TMS FNC PDF Library
DEVELOPERS GUIDE

January 2022

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Getting Started

Included in the TMS FNC Core is a PDF library that is capable of generating PDF files. The PDF library supports creating documents, adding pages and page content such as HTML formatted text, plain text, drawing primitives such as rectangles, circles and lines, Images and many more …

To get started with the PDF library, add the FMX.TMSFNCPDFLib, VCL.TMSFNCPDFLib, LCLTMSFNCPDFLib or WEBLib.TMSFNCPDFLib depending on the chosen framework. The PDF library class is called TTMSFNCPDFLib, and the code is shareable between the four supported frameworks.

Starting a new document

Starting a new document can be file-based, or TMemoryStream-based. To start a new document, call the BeginDocument function. The BeginDocument always needs to be paired with EndDocument, which is responsible for writing the contents to a file or TMemoryStream. When the AFileName parameter in the BeginDocument call is empty, the contents will be written to a TMemoryStream, returned by the EndDocument call. The EndDocument call also has an additional parameter to allow opening the generated PDF file in the default PDF reader application. This only occurs when the file exists and the BeginDocument has received a valid AFileName parameter.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.EndDocument;
  finally
    p.Free;
  end;
end;
```

Adding pages

Adding pages can be done by calling the NewPage function. The NewPage is responsible for starting a new page content stream on which the graphics / text can be written. Each NewPage call will clear the content buffer and allow you to start with new text and graphics. Please note though that all appearance settings such as fill, stroke and font will be preserved, so starting a new page will allow you to continue in the same appearance settings as the previous page. When the NewPage call is missing, the EndDocument call automatically throws an error indicating a new page needs to be added.

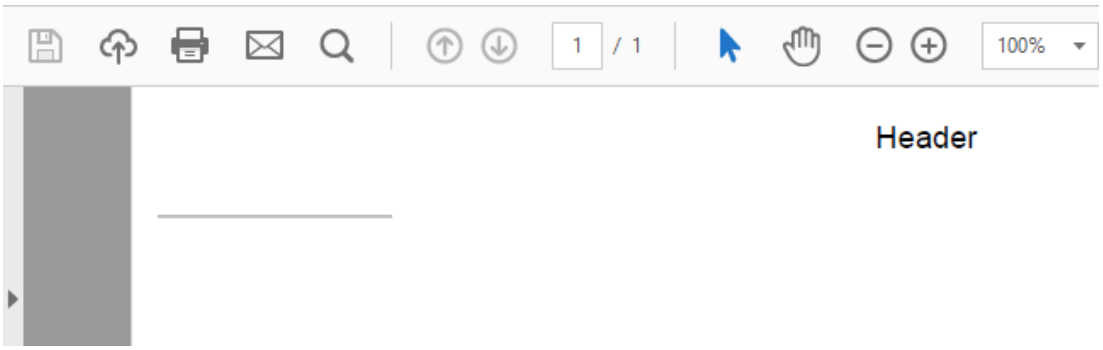
```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.EndDocument;
```

```
  finally
    p.Free;
  end;
end;
```

Drawing content on a page

After adding a new page, the page can be filled with content such as HTML formatted text, plain text as well as some basic drawing primitives such as rectangles, circles and lines. The PDF graphics engine is separate interface implementation that allows you to pass a reference to this interface. This has the advantage that it blocks access to the main pdf library, to avoid document / page corruption.

To starting drawing content, the pdf library has a Graphics property that provides access to the drawing functionality. As a simple example, the following code exports a single-page PDF with a horizontal gray line.



Please note that the Header and Footer text are added by default. The look and feel and text can be configured separately with a series of Header and Footer properties.

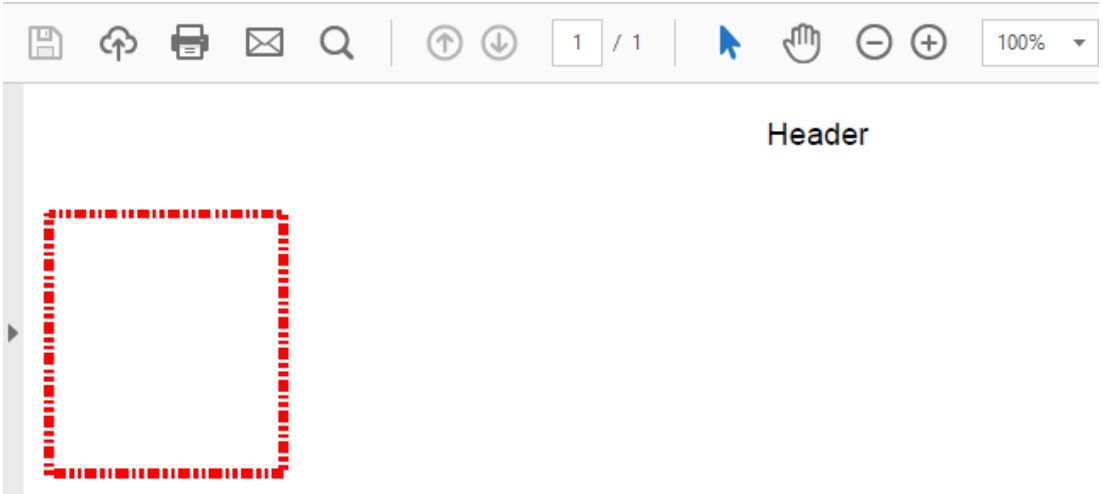
The appearance of the line in the above PDF export is controlled via the Stroke property. The stroke property supports setting the Color, Width and Kind. The Kind property is used to change between a solid, dashed or dotted line, or to completely turn of line drawing. The Stroke property is applied to each primitive that is drawn. The code below changes the appearance of the line to be thicker, have a different kind and color. (Note that most type values and constants can be found in the TTMSFNCGraphics class unit).

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Stroke.Color := gcRed;
    p.Graphics.Stroke.Width := 3;
    p.Graphics.Stroke.Kind := gskDashDotDot;
    p.Graphics.DrawLine(PointF(10, 50), PointF(100, 50));
    p.EndDocument(True);
  finally
    p.Free;
```

```
  end;
end;
```

When keeping the above configuration for stroke and adding code to draw a rectangle instead, the stroke properties are also applied to the rectangle. The difference between a line is that a rectangle is a path shape, and a path can also be filled. The fill is controlled with the Fill property and supports horizontal and vertical gradients.

