lower case.

```
$red = "#FF0000";
```

TCPDF has a list of named colors (defined in the module *tcpdf_colors.php*) which you can use via the function *getColorByName* in the *Report* class. If you pass a valid color name, the function will return a color hex string that you can use for any function in the *ReportLib* that needs a color. If the name does not exist, the function will return a color string for black (i.e., "#000000").

```
$color = ReportLib\Report::getColorByName("gold");
```

The example above returns "#ffd700";

2.3.4 Alignments

TCPDF uses the following characters for alignments and the *ReportLib* expects the same values.

Horizontally (hAlignment)

- 'L' Left
- 'R' Right
- 'C' Center
- 'J' Justified (only for text)

Vertically (vAlignment)

- 'T' Top
- 'M' Middle
- 'B' Bottom

So, whenever a function needs a parameter of *hAlignment* or *vAlignment* use one of these letters.

2.4 Class Report

See example – https://www.adiuvaris.ch/example-1

This class manages the printable area on the pages, and it prints the pages into the PDF document during the output operation. It contains a *SerialFrame* (see section 2.14) for the body of the report as well as one for the header and for the footer. To build up a report you can add frames with content to the body. The body is the outermost frame in your report.

To create a PDF document, you have to call the *output* function after you have added all content to the report structure. The library will then loop recursively over the structure to calculate sizes and positions. If you need the total number of pages in the report (e.g., in the footer or header) the *ReportLib* will use two passes to count first the number of total pages. If you need that, then you have to force it by calling *setCountPages*.

This is the first class that you have to instantiate if you want to create a report structure. The constructor has a parameter which defines the page format that the library will use for the report (class *PageFormat*). If you do not provide this parameter a default *PageFormat* will be used.

```
// Create report instance
//  default format A4, portrait with margins left = 20mm, top = right = bottom = 10mm
$report = new ReportLib\Report();
```

Now the report structure is prepared, and you can append content to it to create the report. Therefore, you can get a reference to the report body frame which is a vertical organized *SerialFrame*. I describe Serial frames later in this document.

If you want to add content to the report body, you just get a reference of it from the report object and call the *addFrame* function to add the content frame. That may look like in the following code excerpt:

```
// Create report instance
$report = new ReportLib\Report();

// Get ref to the report body
$body = $report->getBody();

// Get the default text style
$tsNormal = ReportLib\TextStyles::getTextStyle(ReportLib\TextStyles::NORMAL);

// Create a text frame
$tf = new ReportLib\TextFrame("Text to print", $ts);

// Add the text to the body
$body->addFrame($tf);
```

This example just adds some text in a *TextFrame* to the report body.

2.4.1 Class PageFormat

See example – https://www.adiuvaris.ch/example-2

This is a structure that holds the information for a page or a set of pages in the report. It defines a page size like *A4* or *Letter* and the orientation of the paper (portrait or landscape). Furthermore, it limits the printable area by the page margins. It also contains a flag which defines if the left and right margin are mirrored on even and odd page numbers.

The following code shows how to create a *PageFormat* for a Letter sized paper in landscape mode. The left margin is one inch, and the others are half an inch. The last parameter determines that the library will mirror the left and right margins. You can use this instance to pass it to the constructor of a new report.

```
// Format letter, landscape with margins top = 1 inch and half an inch on the other
sides
$pageFormat = new ReportLib\PageFormat("Letter", 'L', 25.4/2.0, 25.4, 25.4/2.0,
25.4/2.0);

// Create report instance with the pageFormat
$report = new ReportLib\Report($pageFormat);
```

You can find the possible page sizes in the *TCPDF* library (see section 2.3.2). The letters for the orientation are straight forward the first character of the respective name: 'P'ortrait or 'L'andscape. The margins parameters are in millimeters.

2.4.2 Body, Header, and Footer

See example – https://www.adiuvaris.ch/example-29

The *Report* object holds the whole structure of the report. You can add content to the body by getting the reference from the *Report* object with the function *getBody*. The body is a vertically organized *SerialFrame* (see 2.14). When you add content to the body the library will print these frames one below the other.

Besides the body there is also a *SerialFrame* for a header and for a footer which you can access via *getHeader* and *getFooter* functions in the *Report* object. If you add content to the header and/or the footer their heights will reduce the printable area on the page.

You can add any frame type to the header and footer but normally you will use a *PageFrame* (see 2.16) as the main container. With this frame type you can define on which pages the library should print the header and footer content.

In the following example I add a simple vertical container to the footer and add a box with some text. The text is centered on the bottom of the page.

```
$tsBold = ReportLib\TextStyles::getTextStyle(ReportLib\TextStyles::BOLD);

$report = new ReportLib\Report();
$footer = $report->getFooter();

$vc = $footer->AddVContainer();
$vc->setMarginTop(5.0);
$box = $vc->AddBox();
$box->setUseFullWidth(true);
$tf = $box->AddText("Adiuvaris    -    At the lake 901a", $tsBold);
$tf->setHAlignment('C');
$tf->setVAlignment('B');
```

2.5 Class TextStyles

See example – https://www.adiuvaris.ch/example-5

You can find the list of the predefined text styles earlier in this document (see 1.3). These text styles are in a static class called *TextStyles*. You can use them at any point in the code where you need them. The text styles are instantiated by the first access to a text style. In the next section you will find details about text styles.

