Ciansoft PDFBuilderX User Manual (Version 2.3)

Introduction

Ciansoft PDFBuilderX is an ActiveX control that enables applications to create PDF files. This document contains comprehensive instructions on installing and using PDFBuilderX.

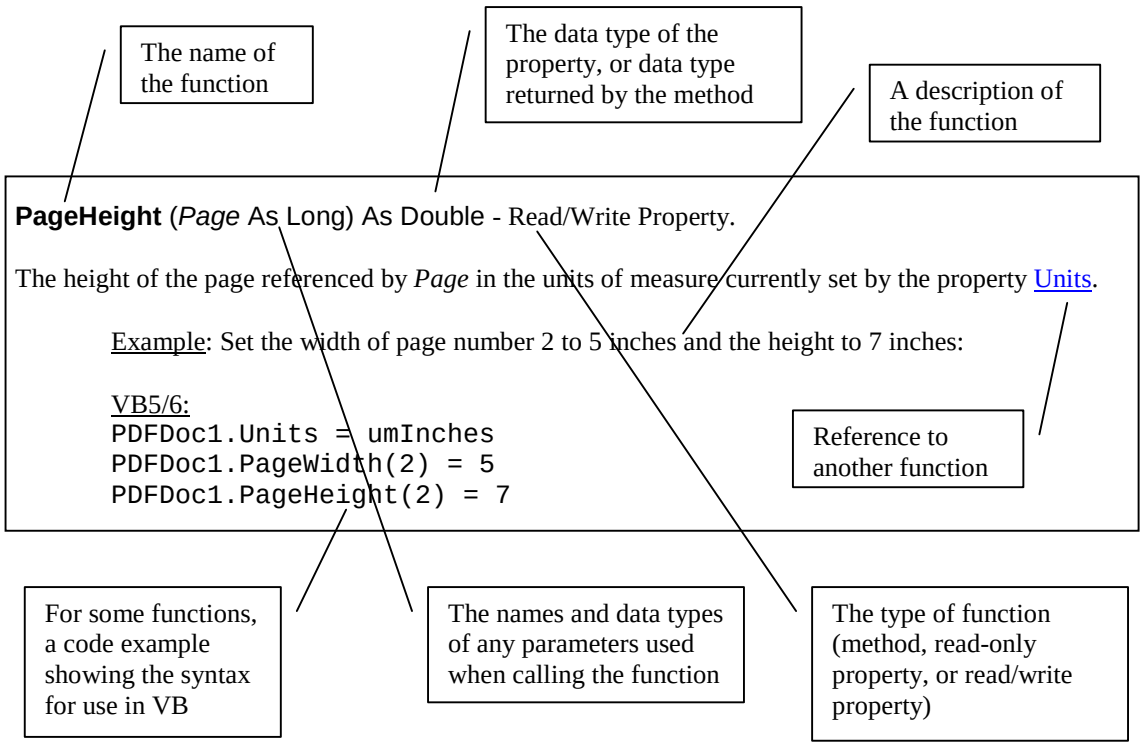
For further information, visit our website: www.ciansoft.com.

Alternatively, contact us by email, we will be pleased to answer your questions: info@ciansoft.com.

How to Use This Manual

The first part of this manual explains how to start using PDFBuilderX. This describes installation of the control, the basic approach to creating a PDF document, and information about the trial version. We especially recommend reading 1.2 Steps to Create PDF Documents first in order to understand the general principles of using this control.

After that, the remainder of the manual describes all the available functions. For each function, details as shown below are available:



A complete Alphabetical List of Functions can be found at the end of this document.

PDFBuilderX is most commonly used in Microsoft Visual Basic applications, although many other development environments supporting the use of ActiveX controls can equally well be used. Throughout the document example code is given to show the syntax for VB5/6 and VB.NET. Where only one example is given, the syntax for the two languages is the same.

TABLE OF CONTENTS

1. Getting Started

1.1. Installation

The OCX file, PDFBuilderX.ocx, must be registered on the computer running the application. This should be done during the installation, but if it is not, the command line utility Regsvr32.exe can be used. This is usually found in the Windows system folder and runs using the syntax:

regsvr32 *ocxname*

where *ocxname* is the path and name of the ocx file to register. When deploying an application that uses PDFBuilderX, the ocx file will also need to be registered on the target machine. Note that the licence file, PDFBuilderX.lic, is needed to use this control in a design environment and this file must not be distributed with an application, as is confirmed in your licence agreement.

To use this control in a VB5 or VB6 project, select Project and Components from the pull down menu. This gives a list of available controls. Select PDFBuilderX Library and click Apply. This adds the control to the component palette. The class name is "PDFDoc" and this will appear in the Object Browser where the properties and methods are listed.

In VB.NET, the control is added to the Toolbox using a menu item under the Tools menu. Depending on the version of VB.NET being used, this item is called "Customize Toolbox" or "Add/Remove Items" or "Choose Toolbox Items".