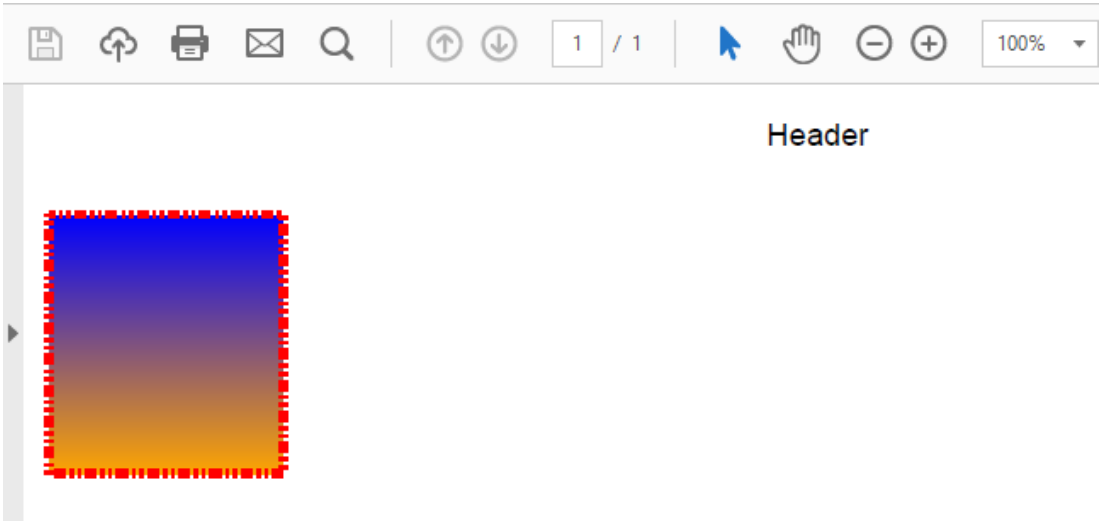
```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Stroke.Color := gcRed;
    p.Graphics.Stroke.Width := 3;
    p.Graphics.Stroke.Kind := gskDashDotDot;
    p.Graphics.DrawRectangle(RectF(10, 50, 100, 150));
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



Applying a gradient is done through the Fill property as demonstrated in the following sample.

```
procedure TForm1.GeneratePDF(AFileName: string);
```

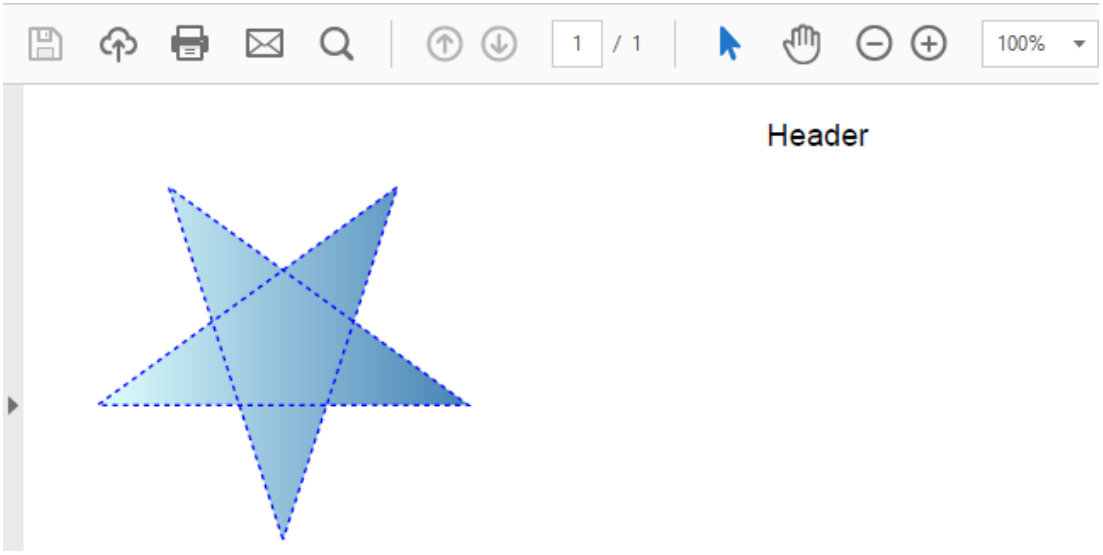
```
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Stroke.Color := gcRed;
    p.Graphics.Stroke.Width := 3;
    p.Graphics.Stroke.Kind := gskDashDotDot;
    p.Graphics.Fill.Kind := gfkGradient;
    p.Graphics.Fill.Color := gcBlue;
    p.Graphics.Fill.ColorTo := gcOrange;
    p.Graphics.DrawRectangle(RectF(10, 50, 100, 150));
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