To use a text style, you can get a reference to one of them using the static function *getTextStyle* in the class *TextStyles*. The function needs the name of the text style as a parameter. You should use the constants in the *TextStyles* class for the names of the predefined text styles.

```
// Get the default text style
$tsNormal = ReportLib\TextStyles::getTextStyle(ReportLib\TextStyles::NORMAL);
```

The text styles have a hierarchical structure i.e., a text style can have a base text style and in the new text style some attributes can be changed (e.g., the font family or the bold flag). The text style remembers which attributes you have changed and uses them. For the other attributes it uses the values from the base text style. When there is no base text style it will use the *NORMAL* text style as a fallback.

You can create new text styles and give them a name to use it at any point in the program like the predefined text styles. You define a new text style by calling the static function *addTextStyle* in the *TextStyles* class. The function only expects a name for the new text style and a base text style.

If you try to add a text style with a name that already exists no new text style is added but you get back a reference to the existing text style with the given name as if you would have called *getTextStyle*. Otherwise, the call of this function returns a reference to the new text style. You can use this return value to change the attributes you want.

The following example shows how to create a new text style based on the NORMAL text style. The new text style gets the name *BigRed* and changes the attribute for the size to a value of 36 points and the text color to red. The library uses for all the other attributes like the font family the values from the NORMAL text style.

```
// Get the default text style
$tsNormal = ReportLib\TextStyles::getTextStyle(ReportLib\TextStyles::NORMAL);

// Create a text style with the name "BigRed", 36 points tall and red
$ts = ReportLib\TextStyles::addTextStyle("MyTextStyle1", $tsNormal);
$ts->setSize(36.0);
$ts->setTextColor("#FF0000");
```

I describe the attributes of a text style in the next section.

2.6 Class TextStyle

See example – https://www.adiuvaris.ch/example-7

The text styles control the format of text output. Text styles define the font as well as the text color and the background color. There are getters and setters for all of the attributes of a text style. You can define the following attributes in a text style:

- name
- bold
- italic
- underline
- sizeDelta – relative change of the size in terms of the base text style (positive or negative)
- size – absolute size of the font in points
- fontFamily – the font e.g., *Helvetica*
- textColor
- backgroundColor

The default text style has a font size of nine points and uses the font family *Helvetica*. The text color is black, and the background is white. The flags bold, italic and underline are false and the sizeDelta value is zero.

There is a flag for almost every attribute in a text style which reflects if the attribute has been set or if the value of the base text style should be used. The only exception is *$sizeDelta* which value is used directly. Therefore, a getter function in the text style class looks like this.

```
public function isBold(): bool
{
    if ($this->boldSet) {
        return $this->bold;
    }
    return $this->defaultStyle != null && $this->defaultStyle->isBold();
}
```

First there is check if the value has been set, if yes, the function returns the value of the object otherwise it returns the value of the base style. In the constructor of a text style the flags all are set to false.

The flags will be set by the setter function as you can see in the following example.

```
public function setBold(bool $bold): void
{
    $this->boldSet = true;
    $this->bold = $bold;
}
```

You can reset all of these flags by calling the *resetToDefault* function for a text style.

2.7 Class ReportFrame

ReportFrame is an abstract base class which defines some functions and attributes which are inherited by concrete classes like *TextFrame*. I will describe these concrete classes in later sections of this document.

As said earlier in this document, a frame is a rectangular region on the paper that may have some content like text or an image or other things.

The class offers the following attributes. They have getters and setters.

- hAlignment, vAlignment – horizontal and vertical alignment
- marginLeft, marginTop, marginRight, marginBottom – margins on all four sides
- useFullHeight, useFullWidth – flags if the frame uses the whole possible space
- keepTogether – flag if the frame and all its content keep together on the same page
- maxWidth, maxHeight – maximal dimensions of the frame

These attributes are available in all the derived classes. They control the calculation of sizes and positions of frames.

ReportFrame offers a lot of functions that will be used by the *Report* class to calculate the size and position of the frame. Normally you must not use any of these functions.

2.7.1 Class ContainerFrame

This class is a derivation of the class *ReportFrame* an inherits all the attributes from it. There is only one additional attribute which can hold an array of frames. It is an abstract base class for the *SerialFrame* or *BoxFrame* classes which I describe later in this document.

This class offers all the necessary functions to manage the array of frames. The array can hold any kind of frame type which is derived from *ReportFrame*.

You can add a frame by calling the *addFrame* function. The new added frame is added at the end of the container. There are also functions to remove one or all frame or to get a certain frame (*getFrame*, *removeFrame* and *clearFrames*). Because a container frame can contain other containers you can create a highly recursive tree structure.

The library calculates the size of the frame by calculating recursively the sizes of all the inner frames.

2.8 Class TextFrame

See example – https://www.adiuvaris.ch/example-3

The *TextFrame* class is derived from *ReportFrame* and holds some kind of text. A *TextFrame* needs a text and a text style. It is a simple frame type with no other frames in it. You can add a text frame to any container frame using the *addFrame* function.

```
$report = new ReportLib\Report();
$body = $report->getBody();
$tsNormal = ReportLib\TextStyles::getTextStyle(ReportLib\TextStyles::NORMAL);
$tf = new ReportLib\TextFrame("Text to print", $ts);
$body->addFrame($tf);
```

The *ReportLib* tries to print the text into the given frame. If necessary, it inserts line breaks or even page breaks to fit the text into the report. The surrounding frame defines the width of the text. In the example the width is the printable area of the paper because the text is added to the main body frame of the report.