For other design environments, consult the documentation for importing ActiveX controls.

As an ActiveX control, PDFBuilderX can be used in a range of Windows based environments and languages. There will be slight differences in syntax especially with the use of parentheses/brackets surrounding method parameters.

1.2. Steps to Create PDF Documents

An instance of PDFBuilderX placed on a form in an application represents a single PDF document. In order to create and save a document, the following steps must be followed:

1. Configure document settings such as the default page size and units of measure.
2. Add pages to the document.
3. Add objects to the document that are to be written on the pages. This can include images, graphics or blocks of text. These objects are referred to as "Resources".
4. Draw resources onto the pages.
5. Save the file to disk.

Resources are added to the document in different ways. For images, the AddImageFile function can be used to add an image from a file on disk. Graphics and blocks of text are created using the CreateGraphic or CreateText functions, with further functions then being used to draw the graphic or add the text block content.

All resources are allocated a unique identifying number. This is the return value of the function used to add or create the resource.

Resources are drawn on pages using the ApplyResource function. This function also returns a reference number, which can then be used to identify the object on the page. This reference number is used when calling functions to change the position or size of the object as it is displayed on the page.

Note that this reference number is not the same as the number that identifies resources in the document as a whole and should not be confused with it. To help clarify this issue, parameters used in functions will usually be called *Resource* where they refer to a resource identifier, *Page* where they refer to a page number and *Index* where they refer to an object on a page.

A resource can be used as many times as necessary, either on the same page, or on multiple pages. This is useful, for example, for setting up headers and footers on pages of a document, or for displaying an image full size on one page and as a thumbnail elsewhere in the document.

For VB5/6 users, a simple example project is included with the installation. This can be accessed from the Windows Start Menu.

1.3. The Trial Version

The trial version of PDFBuilderX is supplied as a different OCX file, called PDFBuilderXTrial.ocx, and has a separate installation programme. The trial version has all the functionality of the full version of the control. The only limitation is that each page of PDF files created using the control will have a line of text written on it indicating that trial software was used. Visit www.ciansoft.com to purchase the full version.

2. Managing Pages in the Document

The following functions are used to add pages to the document, to change the page sizes and to adjust the position and size of objects on a page. Note that the functions Locate, ScaleObject and Rotate can be applied to images and text, but not to graphics.

AddPage (*Page* As Long) - Method.

Adds a new page to the document. The position of the page in the document is defined by *Page*. If *Page* is 0, the page will be added at the end of the document. The page size is initially defined by the DefaultPageSize property, but can be modified after the page has been added.

Example: Add a new page at the end of the document:

VB5/6:
```
PDFDoc1.AddPage 0
```

VB.NET:
```
AxPDFDoc1.AddPage(0)
```

PageSize (*Page* As Long) As TxPageSize - Read/Write Property.

The size of the page referenced by *Page*. This is an enumerated property of type *TxPageSize*, the possible values of which are listed below:

TxPageSize		
psCustom:	0	Page size is defined by the PageWidth and PageHeight properties
psA4_P: (Default)	1	A4 Portrait (210 mm x 297 mm)
psA4_L:	2	A4 Landscape (297 mm x 210 mm)
psLetter_P:	3	Letter Portrait (8.5” x 11”)
psLetter_L:	4	Letter Landscape (11” x 8.5”)
psA3_P:	5	A3 Portrait (297 mm x 420 mm)
psA3_L:	6	A3 Landscape (420 mm x 297 mm)
psA5_P:	7	A5 Portrait (148.5 mm x 210 mm)
psA5_L:	8	A5 Landscape (210 mm x 148.5 mm)
psTabloid_P:	9	Tabloid Portrait (11” x 17”)
psTabloid_L:	10	Tabloid Landscape (17” x 11”)
psLegal_P:	11	Legal Portrait (8.5” x 14”)
psLegal_L:	12	Legal Landscape (14” x 8.5”)
psStatement_P:	13	Statement Portrait (5.5” x 8.5”)
psStatement_L:	14	Statement Landscape (8.5” x 5.5”)
psExecutive_P:	15	Executive Portrait (7.25” x 10.5”)
psExecutive_L:	16	Executive Landscape (10.5” x 7.25”)

Example: Set the size of page number 2 of the document to be A3 Landscape:

VB5/6:
```
PDFDoc1.PageSize(2) = psA3_L
```

VB.NET:
```
AxPDFDoc1.set_PageSize(2, PDFBuilderXTrial.TxPageSize.psA3_L)
```

PageWidth (*Page* As Long) As Double - Read/Write Property.

The width of the page referenced by *Page* in the units of measure currently set by the property Units.

PageHeight (*Page* As Long) As Double - Read/Write Property.

The height of the page referenced by *Page* in the units of measure currently set by the property Units.