As already mentioned, the rectangle, circle and line drawing functionality is based on paths. The pdf library also exposes a set of drawing calls to construct your own path. Below is a sample of drawing a star shape filled with a gradient fill and a dotted stroke.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  st: TPointF;
  rad, angle, x, y: Single;
  I: Integer;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Stroke.Color := gcBlue;
    p.Graphics.Stroke.Width := 1;
    p.Graphics.Stroke.Kind := gskDot;
```

```
    p.Graphics.Fill.Kind := gfkGradient;
    p.Graphics.Fill.Orientation := gfoHorizontal;
    p.Graphics.Fill.Color := gcLightcyan;
    p.Graphics.Fill.ColorTo := gcSteelblue;

    st := PointF(100, 100);
    rad := 75;
    angle := 0;
    x := rad * Sin(angle * Pi / 5) + st.X;
    y := rad * Cos(angle * PI / 5) + st.Y;

    p.Graphics.DrawPathBegin;
    p.Graphics.DrawPathMoveToPoint(PointF(x, y));

    for I := 1 to 5 do
    begin
      x := rad * sin((i * 4 * pi + angle) / 5) + st.X;
      y := rad * cos((i * 4 * pi + angle) / 5) + st.Y;
      p.Graphics.DrawPathAddLineToPoint(PointF(x, y));
    end;

    p.Graphics.DrawPathClose;
    p.Graphics.DrawPathEnd;
    p.Graphics.DrawRestoreState;

    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



Drawing Text

Drawing text on a PDF page is done via the DrawText function. The DrawText has a set of parameters to allow drawing wordwrapped text in a rectangle, or simply as-is at a specific position.

The DrawText function has a set of overloads that also supports column drawing. Each call to DrawText returns a value that can either contain the calculated text rectangle or the amount of left-over characters after an overflow is detected when drawing text in a column.

The font that is used when drawing text can be controlled separately via the Font property. With this property, the font name, size, color and style can be set.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Segoe UI';
    p.Graphics.Font.Size := 16;
    p.Graphics.Font.Color := gcRed;
    p.Graphics.Font.Style := [TFontStyle.fsBold];
    p.Graphics.DrawText('Hello World !', PointF(10, 50));
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```

The above sample draws a 16 point single-line text in a red color with a bold style. The supported fonts in a PDF is based on the operating system font set on which the PDF is generated. If a specific font or a font name and style combination is not found, the font will fall back on one of the entries specified in the FontFallBackList property. The FontFallBackList is already initialized with a limited set of default operating system fonts, but you can easily add your own fonts, if they are installed and available on the operating system on which the PDF is generated.

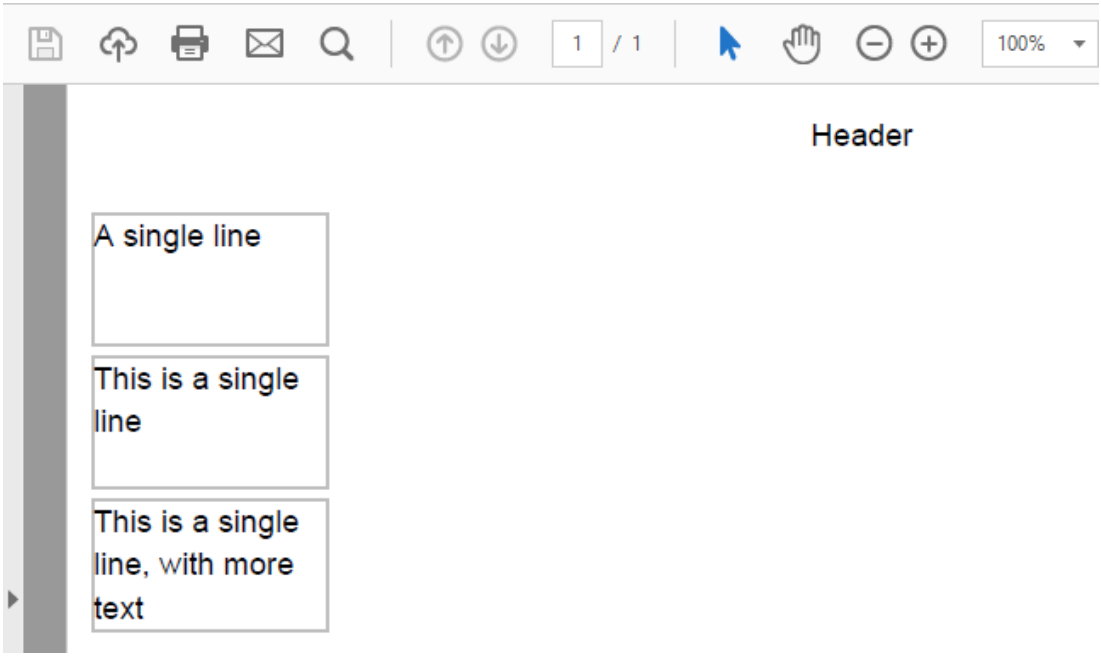
The pdf engine recognizes the Delphi carriage return(#13) and line feed(#10) combination in order to break text in lines. To break the above “Hello World !” in 2 lines add the #13#10 combination.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Segoe UI';
```

```
    p.Graphics.Font.Size := 16;
    p.Graphics.Font.Color := gcRed;
    p.Graphics.Font.Style := [TFontStyle.fsBold];
    p.Graphics.DrawText('Hello'#13#10'World !', RectF(10, 50, 150, 100));
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```