The text style controls the output of the text. But you can change the text color in this class by calling the function *setTextColor*.

By default, the library splits the text that does not fit into the given width of a frame into multiple lines, but you can define that the text should be cut off after one line. For that you have to set the *wordWrap* attribute to false.

The text may contain some variables which will be replaced during the output operation. You have to surround the variable names by square brackets. You can use:

- VAR_PAGE – the page number
- VAR_TOTAL_PAGES – the total number of pages in the document.

You can use these variables at any place, but they are often used in headers or footers.

```
$tsNormal = ReportLib\TextStyles::getTextStyle(ReportLib\TextStyles::NORMAL);
$tf = new ReportLib\TextFrame("Page [VAR_PAGE] of [VAR_TOTAL_PAGES]", $ts);
```

The renderer replaces the texts during the rendering, and it will keep the alignments correct. If you use the total number of pages, you have to inform the library to calculate the pages by calling *setCountPages* of your *Report* object.

2.9 Class BreakFrame

See example – https://www.adiuvaris.ch/example-9

The *BreakFrame* is derived from *ReportFrame*, but it holds no content, and it does not use the attributes of the *ReportFrame* class. You can use it to manually force a page break.

```
$bf = new ReportLib\BreakFrame();
$body->addFrame($bf);
```

This class offers one additional attribute of type *PageFormat* (see 2.4.1). With that you can change the page format that the library will use for the pages after the page break. The new page format keeps valid until you insert another manual page break with a new page format.

```
// Add a manual page break and change the settings to
//     A5 portrait and set the margins - mirror the left and right margins
$pageFormat = new ReportLib\PageFormat('A5', 'P', 20.0, 10.0, 10.0, 10.0, true);
$bf = new ReportLib\BreakFrame($pageFormat);
$body->addFrame($bf);
```

2.10 Class LineFrame

See example – https://www.adiuvaris.ch/example-8

This is another simple frame type. It can be used to add various kinds of lines to a report. You can add horizontal or vertical lines to the report by adding such an object to a container frame.

```
$lf = new ReportLib\LineFrame('H', 0.5, "#00FF00", 120.0);
$body->addFrame($lf);
```

These are the parameters to the constructor.

- direction – horizontal or vertical (‘H’, ‘V’)
- extent – Extent of the line in millimeters
- color
- length – length in millimeters

A *Pen* object (see section 2.10.1) prints the line. Therefore, you can change the look of the line by changing the line attributes. You can use solid or dotted or dashed lines by setting a pen to the frame.

```
// Print a centered dotted line, extent 0.2mm and of blue color
$lf = new ReportLib\LineFrame('H');
$pen = new ReportLib\Pen(0.2, "#0000FF", 'dot');
$lf->setPen($pen);
$lf->setLength(50.0);
$lf->setHAlignment('C');
$body->addFrame($lf);
```

2.10.1 Class Pen

Pen objects describes line styles. The *LineFrame* (as seen in the last section) but also the *Box-Frame* use pens. A pen contains a color, an extent (thickness of the line) and a line style (e.g., solid, dashed et cetera). You can create a new pen as follows.

```
$pen = new ReportLib\Pen(0.2, "#0000FF", 'dot');
```

You can pass all the attributes or none of them. In this case you will get a default pen which has an extent of 0.1mm and it is a solid black line. *TCPDF* supports self-definable line styles, but *ReportLib* accepts only the following values for the line style parameter:

- ‘solid’
- 'dash'
- 'dot'
- 'dashdot’

2.11 Class ImageFrame

See example – https://www.adiuvaris.ch/example-10

This class is derived from *ReportFrame*, and you can use it to add an image to the report. The image may be a PNG or a JPEG file (or anything supported by the *TCPDF* library). The file must be readable by the *ReportLib*.

```
$ifr = new ReportLib\ImageFrame("image.jpg", 100.0, 100.0, true);
$body->addFrame($ifr);
```

The parameters in the constructor are all the additional attributes for an *ImageFrame*. Besides the filename of the image, you can define the maximal width and height in millimeters and if the library will respect the aspect ratio when it prints the image.

If you do not pass a maximal width and height, the image will use the whole width and height of the parent frame. By default, the image keeps the aspect ratio. You can pass a maximal width or a maximal height or both. If you omit one of them, the image uses the corresponding size of the parent.

If the aspect ratio flag is false, the image will fill the calculated rectangle. If the aspect ratio flag is true, the image will use a rectangle as big as possible. Therefore, the frame can have white space around the picture depending on the alignments and the given sizes.

2.12 Class BarcodeFrame

See example – https://www.adiuvaris.ch/example-11

This class is derived from *ReportFrame*, and you can use it to add a barcode to the report. The barcode may be *QRCODE*, *PDF417* or *DATAMATRIX*. For QRCODE you can give an additional value which defines the level of error correction (L=low, M=medium, and H=high). You have to add the letter after a comma as you can see in the following example.

```
$bcfr = new ReportLib\BarcodeFrame("text in the qr code", "QRCODE,H", 50.0, 50.0);
$body->addFrame($bcfr);
```

The parameters in the constructor are the additional attributes for the *BarcodeFrame*. First there is the text of the barcode. Then follows the type of barcode. At last, you can pass a maximal width and height in millimeters for the barcode image.