Example: Set the width of page number 2 to 5 inches and the height to 7 inches:

VB5/6:
```
PDFDoc1.Units = umInches
PDFDoc1.PageWidth(2) = 5
PDFDoc1.PageHeight(2) = 7
```

VB.NET:
```
AxPDFDoc1.Units = PDFBuilderXTrial.TxUnitsMeas.umInches
AxPDFDoc1.set_PageWidth(2, 5.0)
AxPDFDoc1.set_PageHeight(2, 7.0)
```

DefaultPageSize As TxPageSize - Read/Write Property.

The page size that will be used for new pages as they are added to the document. See PageSize for definition of possible values. (Default = psA4_P).

Example: Set the default page size to US Letter size (8.5" x 11"):

VB5/6:
```
PDFDoc1.DefaultPageSize = psLetter_P
```

VB.NET:
```
AxPDFDoc1.DefaultPageSize = _
  PDFBuilderXTrial.TxPageSize.psLetter_P
```

ApplyResource (*Page* As Long, *Resource* As Long) As Long - Method.

Applies the resource referenced by *Resource* to the page referenced by *Page*. The resource is applied using default sizing and positioning which can then be modified using the Locate or ScaleObject functions. The return value of this method indicates the index number of this specific instance of the resource on this specific page.

Example: Apply a text block referenced by the resource number Text1 (which is the return value of the CreateText method) to page 1 of the document:

```
TextIndex = PDFDoc1.ApplyResource(1, Text1)
```

Locate (*Page* As Long, *Index* As Long, *X* As Double, *Y* As Double) - Method.

The object referenced by *Index* on the page referenced by *Page* will be positioned on the page. If the object is an image or a left-justified text block, its top-left corner will be at the co-ordinates *X*, *Y* measured from the bottom-left corner of the page. If the object is a centred or right-justified text block, then the co-ordinates will refer to the top-centre or top-right of the object instead.

Example: Locate the text block that was applied in the above example, so that it is positioned in the top-left corner of the page. Note the use of the PageHeight property to determine how far the top of the page is from the bottom of the page that is used as the reference point:

VB5/6:
```
PDFDoc1.Locate 1, TextIndex, 0, PDFDoc1.PageHeight(1)
```

VB.NET:
```
AxPDFDoc1.Locate(1, TextIndex, 0, AxPDFDoc1.get_PageHeight(1))
```

ScaleObject (*Page* As Long, *Index* As Long, *ScaleFactor* As Double) - Method.

The object referenced by *Index* on the page referenced by *Page* will be scaled. *ScaleFactor* is a percentage value, so an object will be displayed at normal size if this value is 100. For example, to display an image as a thumbnail, one eighth of its normal size, use a *ScaleFactor* of 12.5, i.e., 100/8.

Example: Scale the text block that was applied in the above examples, so that it is displayed at twice its original size:

VB5/6:

```
PDFDoc1.ScaleObject 1, TextIndex, 200
```

VB.NET:

```
AxPDFDoc1.ScaleObject(1, TextIndex, 200.0)
```

Rotate (*Page* As Long, *Index* As Long, *Angle* As Double) - Method.

The object referenced by *Index* on the page referenced by *Page* will be rotated counter-clockwise by *Angle* degrees. The object pivots on its top-left corner.

Example: Rotate the text block that was applied in the above examples, so that it is inclined 10 degrees counter-clockwise from the horizontal:

VB5/6:

```
PDFDoc1.Rotate 1, TextIndex, 10
```

VB.NET:

```
AxPDFDoc1.Rotate(1, TextIndex, 10.0)
```

3. Working with Images

Images can be added to the document as resources either by reading image files in a supported format from disk, by copying a bitmap in memory as a bitmap handle or by reading an image stored as an array of bytes in memory.

AddImageFile (*FileName* As String) As Long - Method.

Adds an image from a file on disk as a resource in the document. The return value of the function is the resource identifying number. Images in the following file formats can be used: .bmp, .tif, .jpg, .png, .gif, .pcx, .psd, .wbmp. *FileName* is a String and must be a complete path to the file including the file extension.

Example: Read a GIF file from the current directory and store the resource number referencing the image in the variable Logo:

```
Logo = PDFDoc1.AddImageFile("mylogo.gif")
```

AddImageBMPHandle (*Handle* As Long) As Long - Method.

Adds an image referenced by a bitmap handle. This can be used to transfer an image to PDFBuilderX directly from another control used for processing images, without the need to save the image to disk and read it back into memory. The return value of the function is the resource identifying number.