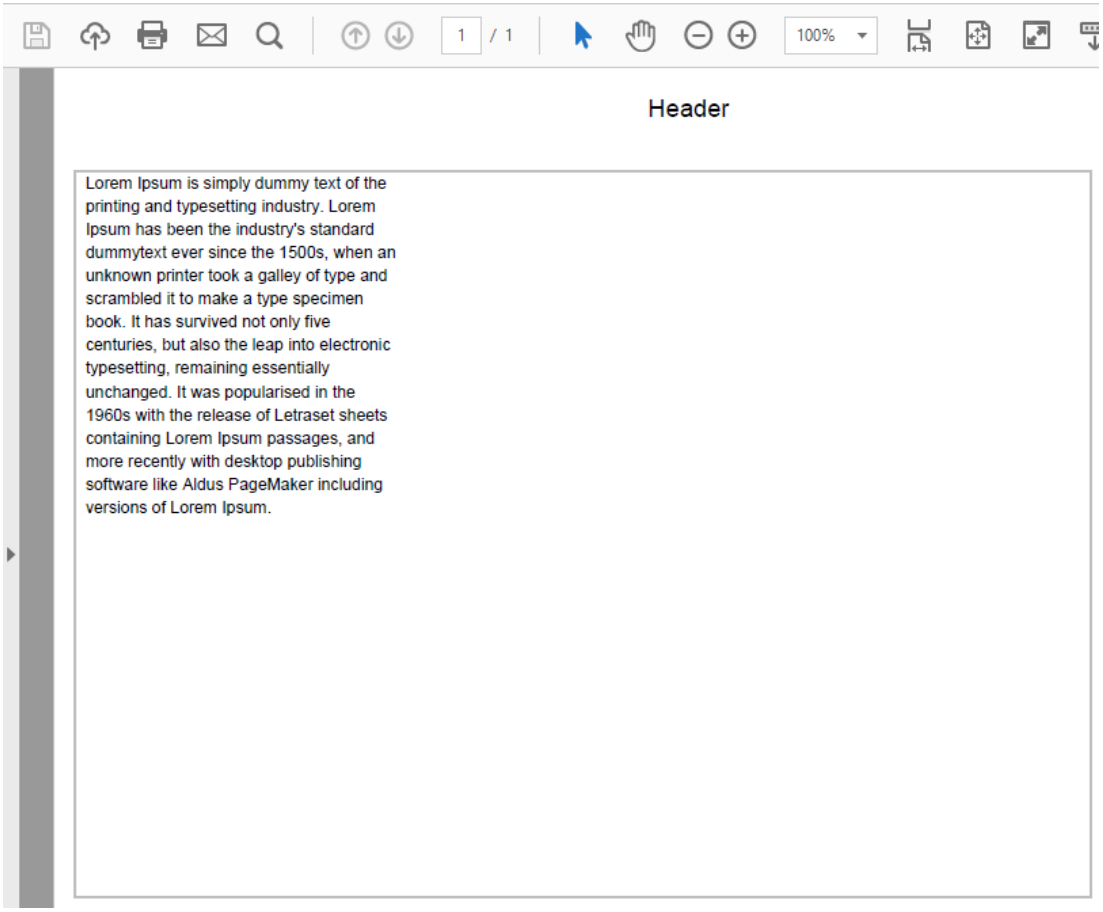
If you are not using #13#10 in your text, but want to automatically wordwrap text based on the width of a rectangle then you can change the drawing functionality to add a TRectF parameter instead of a TPointF. The following sample draws 3 different lines of text in the same rectangle. As the text length changes, the text is automatically wordwrapped before the width of the rectangle is reached.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  r: TRectF;
begin
  p := TTMSFNCPDFLib.Create;
  try
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Arial';
    p.Graphics.Fill.Color := gcNull;
    r := RectF(10, 50, 100, 100);
    p.Graphics.DrawRectangle(r);
    p.Graphics.DrawText('A single line', r);
    r := RectF(10, 105, 100, 155);
    p.Graphics.DrawRectangle(r);
    p.Graphics.DrawText('This is a single line', r);
    r := RectF(10, 160, 100, 210);
    p.Graphics.DrawRectangle(r);
    p.Graphics.DrawText('This is a single line, with more text', r);
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



As already mentioned, the PDF library also supports drawing text in one or multiple columns. If the end of a column is reached, and the drawing call specifies multiple columns, the text is automatically drawn in the next available column. Below is a sample that demonstrates this based on a long text drawn in 3 columns. The first drawing sets a font of 8 points which draws the text completely in the first column. As the font size increases, more columns are used.