If you do not pass a maximal width and height, the barcode image will use the whole width and height of the parent frame. You can pass a maximal width or a maximal height or both. If you omit one of them, the barcode frame uses the corresponding size of the parent.

The barcode type defines the form of the barcode image. *QRCODE* and *DATAMATRIX* codes uses squares and *PDF417* uses a rectangle shape. The code image uses as much space as possible in the calculated rectangle. Therefore, the frame can have white space around the code image depending on the alignments and the given sizes.

2.13 Class BoxFrame

See example – https://www.adiuvaris.ch/example-20

A *BoxFrame* is a special kind of *ContainerFrame*. It is derived from the class *ContainerFrame*, but it can manage only one frame in the container. This limitation is not very restricting because you can add a container frame.

A *BoxFrame* can have a border and a background color. You can specify a width and/or a height that the box may use in the report. The frame can have different paddings on all four sides. A padding reduces the possible size of the content in the box by adding some whitespace.

The width and the height can be set independently or not at all. If you do not set the width, then the content will define the width of the *BoxFrame*.

You will use *BoxFrame* and *SerialFrame* (see 2.14) a lot because they are very flexible objects to control the layout in a report.

The constructor of a *BoxFrame* has the following parameters

- width – can be zero
- height – can be zero
- borderExtent – the extent for a border line (default 0.0)
- borderColor – default black
- backgroundColor – default white

If you create a *BoxFrame* with no parameters in the constructor and you do not add any content to it, nothing will be printed because this box will not use any space. You have to define a width, or you have to add content to make a box visible.

The following example creates a red box in the report.

```
$box = new ReportLib\BoxFrame(70.0, 50.0, 0.1, "#FF0000");
$body->addFrame($box);
```

That may look like the following figure.

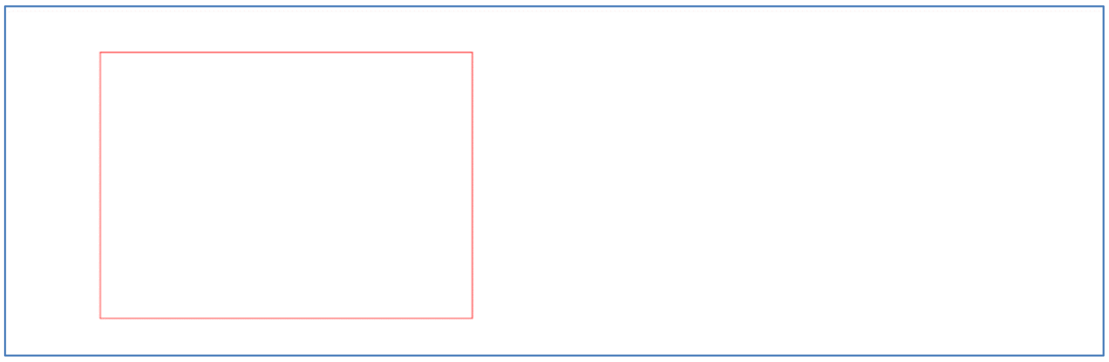


Figure 3 Simple BoxFrame example

The background will fill the whole box including the paddings. The next example defines a box with a black border and a yellow filling. The text will respect a padding of 5mm inside the box.

```
$bf = new ReportLib\BoxFrame(50.0, 20.0, 0.3);
$bf->setPadding(5.0);
$bf->setBackground("#FFFF00");
$body->addFrame($bf);

$tf = new ReportLib\TextFrame("This is text in a box.", $tsNormal);
$bf->addFrame($tf);
```

This example produces the following output

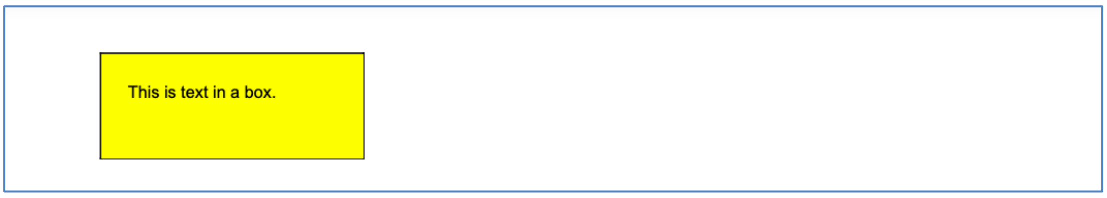


Figure 4 BoxFrame with background color and a padding

2.13.1 BoxFrame width in percent

See example – https://www.adiuvaris.ch/example-26

You can define the width and height of a *BoxFrame* in percent of the width of the parent frame. To do that you have to pass the width as string. You can even mix percent and absolute values as in the following example.

```
$bf = new ReportLib\BoxFrame("70.0%", 20.0, 0.1);
$bf->setPadding(1.0);
$body->addFrame($bf);

$tf = new ReportLib\TextFrame("This is text in a box.", $tsNormal);
$bf->addFrame($tf);
```

The example shows a box frame with 70% of the width of the parent frame and a height of 20mm with a thin black border. The box gets some example text in a TextFrame as content.

The output will look as follows.