For example, to add an image from a VB PictureBox control, the syntax would be in the form:

```
Dim NewRes As Integer

NewRes = PDFDoc1.AddImageBMPHandle(Picture1.Picture.Handle)
```

PDFBuilderX does not take ownership of the handle unless the ReleaseBMPHandle property has been set to False, in which case the original copy of the image will be freed from memory. If an image is copied from the Image property of a VB PictureBox, the handle must not be released as the PictureBox will not clear the memory used for transferring the image and a memory leak will result. In this case, the following code should be used:

```
PDFDoc1.ReleaseBMPHandle = False
NewRes = PDFDoc1.AddImageBMPHandle(Picture1.Image.Handle)
```

It is also necessary to set the ReleaseBMPHandle property to False when transferring an image from Ciansoft TwainControlX, or from the csXImage control from Chestysoft, our recommended control for image processing. Example code for use with TwainControlX is as follows:

```
PDFDoc1.ReleaseBMPHandle = False
NewRes = PDFDoc1.AddImageBMPHandle(Twain1.BMPHandle)
```

AddImageBytes (*ImageData* As Variant) As Long - Method.

Adds an image currently held in memory as an array of bytes. This method might typically be used when images are retrieved from a database as binary data. The return value of the function is the resource identifying number.

Example:

```
Image1 = PDFDoc1.AddImageBytes(Data)
```

ReleaseBMPHandle As Boolean - Read/Write Property.

This property is used to determine whether the handle is released to its original owner when the AddImageBMPHandle function is used. See above examples for further explanation. (Default = True).

ImageReadNumber As Long - Read/Write Property.

When adding an image resource from a TIFF file containing multiple images, this property indicates the number of the image in the file that is to be read. (Default = 1).

Example: Select the 3rd image on a TIFF file to be read using AddImageFile:

```
PDFDoc1.ImageReadNumber = 3
Image1 = PDFDoc1.AddImageFile("multipage.tif")
```

ImageCount (*FileName* As String) As Long - Read-only Property.

Gives the number of images contained within a file. Reading of multiple images is only supported for TIFF files, so this will normally return the value 1 for any other files. *FileName* is a String and must be a complete path to the file including the file extension.

Example:

```
NImages = PDFDoc1.ImageCount("multipage.tif")
```

ImageWidth (*Resource* As Long) As Double - Read-only Property.

The width of the image resource referenced by *Resource* in the units of measure currently set by the property Units.

ImageHeight (*Resource* As Long) As Double - Read-only Property.

The height of the image resource referenced by *Resource* in the units of measure currently set by the property Units.

Example: Find the width and height of an image previously read from file and referenced by the variable Image1:

VB5/6:
```
W = PDFDoc1.ImageWidth(Image1)
H = PDFDoc1.ImageHeight(Image1)
```

VB.NET:
```
W = AxPDFDoc1.get_ImageWidth(Image1)
H = AxPDFDoc1.get_ImageHeight(Image1)
```

SetImageLink (*Page* As Long, *Index* As Long, *Link* As String) - Method.

Attaches a hyperlink to the image referenced by *Index* on the page referenced by *Page*. The parameter *Link* is the URL that will be linked to. *Link* should begin with 'http://' for a URL, or alternatively can be an email address prefixed with 'mailto:'.

The image must not be rotated. If LinkBorder is True, a rectangular border will be displayed around the image.

Example:

VB5/6:
```
PDFDoc1.SetImageLink 1, ImageIndex, "http://www.ciansoft.com"
```

VB.NET:
```
AxPDFDoc1.SetImageLink _
  (1, ImageIndex, "http://www.ciansoft.com")
```

MergeAlpha As Boolean - Read/Write Property.

Image files in PNG format may include an alpha channel containing transparency data. PDFBuilderX does not support alpha transparency, but the image can be made to appear transparent on a plain background by merging the image with its alpha channel. To achieve this effect, this property must be set to True before loading the image with the AddImageFile function. The background colour for the merge must be set using the MergeAlphaColor property. (Default = False).

MergeAlphaColor As OLE_COLOR - Read/Write Property.

The colour to be used for the background when a PNG image is merged with its alpha channel. (Default = White).

3.1. Compression of Image Data

For most purposes, it is not necessary for the user of PDFBuilderX to be concerned about the compression algorithms used for storing images. The default behaviour of the control is appropriate for most situations.

Images stored in a PDF document will be either uncompressed or compressed using one of the following methods:

TxCompressFormat		
cfNone:	0	Uncompressed.
cfGroup4:	1	CCITT Group4 compression. Used only for black and white images.
cfZIP:	2	ZIP (or Flate) compression.
cfJPEG:	3	JPEG compression.

Group4 and ZIP are lossless compression methods which means that the full quality of the image is retained whilst the space occupied by the image on disk is reduced. JPEG compression is a lossy method which sacrifices some image quality for a significant reduction in size and is commonly used for full colour or greyscale photographic images.

The compression method can be set for each possible image type independently using the following properties.

CompressionBW As TxCompressFormat - Read/Write Property.

The compression method to be used for storing black and white images in the document. (Default = cf Group4).