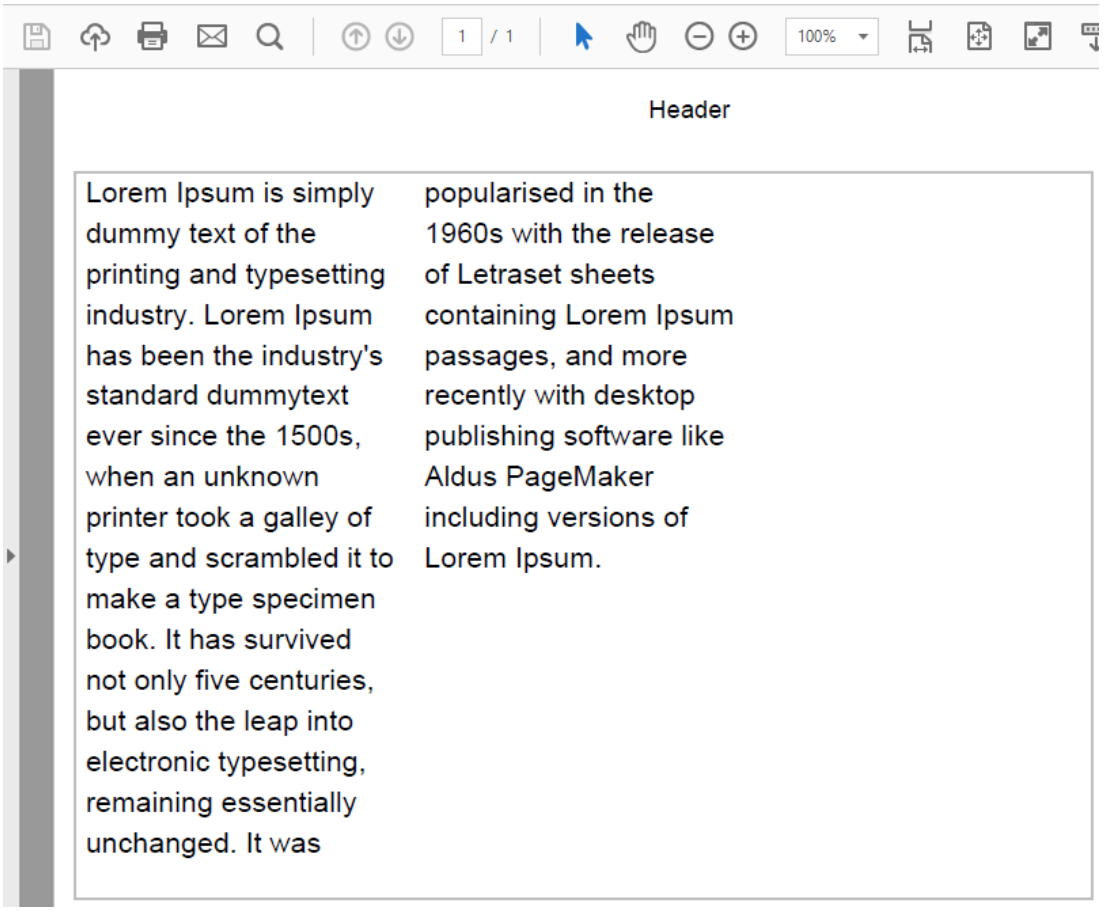
```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  s: string;
begin
  p := TTMSFNCPDFLib.Create;
  try
    s := 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has
been the industry''s standard dummy'+
         'text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to
make a type specimen book. '+
         'It has survived not only five centuries, but also the leap into electronic typesetting,
remaining essentially unchanged. It'+
         ' was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum
passages, and more recently with des'+
         'ktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.';
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Arial';
    p.Graphics.Font.Size := 8;
    p.Graphics.Fill.Color := gcNull;
    p.Graphics.DrawText(s, RectF(10, 50, 500, 400), 3);
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



When modifying the same sample and choosing a different font size, the text is flowed into the next column.

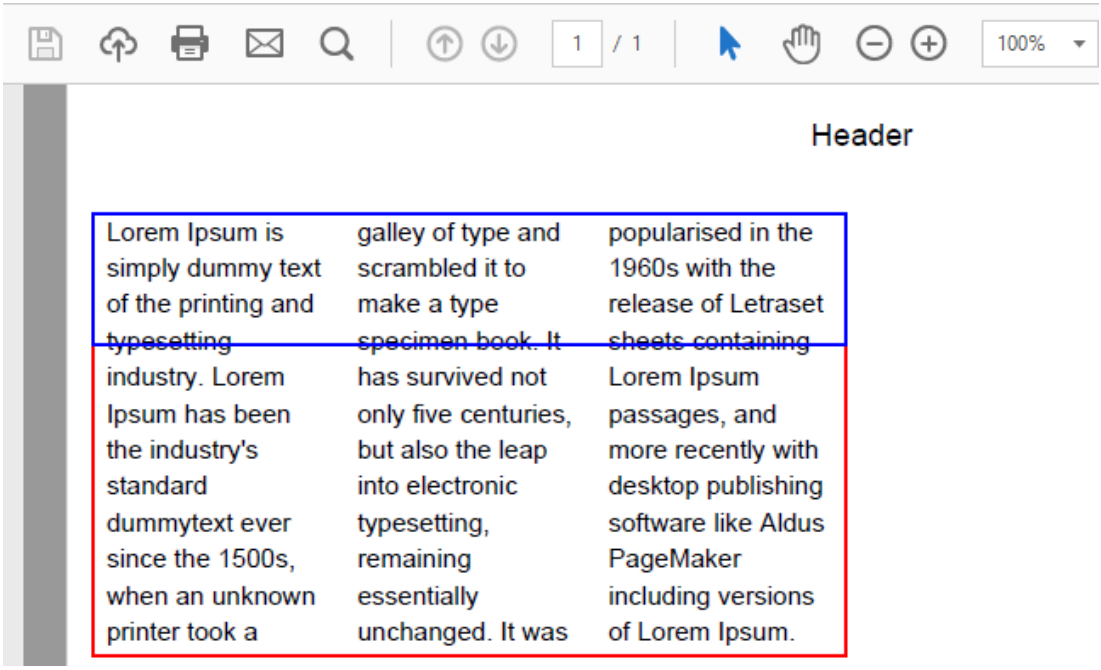
```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  s: string;
  r: TRectF;
begin
  p := TTMSFNCPDFLib.Create;
  try
    s := 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy'+
         'text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. '+
         'It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It'+
         ' was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with des'+
         'ktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.';
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Arial';
    p.Graphics.Font.Size := 14;
    p.Graphics.Fill.Color := gcNull;
```

```
    r := RectF(10, 50, 500, 400);
    p.Graphics.DrawRectangle(r);
    p.Graphics.DrawText(s, r, 3);
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



When the rectangle is not large enough to contain all text, or the amount of columns specified does not suffice for displaying all of the text, the DrawText function returns a value of the amount of characters that are not drawn. Each DrawText call has an additional parameter to calculate the text. In this case, the method also returns a TRectF or Integer parameter but does not actually draw the text. In particular for the DrawText functionality that has the ability to draw text in multiple columns, you can increase the width/height or amount of columns in a while loop, until the character overflow count is 0. This is demonstrated in the next sample. Note that this sample shows the original requested rectangle, and the rectangle necessary for displaying all text after calculating the amount of overflow characters.