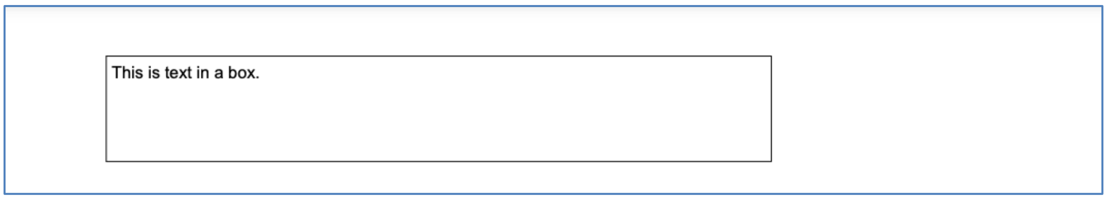


Figure 5 BoxFrame with relative width

2.13.2 Class Border

The *BoxFrame* uses the *Border* class to manage the border around the box. The border contains four different *Pen* objects for the four sides left, top, right and bottom. Therefore, it is possible to have four different line types (or no line at all) around a box frame.

You can set the same pen for all four sides or an individual pen for a certain side. The following example sets the pen for all sides and creates then some special pens for the top and the bottom sides by changing them in the border object.

```
$box = new ReportLib\BoxFrame(50.0, 10.0);
$box->getBorder()->setPen(new ReportLib\Pen(0.2, "#0000FF"));
$box->getBorder()->getTopPen()->setExtent(1.0);
$box->getBorder()->getBottomPen()->setExtent(3.0);
$box->getBorder()->getBottomPen()->setColor("#FF00FF");
$body->addFrame($box);

$tf = new ReportLib\TextFrame("This is text in a box.", $tsNormal);
$box->addFrame($tf);
```

The output looks as follows.

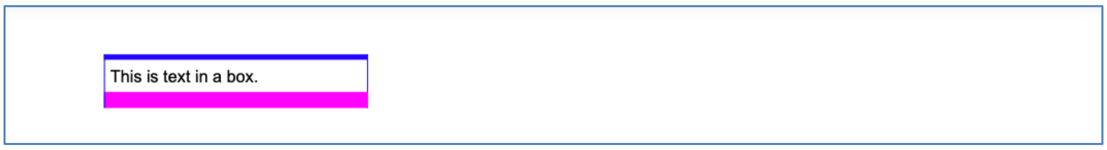


Figure 6 BoxFrame with special border

2.14 Class SerialFrame

See example – https://www.adiuvaris.ch/example-22

This is a derivation of the *ContainerFrame* class which can contain a series of frames. A *SerialFrame* has a direction which is either *horizontal* or *vertical*. All frames may contain any kind of content or other container frames.

The following example shows how you instantiate a horizontal *SerialFrame*. For a vertical *SerialFrame* you obviously use the parameter 'V'.

```
$sf = new ReportLib\SerialFrame('H');
$body->addFrame($sf);
```

There is only one additional attribute for this class which defines how the library places the frames on the page. Either it can be horizontally i.e., one frame right to the previous one or vertically which means the frames are one below the other.

The *SerialFrame* inherits the *addFrame* function from the *ContainerFrame* which you can use to add content or other container frames to the frame.

As mentioned in the last section you will use the *BoxFrame* and the *SerialFrame* very often to create a report. Because you can add containers into containers you can create complex tree structures. But it is also a pitfall because you can create endless recursive structures when you add the wrong object to a frame – a circular dependency. The *ReportLib* checks that when you try to create output of your report structure and throws an exception if a circular dependency is detected.

2.15 Class FixposFrame

See example – https://www.adiuvaris.ch/example-23

The library places the content of this container on an absolute position on the page. When you create such an object, you have to define an offset in millimeters from the left and the top border of the paper. These coordinates define the top left corner of the frame. You can add a third parameter in the constructor. This parameter defines if the library is allowed to overlay other frames with this one. The default for the overlay flag is false.

The following code creates a *FixposFrame* and adds a *BoxFrame* to it. The library places the top left corner of the box at the coordinates 60mm from the left side and 130mm from the top of the paper. It will not print over other frames and create a page break if other content is in the way.

```
$fix = new ReportLib\FixposFrame(60.0, 130.0);
$box = new ReportLib\BoxFrame();
$box->setWidth(100.0);
$box->setHeight(50.0);
$box->setBorderPen(new ReportLib\Pen(0.1, "#CCCCCC"));
$fix->addFrame($box);
$body->addFrame($fix);
```

The library prints the content of this container at the defined position (top left corner). But it will normally not overwrite any existing content. If the space that is needed for the content of a *FixposFrame* overlaps with any other content on the page, the library will add a page break.

If you want to print a content on any case at a certain spot you can add the third parameter with the value *true*. This is also necessary if you want to define offsets that are outside of the printable area.

2.16 Class PageFrame

See example – https://www.adiuvaris.ch/example-25

This is also a *ContainerFrame*. With this frame type you can control on which pages the library prints the content of this frame. You can define that the content prints on a certain page or on odd or even pages or on all pages without the first page. You can use this kind of frame type in headers and footers if you want different headers and/or footers on even and odd pages, but it is possible to use it anywhere.