Example: Save black and white images uncompressed:

VB5/6:
```
PDFDoc1.CompressionBW = cfNone
```

VB.NET:
```
AxPDFDoc1.CompressionBW = _
  PDFBuilderXTrial.TxCompressFormat.cfNone
```

CompressionGray As TxCompressFormat - Read/Write Property.

The compression method to be used for storing greyscale images in the document. (Default = cfZIP).

Example: Save greyscale images uncompressed:

VB5/6:
```
PDFDoc1.CompressionGray = cfNone
```

VB.NET:
```
AxPDFDoc1.CompressionGray = _
  PDFBuilderXTrial.TxCompressFormat.cfNone
```

CompressionIndexed As TxCompressFormat - Read/Write Property.

The compression method to be used for storing indexed (paletted) colour images in the document. (Default = cfZIP).

Example: Save indexed images uncompressed:

VB5/6:
```
PDFDoc1.CompressionIndexed = cfNone
```

VB.NET:
```
AxPDFDoc1.CompressionIndexed = _
  PDFBuilderXTrial.TxCompressFormat.cfNone
```

CompressionRGB As TxCompressFormat - Read/Write Property.

The compression method to be used for storing full colour images in the document. (Default = cfZIP).

Example: Save full colour images using JPEG compression:

VB5/6:
```
PDFDoc1.CompressionRGB = cfJPEG
```

VB.NET:
```
AxPDFDoc1.CompressionRGB = _
  PDFBuilderXTrial.TxCompressFormat.cfJPEG
```

UseSourceCompression As Boolean - Read/Write Property.

If this is set to True, the image compression method used in the source file will be retained if possible. This is used to avoid unnecessary decoding of images to satisfy the settings of one of the other compression properties when the image is already compressed using an efficient method.
(Default = True).

Example:

```
PDFDoc1.UseSourceCompression = False
```

4. Working with Graphics

Graphics are drawings made up of combinations of shapes and lines. When drawing a graphic as a resource, the size of the page on which the graphic will eventually be displayed should be kept in mind. The graphic functions require that the co-ordinates of the shapes and lines on the page be specified during drawing and these co-ordinates cannot be changed using the Locate function when the graphic resource is later applied to a page. Also, the graphic cannot be resized using the ScaleObject function nor rotated using the Rotate function.

Typical uses for graphics in a document are the drawing of borders and tables.

CreateGraphic ( ) As Long - Method.

Creates a new graphic resource. The return value of the function is the resource identifying number. The value of CurrentGraphic will automatically be set to this value.

Example:

```
Graphic1 = PDFDoc1.CreateGraphic
```

CurrentGraphic As Long - Read/Write Property.

The reference number of the graphic resource that is currently in use. This must be set before any drawing is done on the graphic by using the DrawLine, Rectangle functions etc.

Example:

```
PDFDoc1.CurrentGraphic = Graphic1
```

DrawLine (*X1* As Double, *Y1* As Double, *X2* As Double, *Y2* As Double) - Method.

Draws a line on the current graphic using the current line settings (LineWidth, LineColor etc.) from co-ordinates *X1*, *Y1* to *X2*, *Y2*.

Example: Draw a line from co-ordinates (10, 100) to (50, 100):

VB5/6:
```
PDFDoc1.DrawLine 10, 100, 50, 100
```

VB.NET:
```
AxPDFDoc1.DrawLine(10.0, 100.0, 50.0, 100.0)
```

Rectangle (*X1* As Double, *Y1* As Double, *X2* As Double, *Y2* As Double) - Method.

Draws a rectangle on the current graphic using the current line and fill settings (LineWidth, LineColor, FillColor) with opposite corners at co-ordinates *X1*, *Y1* and *X2*, *Y2*.

Example: Draw a rectangle with opposite corners at co-ordinates (10, 100) and (50, 200):

VB5/6:
```
PDFDoc1.Rectangle 10, 100, 50, 200
```

VB.NET:
```
AxPDFDoc1.Rectangle(10.0, 100.0, 50.0, 200.0)
```

Circle (*X* As Double, *Y* As Double, *R* As Double) - Method.

Draws a circle on the current graphic using the current line and fill settings (LineWidth, LineColor, FillColor). The circle will be centred at *X*, *Y* and have radius *R*.

Example: Draw a circle with centre at co-ordinates (50, 100) with a radius of 20 units:

VB5/6:
```
PDFDoc1.Circle 50, 100, 20
```

VB.NET:
```
AxPDFDoc1.Circle(50.0, 100.0, 20.0)
```

Ellipse (*X* As Double, *Y* As Double, *RX* As Double, *RY* As Double, *Angle* As Double) - Method.

Draws an ellipse on the current graphic using the current line and fill settings (LineWidth, LineColor, FillColor). The ellipse will be centred at *X*, *Y* and have a radius *RX* in the X-direction, *RY* in the Y-direction and will be rotated counter-clockwise through *Angle* degrees.