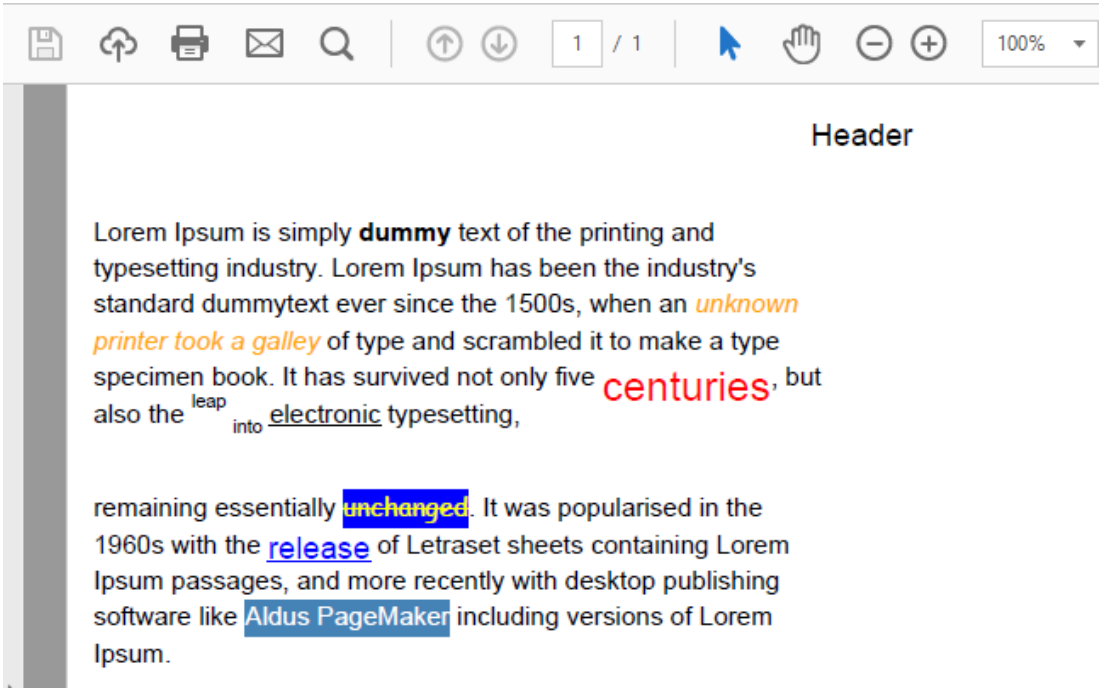
The sample indicates the original requested rectangle to draw the text in blue and the actual rectangle calculated with detection of character overflow in red.



The PDF library also supports HTML formatted text drawing. HTML formatted text is only available for drawing at a specific position or wordwrapped in a specific rectangle. Character overflow detection is not supported for HTML formatted text drawing. Below is a sample that draws the above text with HTML formatting applied.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  s: string;
  r: TRectF;
begin
  p := TTMSFNCPDFLib.Create;
  try
    s := 'Lorem Ipsum is simply <b>dummy</b> text of the printing and typesetting industry. Lorem
Ipsum has been the industry''s standard dummy'+
         'text ever since the 1500s, when an <i><font color="gcOrange">unknown printer took a
galley</font></i> of type and scrambled it to make a type specimen book. '+
         'It has survived not only five <font size="16" color="gcRed">centuries</font>, but also the
<sup>leap</sup> <sub>into</sub> <u>electronic</u> typeset'+
         'ting, <br><br>remaining essentially <font color"gcYellow" face="Comic Sans MS"
bgcolor="gcBlue"><s>unchanged</s></font>. It'+
         ' was popularised in the 1960s with the <font size="12"><a
href="http://www.tmssoftware.com">release</a></font> of Letraset sheets containing Lorem Ipsum
passages, and more recently with des'+
         'ktop publishing software like <font bgcolor="gcSteelBlue" color="gcWhite">Aldus
PageMaker</font> including versions of Lorem Ipsum.';
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Arial';
    p.Graphics.Font.Size := 10;
    p.Graphics.Fill.Color := gcNull;
    r := RectF(10, 50, 300, 400);
    p.Graphics.DrawHTMLText(s, r);
```

```
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



The above code renders the HTML formatted text in a rectangle with wordwrapping, but can easily be rendered as-is starting from a specific position, and then only breaks the text on the
 tag. The HTML formatted text is based on the Mini HTML rendering engine.

Each DrawText or DrawHTMLText call returns a TRectF containing the text width/height or character count overflow. These calculations can also be retrieved with the CalculateText and CalculateHTMLText calls.