The following code creates a *PageFrame* that is printed on all pages but not on the first page. The content is a box with a grey background using the full width of the printable area and a height of 15mm.

```
$pf = $header->AddPageFrame(ReportLib\PageFrame::C_OnAllButFirstPage);
$box = $pf->AddBox();
$box->setUseFullWidth(true);
$box->setHeight(15.0);
$box->setBackground("#EEEEEE");
```

The first parameter of the constructor is the page number on which the content has to be print or one of the following values. You find them in the class *PageFrame*.

```
const C_OnAllPages = 0;
const C_OnOddPages = -1;
const C_OnEvenPages = -2;
const C_OnAllButFirstPage = -3;
```

2.17 Class TableFrame

See example – https://www.adiuvaris.ch/example-12

A *TableFrame* is a simple frame type derived from *ReportFrame*. It has a lot of features to create tables in a report. It contains a sub structure (columns), but it is not a container. The *TableFrame* manages the columns of the table. You can add columns to the table and specify how the column will appear on the page.

The library prints a table header automatically (also on every new page), but it can be suppressed if no header is needed. The header row uses the title of the column as heading, so if you leave the title of a column empty there is no text in the header for that column.

The width of the table is the sum of the widths of the columns. There are multiple ways to define the widths of columns and it is possible that the table automatically uses the width of the parent frame. In this case the library enlarges the columns relative to their original sizes.

The lines in the table can be set individually. You can define a border around the table, horizontal lines between rows and/or vertical lines between columns. The lines use *Pen* objects and it possible to print all the lines with different pens. There are separate pens for the line below the header and above a total row and the other horizontal lines between rows.

The table manages the data in rows. Each row has an array of text for the columns of the table. It also has a row type which defines the text style that the library uses to print the data of the row.

- 'H' for the header row
- 'D' for detail rows
- 'S' for subtotal rows
- 'T' for total rows

There are predefined text styles for the different row types. You can change the text styles as you need them, or you can create new text styles and set them in the *TableFrame* object.

2.17.1 Class TableColumn

Often you do not need to change anything directly in objects of this class because the columns are managed by the *TableFrame*. But there are some special attributes which are not accessible via the *TableFrame* (e.g., the right pen or the paddings). There are getters and setters for the attributes of the class.

- columnName – the name of the column, it has to be unique for a table. You will use it to add data to the table.
- title – the title text for the header row
- lineBreak – flag if there is line break after the column
- rightPen – the pen for vertical lines right to the column (has no effect for the last column if there is a border around the table)
- sizeWidthToContents – flag if the width of the column has to be set to the calculated maximum width of the content of that column.
- sizeWidthToHeader – flag it the width of the column has to be set to the width of the title text
- hAlignment – horizontal alignment (default left)
- vAlignment – vertical alignment (default top)
- paddingLeft, paddingRight, paddingTop, paddingBottom – paddings in the cells of the table.

The function to add columns to a table returns a reference to the newly added column. With this reference you can modify the column. The next example code shows how you can do it. The pen right of the column is set to 0.1mm with a grey color and there is a padding on the right side of 2mm.

```
$tcol = $table->addColumn("frametype", "Frame type", 40.0);
$tcol->setRightPen(new ReportLib\Pen(0.1, "#C0C0C0"));
$tcol->setPaddingRight(2.0);
```

The next section shows how to add columns to a table frame and how you can adjust them.

2.17.2 Create a table and its columns

First you have to create an instance of a *TableFrame* object and add that to a container frame (in the example the main body is the container). Then you can set special attributes of the *TableFrame*.

```
// Add the table to the report body
$table = new ReportLib\TableFrame();
$body->addFrame($table);

// Adjust the table
$table->setColumnLines(true);
$table->setInterRowSpace(1.0);
$table->getBorder()->getTopPen()->setExtent(1.0);
$table->getBorder()->getBottomPen()->setExtent(1.0);
$table->getBorder()->getBottomPen()->setColor("#0000FF");
```

In this example the *TableFrame* gets a border line on the top and on the bottom. The lines have an extent of 1mm, and the top line will be black, and the bottom line will be blue. The library prints vertical lines between columns and there will be white space of 1mm between the rows.

Here is a list of attributes that you can use for table frames.

- columnLines – flag which defines if the library prints vertical lines (default false)
- maxHeaderRowHeight – maximum height of the header (default 100mm)
- maxDetailRowHeight – maximum height of a detail row (default 100mm)
- marginBottomSubtotal – a margin that will be added on rows of type Total or Subtotal
- interRowSpace – additional whitespace between rows of data
- headerTextStyle – text style for header rows
- detailRowTextStyle – text style for detail rows
- alternatingRowTextStyle – text style for detail rows – if this text style is not null then the detail rows will print alternately with the text style for detail rows and this text style.
- subTotalRowTextStyle – text style for subtotal rows
- totalRowTextStyle – text style for total rows
- minDataRowsFit – minimal number of data rows that must fit on a page before the library adds a page before the table.
- border – a border object to draw a border around the table
- repeatHeaderRow – flag if the library prints the header row on every new page
- suppressHeaderRow – flag if the library prints a header row at all
- innerPenHeaderBottom – a pen for the line below the header
- innerPenTotalTop – a pen for the line above the total row
- innerPenRow – a pen for the lines between detail rows.

After you have defined the table, you can add columns as shown in the following example. The function *addColumn* needs only a unique column name, a title text, a maximum width, and a horizontal alignment. If the alignment is left, you can omit the last parameter.

```
// Add the table to the report body
$table = new ReportLib\TableFrame();
$body->addFrame($table);

// Add four columns to the table
$table->addColumn("frametype", "Frame type", 40.0);
$table->addColumn("container", "Container type", 30.0, ‘C’);
$table->addColumn("description", "Description", 60.0);
$table->addColumn("number", "Number", 20.0, ‘R’);
```

This simple example shows a table with four columns. The library prints the content of the column with the name *container* centered and the content of the column *number* right aligned. The widths of the columns are in millimeters. The table needs at least 150mm width.