Example:

VB5/6:
```
PDFDoc1.Ellipse 50, 100, 20, 30, 45
```

VB.NET:
```
AxPDFDoc1.Ellipse(50.0, 100.0, 20.0, 30.0, 45.0)
```

LineColor As OLE_COLOR - Read/Write Property.

The colour to be used on the current graphic for drawing lines. (Default = Black).

Example:

```
PDFDoc1.LineColor = vbRed
```

LineWidth As Long - Read/Write Property.

The thickness of lines drawn on the current graphic. A value of zero will give a line of the minimum thickness that can be rendered by the output device. (Default = 1).

Example:

```
PDFDoc1.LineWidth = 2
```

FillColor As OLE_COLOR - Read/Write Property .

The colour to be used on the current graphic for filling shapes. (Default = White).

Example:

```
PDFDoc1.FillColor = vbYellow
```

Fill As Boolean - Read/Write Property.

Determines whether shapes (Rectangle, Circle, Ellipse) drawn on the current graphic will be filled. Setting this property to False has the effect of drawing transparent shapes. (Default = True).

5. Working with Text

Blocks of text can be added to the document as resources. Each block consists of any number of single lines of text. The lines are written either underneath each other, or appended to the end of the previous line, depending on the value of AppendText. Within each line of text, a single font, font size and colour must be used, but these properties can be different for each line of text within the block. Line spacing and the maximum line width can also be varied within the block. The whole block will be positioned according to the value of TextAlign.

This means that the procedure for building a block of text should be to set the values of AppendText, TextFont, TextSize, TextColor, TextUnderline, TextLineSpacing, TextMaxWidth, TextSkewX and TextSkewY before writing the first line of text. These properties can then be changed as required before subsequent lines are written. The value of TextAlign can be set at any time prior to generating the PDF document with the SaveToFile command.

Text can be wrapped from one line to the next by setting a value for the TextMaxWidth property.

CreateText ( ) As Long - Method.

Creates a new text resource. The return value of the function is the resource identifying number. The value of CurrentText will automatically be set to this value.

Example:

```
Text1 = PDFDoc1.CreateText
```

CurrentText As Long - Read/Write Property.

The reference number of the text resource that is currently in use. This must be set before writing any text on the text block, or modifying settings such as TextColor or TextFont.

Example:

```
PDFDoc1.CurrentText = Text1
```

WriteText (*Text* As String) - Method.

Adds a single line of text to the current text resource. The text will be written immediately below the last line of text to be written. The first line of text added to the resource will be written at the position on the page defined by a call to the Locate function.

Example:

VB5/6:

```
PDFDoc1.WriteText "A line of text."
```

VB.NET:

```
AxPDFDoc1.WriteText("A line of text.")
```

AppendText As Boolean - Read/Write Property.

If this property is set to True the next line of text will be appended on the end of the previous line, otherwise it will be written below. (Default = False).

Example:

```
PDFDoc1.AppendText = True
```

TextColor As OLE_COLOR - Read/Write Property.

The colour to be used on the current text block for text. (Default = Black).

Example:

```
PDFDoc1.TextColor = vbBlue
```

TextSize As Long - Read/Write Property.

The size of the text for the current text block, in points. If a block of text is scaled using the ScaleObject after it is applied to a page, the size of the text will also be scaled accordingly. (Default = 10).

Example:

```
PDFDoc1.TextSize = 14
```

TextUnderline As Boolean - Read/Write Property.

If this property is set to True the text will be underlined. A True Type font must be used if this property is set to True. (Default = False).

Example:

```
PDFDoc1.TextUnderline = True
```

TextLineSpacing As Double - Read/Write Property.

The size of the gap that will be left between the previous line of text and the next line of text to be written. The value is expressed in terms of percentage of the height of one line of text. (Default = 15.0).

Example: Double space lines by leaving a gap equal to the height of the text:

```
PDFDoc1.TextLineSpacing = 100
```

TextMaxWidth As Double - Read/Write Property.

The maximum width of a line of text in the units of measure defined by Units. By setting a value for this property, long lines of text will be wrapped to use two or more lines. A value of -1 indicates that no maximum width is set and there will be no wrapping of the text. (Default = -1.0).

To use this property and wrap lines of text, a TrueType font must be used. This property has no effect with the 14 standard fonts.

Example:

```
PDFDoc1.TextMaxWidth = 150
```

TextAlign As TxTextAlign - Read/Write Property.

This property determines how the lines of text in a text block are positioned relative to each other. It can take any of the following values:

TxTextAlign		
taLeft: (Default)	0	Left-justified text.
taCenter:	1	Centred text.
taRight:	2	Right-justified text.

This property cannot be set to different values for each line of text within a single text block.

If any of the 14 standard fonts are used in the text block, then the text will always be left-justified. In order to use centred or right-justified text, a TrueType font must be used.