Font Embedding

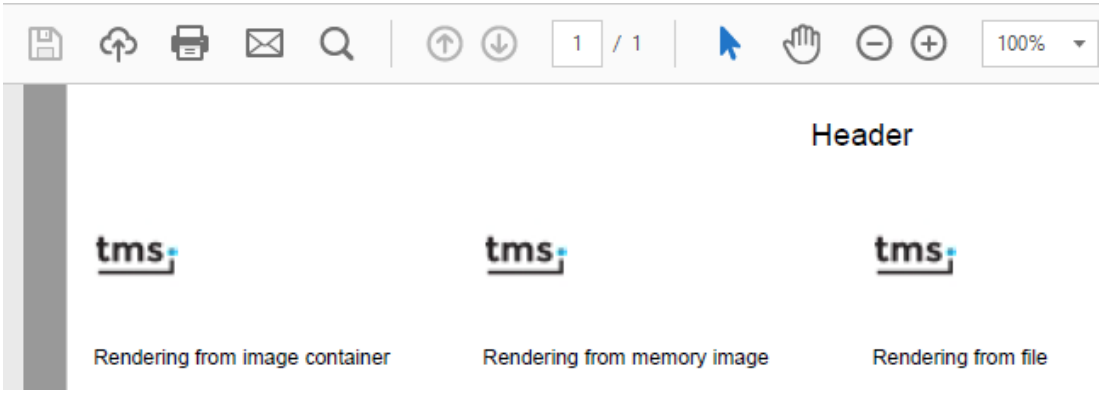
By default, each font that is used in the PDF library is embedded to make sure the PDF is readable on other operating systems. This means that for the rendered characters, a font subset is created and included to make sure the size of the PDF file is a small as possible. If the PDF file is created for printing purposes on the operating system on which it is generated, you could consider turning off font embedding, to have even smaller PDF files and speed up the creation process. Font embedding can be controlled with the EmbedFonts property.

Images

The PDF library supports drawing images at a specific position or with optional stretching and aspect ration in a rectangle. To draw images, you can use the DrawImage, DrawImageFromFile or DrawImageWithName functions. The first method takes an existing image object, the second method takes an existing image file and the last method requires assigning a BitmapContainer and

then uses the ABitmapName parameter to find & draw the image on the PDF page canvas. Below is a sample that demonstrates these three ways of adding images.

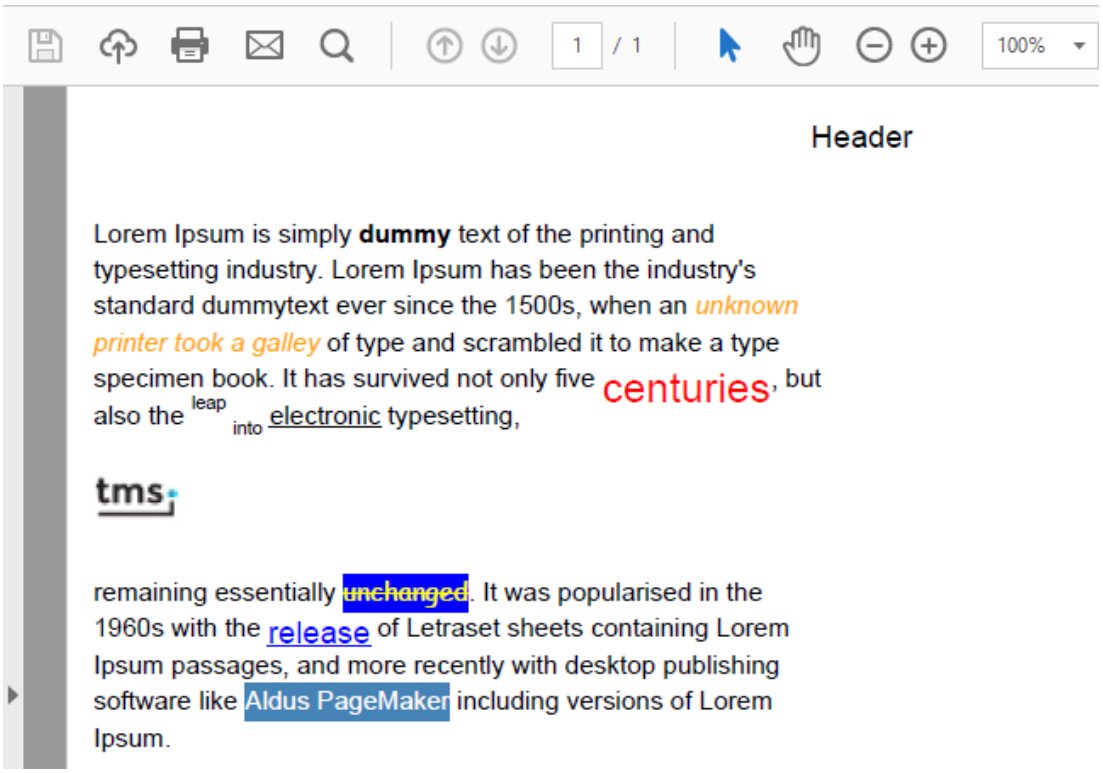
```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  MyImage: TBitmap;
begin
  p := TTMSFNCPDFLib.Create;
  MyImage := TBitmap.Create;
  try
    p.BitmapContainer := TMSFNCBitmapContainer1;
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Arial';
    p.Graphics.Font.Size := 8;
    p.Graphics.Fill.Color := gcNull;
    p.Graphics.DrawImageWithName('MyImage', PointF(10, 50));
    p.Graphics.DrawImageFromFile('MyImage.jpg', PointF(160, 50));
    MyImage.LoadFromFile('MyImage.jpg');
    p.Graphics.DrawImage(MyImage, PointF(310, 50));
    p.Graphics.DrawText('Rendering from image container', PointF(10, 100));
    p.Graphics.DrawText('Rendering from memory image', PointF(160, 100));
    p.Graphics.DrawText('Rendering from file', PointF(310, 100));
    p.EndDocument(True);
  finally
    MyImage.Free;
    p.Free;
  end;
end;
```