This table would look like the following figure.

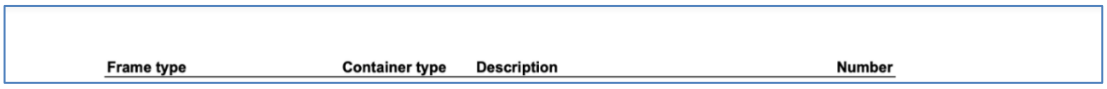


Figure 7 Table header

Because there were no data rows added only the header of the table appears in the output. But the alignments of the columns are obviously correct.

2.17.3 Use the full width

See example – https://www.adiuvaris.ch/example-13

The *TableFrame* inherits the attribute *useFullWidth* from the *ReportFrame*. If this value is set to true the table uses the width of the parent frame (e.g., the printable width of the page).

```
// Add the table to the report body
$table = new ReportLib\TableFrame();
$table->setUseFullWidth(true);
$body->addFrame($table);

// Add four columns to the table
$table->addColumn("frametype", "Frame type", 40.0);
$table->addColumn("container", "Container type", 30.0, ‘C’);
$table->addColumn("description", "Description", 60.0);
$table->addColumn("number", "Number", 20.0, ‘R’);
```

The library enlarges all columns relative to their original widths. The output of that looks like the following figure.

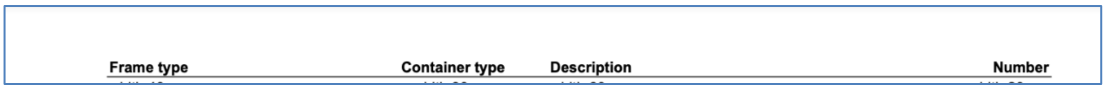


Figure 8 Table header using full width

As you can see the columns are a little bit wider than in the previous example and the table fills the printable area of the paper.

2.17.4 Column widths in percent

See example – https://www.adiuvaris.ch/example-14

It is possible to define the widths of columns in percent of the parent frame. To achieve that you have to pass the values for the widths as strings.

```
// Add the table
$table = new ReportLib\TableFrame();
$body->addFrame($table);

// Add four columns to the table
$table->addColumn("frametype", "Frame type", "15.0%");
$table->addColumn("container", "Container type", "15.0%", 'C');
$table->addColumn("description", "Description", "40.0%");
$table->addColumn("number", "Number", "10.0%", 'R');
```

In the case of widths in percent the library always uses the full width of the parent frame as 100% to calculate the width of the columns. The code above will produce an output like this.

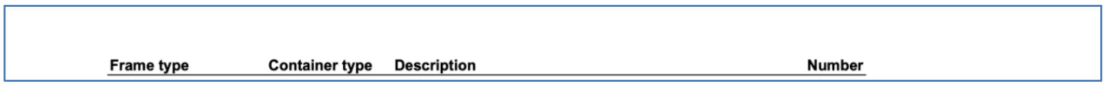


Figure 9 Table header with relative widths

It uses 80% of the width of the parent frame. The advantage of defining width in percent is that you have not to adjust the width in the case of changing paper sizes or margins.

You can use the attribute *useFullWidth* from the *ReportFrame* here as well. If this value is set to true the library enlarges the table to fit into the width of the parent frame (e.g., the printable width of the page). In this case the table calculates the width of the columns relative to their original size.

2.17.5 Line breaks in table rows

See example – https://www.adiuvaris.ch/example-15

The sum of the widths of all columns of a table can be larger than the maximum width of the frame. In this case the *ReportLib* inserts line breaks before the column that would make the table wider than the frame. The library prints the data of a row on multiple lines.

```
// Add the table
$table = new ReportLib\TableFrame();
$body->addFrame($table);

// Add columns to the table where the sum
// of the widths is greater than the width of the surrounding frame
$table->addColumn("frametype", "Frame type", 50.0);
$table->addColumn("container", "Container type", 30.0);
$table->addColumn("description", "Description", 80.0);
$table->addColumn("dummy", "", 50.0);
$table->addColumn("number", "Number", 30.0);
```

In this example I added a dummy column to align the columns on the different lines in the output. The header would look like the following figure. I left the title for the *dummy* column empty. Because the dummy column has the same width as the *frame type* column the library prints the *number* column directly below and aligned to the *container* column.

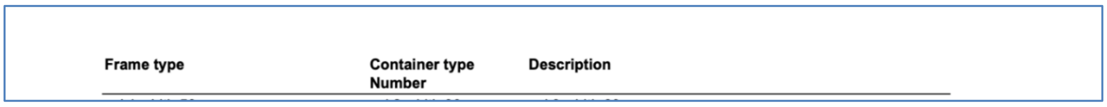


Figure 10 Table header with line breaks

You could add line breaks manually by setting the corresponding attribute in a *TableColumn* if you want to split the data of a table row onto multiple lines.

2.17.6 *Class TableRow*

This class defines a row in a table. It contains the data for the columns and a few attributes to control the output. The data is saved in an array where the keys are the names of the columns, and the values are strings. It is not necessary to add empty values.

The row type defines the text style that the library will use to print the data of the row.