Example: Centre the text:

VB5/6:
```
PDFDoc1.TextAlign = taCenter
```

VB.NET:
```
AxPDFDoc1.TextAlign = PDFBuilderXTrial.TxTextAlign.taCenter
```

TextSkewX As Double - Read/Write Property.

Setting this property to a value other than zero will skew the text by the given number of degrees in the horizontal direction. (Default = 0.0).

Example:

```
PDFDoc1.TextSkewX = 10
```

TextSkewY As Double - Read/Write Property.

Setting this property to a value other than zero will skew the text by the given number of degrees in the vertical direction. The main use of this property is to create italic text when using a font that does not include an in-built italic character set. A typical value to use for italic text is 18.0. (Default = 0.0).

Example: Skew text by 18 degrees to give an italicised effect:

```
PDFDoc1.TextSkewY = 18
```

TextFont As Long - Read/Write Property.

Text can be written using any TrueType font previously added to the document by the AddFont method. Alternatively, any one of the 14 standard Type 1 fonts listed below can be used. The TextFont property indicates the font to be used for the current text resource.

1	Courier
2	Courier-Bold
3	Courier-BoldOblique
4	Courier-Oblique
5 (Default)	Helvetica
6	Helvetica-Bold
7	Helvetica-BoldOblique
8	Helvetica-Oblique
9	Times-Roman
10	Times-Bold
11	Times-Italic
12	Times-BoldItalic
13	Symbol
14	ZapfDingbats

It is strongly recommended that True Type fonts should be used instead of the above standard fonts. With the use of standard fonts there are restrictions on which of the text handling functions described in this section can be used.

Example: Select the standard Courier font:

```
PDFDoc1.TextFont = 1
```

AddFont (*FontName* As String) As Long - Method.

Adds a TrueType font from a file on disk. *FontName* is the full path and name of the font file, which will usually have a .ttf extension. The return value of the function identifies the font and is used to reference it each time it is used in the document.

Example: Add the Arial font:

```
F = PDFDoc1.AddFont("Arial.ttf")
```

EmbedFont (*FontNum* As Long) As Boolean - Read/Write Property.

If this property is set to True for a font that has been added using the AddFont function, the font will be embedded into the PDF document. This ensures that the font will be available to the end user who is viewing or printing the PDF, regardless of whether the chosen font is already installed on their system. If the font is not embedded in the document, and is not available to the end user, the software used to view or print the document will substitute another font. (Default = False).

Note: Font files can be protected by copyright. Check that you have the legal right to embed the font file in a PDF document for distribution before doing so.

Example: This code shows how to add a font, select the font for use in the current text resource, and embed the font file in the PDF document:

VB5/6:

```
F = PDFDoc1.AddFont("Arial.ttf")
PDFDoc1.EmbedFont(F) = True
PDFDoc1.TextFont = F
```

VB.NET:

```
F = AxPDFDoc1.AddFont("Arial.ttf")
AxPDFDoc1.set_EmbedFont(F, True)
AxPDFDoc1.TextFont = F
```

The following read-only properties retrieve information about the space that text will occupy when displayed.

TextLineWidth (*Text* As String) As Double - Read-only Property.

Returns the width that a single line of text will occupy if added using the current values of TextFont and TextSize, in the units of measure currently set by the property Units. A True Type font must be used for this property to return a value. The current value of TextMaxWidth is not taken into consideration when calculating this value.

Example:

```
W = PDFDoc1.TextLineWidth("A line of text.")
```

TextBlockWidth As Double - Read-only Property.

Returns the width that the current text block will occupy when added to a page, in the units of measure currently set by the property Units. This property requires that only True Type fonts are in use in the text block, otherwise the value -1.0 will be returned.

Example:

```
W = PDFDoc1.TextBlockWidth
```

TextBlockHeight As Double - Read-only Property.

Returns the height that the current text block will occupy when added to a page, in the units of measure currently set by the property Units.

Example:

```
H = PDFDoc1.TextBlockHeight
```

5.1. Adding Hyperlinks to Text

Links to URLs (web addresses) can be included in text blocks by calling the WriteLink method. For hyperlinks to be used, there are some restrictions on the text formatting. A True Type font must be used, not a standard font. Also, the text must not be rotated. If these restrictions are not obeyed, then hyperlinks will revert to normal text.

Links can also be attached to images using the SetImageLink method.

WriteLink (*Text* As String, *Link* As String) - Method.

This method works in a similar way to the WriteText command, adding a single line of text at the end of the current text resource. The second parameter *Link* is the URL that will be linked to by that line of text in the document. *Link* should begin with 'http://' for a URL, or alternatively can be an email address prefixed with 'mailto:'

Example:

VB5/6:

```
PDFDoc1.WriteLink "A link to a URL", "http://www.ciansoft.com"
PDFDoc1.WriteLink "An email link", "mailto:info@ciansoft.com"
```

VB.NET:

```
AxPDFDoc1.WriteLink _
  ("A link to a URL", "http://www.ciansoft.com")
AxPDFDoc1.WriteLink _
  ("An email link", "mailto:info@ciansoft.com")
```

LinkStandardFormat As Boolean - Read/Write Property.