Assigning an image container can also be used to render images that are added through the IMG tag. This is demonstrated below, based on the HTML formatted text sample.

```
procedure TForm1.GeneratePDF(AFileName: string);
var
  p: TTMSFNCPDFLib;
  s: string;
  r: TRectF;
begin
  p := TTMSFNCPDFLib.Create;
  try
```

```
    s := 'Lorem Ipsum is simply <b>dummy</b> text of the printing and typesetting industry. Lorem
Ipsum has been the industry"s standard dummy'+
         'text ever since the 1500s, when an <i><font color="gcOrange">unknown printer took a
galley</font></i> of type and scrambled it to make a type specimen book. '+
         'It has survived not only five <font size="16" color="gcRed">centuries</font>, but also the
<sup>leap</sup> <sub>into</sub> <u>electronic</u> typeset'+
         'ting, <br><img src="TMSLogo.ico"/>remaining essentially <font color"gcYellow" face="Comic
Sans MS" bgcolor="gcBlue"><s>unchanged</s></font>. It'+
         ' was popularised in the 1960s with the <font size="12"><a
href="http://www.tmssoftware.com">release</a></font> of Letraset sheets containing Lorem Ipsum
passages, and more recently with des'+
         'ktop publishing software like <font bgcolor="gcSteelBlue" color="gcWhite">Aldus
PageMaker</font> including versions of Lorem Ipsum.';
    p.BitmapContainer := TMSFNCBitmapContainer1;
    p.BeginDocument(AFileName);
    p.NewPage;
    p.Graphics.Font.Name := 'Arial';
    p.Graphics.Font.Size := 10;
    p.Graphics.Fill.Color := gcNull;
    r := RectF(10, 50, 300, 400);
    p.Graphics.DrawHTMLText(s, r);
    p.EndDocument(True);
  finally
    p.Free;
  end;
end;
```



Graphics engine

The PDF library is capable of rendering primitives such as rectangles, ellipses or more complex paths built up of points. This can be done by combining the correct DrawPath* calls. To simplify this, the TTMSFNCGraphicsPDFEngine is available, that has the same methods as if you would draw on a canvas within WEB, FMX, VCL or LCL. So as an alternative to drawing, constructing paths via direct PDF draw calls, you can create an instance of TTMSFNCGraphicsPDFEngine which eliminates these complex path draw calls. Below is a sample that renders a diamond shape with rotated text.

```
uses
  FMX.TMSFNCPDFLib, FMX.TMSFNCGraphicsTypes, FMX.TMSFNCGraphicsPDFEngine;

procedure TForm1.GeneratePDF;
var
  p: TTMSFNCPDFLib;
  g: TTMSFNCGraphicsPDFEngine;
  pth: TTMSFNCGraphicsPath;
  r: TRectF;
begin
  p := TTMSFNCPDFLib.Create;
  g := TTMSFNCGraphicsPDFEngine.Create(p);
  pth := TTMSFNCGraphicsPath.Create;
  try
    p.BeginDocument('rotation.pdf');
    p.NewPage;
    pth.MoveTo(PointF(200, 200));
    pth.AddLine(PointF(200, 200), PointF(300, 300));
    pth.AddLine(PointF(300, 300), PointF(200, 400));
    pth.AddLine(PointF(200, 400), PointF(100, 300));
    pth.ClosePath;
    g.DrawPath(pth);

    r := RectF(100, 200, 300, 400);
    g.DrawText(r, 'Hello World!', False, gtaCenter, gtaCenter, gttNone, -45);

    p.EndDocument(True);
  finally
    pth.Free;
    g.Free;
    p.Free;
  end;
end;
```

Document & Page Settings

A new document can be configured to add meta-data. Below is an overview which meta-data is supported by the PDF library.

Author, Creator, Title, Subject and Keywords. The first four meta-data entries are strings and the Keywords entry is a list of strings. These values can be added directly on PDF document level.

Each new page starts with a header and footer and a specific page size. The header & footer can be customized with the Header* and Footer* properties. These include size, margins, alignment and font. The header and footer also supports HTML formatted text. By default, the page size is set to portrait A4, but can easily be changed by setting the PageSize and PageOrientation properties. Optionally a custom page size can be configured by setting the PageSize property to psCustom. The PageWidth and PageHeight properties can be used to set the custom page size. These properties can also be used to retrieve the current applied PageWidth and PageHeight independent of the PageSize property setting.

Components that support PDF Export

Below is a list of components that support exporting to a PDF, all components inherit from a base TTMSFNCPDFIO component that supports configuration of meta-data and header and footer as well as custom drawing / customization through events.

- TTMSFNCGrid
- TTMSFNCChart
- TTMSFNCRichEditor

To get started, drop one of the TTMSFNC*PDFIO components on the form. Call the Save function to save to a file or stream.

Additionally, the TTMSFNCGraphicsPDFIO component is capable of rendering a PDF version of the components as if painted within a Canvas.

WEB Support

Font caching

In a TMS WEB Core application, it is required to pre-load fonts to make PDF export function properly. This can be done by using the AddFontToCache method and specify a TTF file representing the fonts that you are using inside the component, or when generating a custom PDF. The font that is used by default is Arial, and the font used inside components is Tahoma. Below is a sample code that demonstrates how to add fonts.

```
AddFontToCache('http://www.myserver.com/fonts/arial.ttf');
AddFontToCache('http://www.myserver.com/fonts/tahoma.ttf');
```

As the fonts are loaded asynchronously, there is an event that is triggered when all fonts added to the font cache are preloaded. The OnFontCacheReady event can be used to make sure that the fonts are ready to use when exporting to PDF.

```
Application.OnFontCacheReady := DoFontCacheReady;
```

Compression

When a pdf is generated, the content and fonts that are embedded are compressed. In FMX, VCL and LCL this is automatically done, but on WEB a third party library is needed, based on the zlib compression algorithm. Whenever a new WEB project is created you should, by default, add the pako.js library. To enabled this, click on the “Manage JavaScript Libraries” project right-click context menu item and choose the “Pako ZLib Compression (PDF)” menu item.