If you want to add a row to a table, you have to create a *TableRow* object. The parameter defines the row type. The row type only defines the text style for the row.

```
// Create a table row of type detail
$row = new ReportLib\TableRow('D');
```

You can use the following row types:

- 'H' for a row that uses the *headerTextStyle*
- 'D' for a row that uses the *detailRowTextStyle*
- 'S' for a row that uses the *subTotalRowTextStyle*
- 'T' for a row that uses the *totalRowTextStyle*

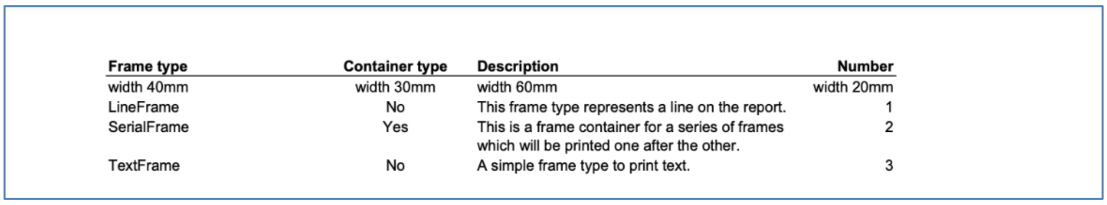
To add data for the columns, you have to call the *setText* function with the name of the column as first and the value as the second parameter. The column name must correspond with the columns added the table. It is a clever idea to create constants for them. Here we use simple strings with the names that I used to create the *TableFrame*.

```
$row->setText("frametype", "LineFrame");
$row->setText("container", "No");
$row->setText("description", "This frame type represents a line on the report.");
$row->setText("number", "1");
```

At the end you have to add the row to the table. The library prints them in the same order as you add them to the table.

```
// Add the row to the table
$table->addDataRow($row);
```

The output of a table with data looks like this. This is the look of a table when you do not change any attributes of the *TableFrame* or the *TableColumn*. It prints a header row and a line below that followed by the data rows.



Frame type	Container type	Description	Number
width 40mm	width 30mm	width 60mm	width 20mm
LineFrame	No	This frame type represents a line on the report.	1
SerialFrame	Yes	This is a frame container for a series of frames which will be printed one after the other.	2
TextFrame	No	A simple frame type to print text.	3

Figure 11 Simple table with data

In the section about the *TableFrame* class you saw that you can define a lot of things to format a table. You can set the text styles or the lines and the border.

2.17.7 *Joining columns*

You can join multiple columns in a *TableRow*. The value of the first column in the joint is printed the others are ignored. You can use that to print subtitles in a row which need more space than one column offers.

In the sample table with four columns, you could add a row as follows to print the text of the first column over the whole row. Normally it would wrap to a second line because the first column is only 40mm wide.

```
$row = new ReportLib\TableRow('T');
$row->setText("frametype", "This is a join from the first to the last column.");
$row->setJoinStart(0);
$row->setJoinEnd(3);
$table->addDataRow($row);
```

2.18 Convenience functions in ContainerFrame

See example – https://www.adiuvaris.ch/example-29

The class *ContainerFrame* offers a list of functions which makes the work with the library a little bit easier. You can call these functions from any object that is derived from *FrameContainer*, and they return a reference to the newly created object.

- *AddVContainer*: Adds a vertical organized container
- *AddHContainer*: Adds a horizontal organized container
- *AddHDistance*: Add a horizontal distance
- *AddVDistance*: Adds a vertical distance
- *AddHBox*: Like AddHContainer but with a box around the container
- *AddVBox*: Like AddVContainer but with a box around the container
- *AddHBlock*: Like AddHContainer but with a background color
- *AddVBlock*: Like AddVContainer but with a background color
- *AddLine*: Adds a line
- *AddHLine*: Horizontal line
- *AddVLine*: Vertical line
- *AddText*: Adds text
- *AddTexInBox*: Adds some text with a fix width
- *AddImage*: Adds an image
- *AddPageBreak*: Adds a manual page break
- *AddTable*: Adds a table
- *AddTextBlock*: Adds text with fix width and with a background color

2.19 Exceptions

See example – https://www.adiuvaris.ch/example-28

If the library is not able to calculate and position all the frames it will throw an exception. This happens also if you create a circular dependency in your report tree structure. Another problem is if you add an ImageFrame with invalid image file (no access or not an image). Therefore, you should catch exceptions when you call the output function like in the following example.

```
// Endless loop because of circle reference in the structure
$box1 = $body->AddBox();
$box2 = $box1->AddBox();
$box2->addFrame($body);
try {
    $report->output(__DIR__ . "/example_028.pdf");
} catch (Exception $e) {
    echo ($e);
}
```

2.20 Internal classes

ReportLib uses the following classes internally to calculate sizes and positions of frames.

2.20.1 Class Rect

The library uses this class to calculate rectangles regarding margins, paddings, or alignments. One function uses the current rectangle as a basis to calculate a rectangle that have added the margins on all four sides. Another function does the opposite thing and subtracts the paddings from current rectangle to calculate an inner rectangle.

2.20.2 Class Size

This class represents a rectangular space with a width and a height independent of a position. The library uses it to hold calculated sizes of frames.

2.20.3 SizeState

This class holds the current state of a frame during the calculation of sizes and positions. It contains a required size and if the remaining space on a page is big enough for the frame.