If this property is set to True, the appearance of all hyperlinks will be determined by the LinkFont, LinkColor, LinkSize and LinkUnderline properties. If False, then each hyperlink will use the values of the text formatting properties TextFont, TextColor etc. (Default = True).

LinkFont As Long - Read/Write Property.

Sets the font to be used for hyperlinks when LinkStandardFormat is set to True. Usage of fonts is similar to the TextFont property. Standard fonts cannot be used for hyperlinks, so the minimum value of this property is 15, which will be the index number of the first font added to the document using AddFont. (Default = 15).

LinkColor As OLE_COLOR - Read/Write Property.

The colour to be used for hyperlinks when LinkStandardFormat is set to True. (Default = Blue).

LinkSize As Long - Read/Write Property.

The size of the text to be used for hyperlinks when LinkStandardFormat is set to True. (Default = 10).

LinkUnderline As Boolean - Read/Write Property.

Defines whether or not hyperlinks will be underlined when LinkStandardFormat is set to True. (Default = True).

LinkBorder As Boolean - Read/Write Property.

If this property is set to True, hyperlinks will be displayed in the document with a rectangular border. (Default = False).

6. Generating the PDF Document

After the configuration of the PDF document is complete, the document can be generated by saving to disk.

SaveToFile (*FileName* As String) - Method.

Saves the document to disk in PDF format at the path and file name given in *FileName*.

Example:

VB5/6:

```
PDFDoc1.SaveToFile "NewFile.pdf"
```

VB.NET:

```
AxPDFDoc1.SaveToFile("NewFile.pdf")
```

7. General Document Functions

General functions operating at a document level.

Clear ( ) - Method.

Deletes all pages and all resources from the control, allowing a new document to be started.

DeleteAllPages ( ) - Method.

Deletes all pages from the control, but retains all resources for use in a new document.

Units As TxUnitsMeas - Read/Write Property.

The units of measure to be used for sizing and locating all pages and objects in the current document. It is recommended that this property be set to the preferred value before any pages or resources are added to the document, and not subsequently changed. The results can be confusing if resources created using one value for this property are then drawn onto pages using a different value.

This property can take any of the following values:

TxUnitsMeas		
umPoints: (Default)	0	A point is 1/72 of an inch.
umInches:	1	Inches. 1 inch = 72 points.
umCentimeters:	2	Centimeters. 1 cm = 28.3465 points
umMillimeters:	3	Millimeters. 1 mm = 2.83465 points

All PDF documents generated by PDFBuilderX use points as the internal unit of measure. Nevertheless, any of the above options can be used in your application as all conversions will be made automatically.

All co-ordinates used in the Locate function and graphic drawing functions are based on Units and are measured from the bottom-left corner of each page.

CompressContents As Boolean - Read/Write Property.

The content of each page in a PDF file, which includes text, graphics and references to images, is usually compressed. By setting this property to False the compression is disabled. (Default = True).

Important note: *CompressContents* cannot be set to False in the trial version.

The following properties allow general information about the document to be included. They are all strings and their use is self-explanatory. All are empty strings by default.

Title As String- Read/Write Property.

Subject As String- Read/Write Property.

Author As String- Read/Write Property.

Keywords As String- Read/Write Property.

8. Deploying an Application

In order to deploy an application that uses PDFBuilderX you will need to distribute the OCX file, PDFBuilderX.ocx, together with the files that make up your application. This file will need to be registered on the machine running your application and you may wish to use a proprietary installer to do this. When PDFBuilderX was installed on your system our installer will have copied the OCX file to the directory "Program Files\Ciansoft\ PDFBuilderX \", assuming you used the installer and accepted the defaults.

The number of copies of the OCX file that may be distributed is not limited by the licence. In order to use the control in a design environment the licence file, PDFBuilderX.lic, is also required. A separate licensed copy of PDFBuilderX is required for each computer it is used on in the design environment.

9. Revision History

The current version of PDFBuilderX is 2.3

New in Version 1.1

Support for TrueType fonts using the AddFont function.
EmbedFont property.
AddImageBytes function.

New in Version 2.0

Improved functionality for writing text blocks including TextAlign and TextLineSpacing functions and greater flexibility to mix fonts and sizes within a single text block.
Rotate function.
General document information properties (Title, Subject, Author, Keywords).
ImageCount property.

New in Version 2.1

TextMaxWidth property.
TextSkewX and TextSkewY properties.
Circle and Ellipse functions.

New in Version 2.2

Hyperlinks.
TextUnderline property.
TextLineWidth, TextBlockWidth and TextBlockHeight properties.

New in Version 2.3

MergeAlpha & MergeAlphaColor properties.
AppendText property.

10. Alphabetical List of Functions