Table of Contents

Part I Xbase++ PDF Class 1

Part II Overview 1

1 Installing 2
2 Changes 2
3 Acknowledgements 6
4 Demo 6
5 GraDemo 8
6 Charts 10
7 Notes 11

Part III Class Methods 11

1 PDF Creation 11

Document **11**
New 11
Create 11
New Page 12
EndDoc 14
Outline 14
PageMode 15
Password 15
Encryption 15
Permission 16
NumPages 16
GoToPage 16

Text and Fonts **16**
Font 16
FontDef 19
Text 19
mText 20
rText 21
TextArc 22
TextColor 23
ChineseFonts 23

Drawing **23**
Line 23
Box 24
Shape 24
Image 25
LineColor 26
BoxColor 26
LineWidth 27

Charts **27**
BarChart 27
LineChart 29
PieChart 31

Xbase Graxx Methods ... **32**
GraSetAttrArea ... 32
GraSetAttrLine ... 32
GraSetAttrMarker ... 32
GraSetAttrString ... 33
GraSetColor ... 33
GraSetFont ... 33
GraStringAt ... 33
GraPos ... 33
GraQueryTextBox ... 34
GraArc ... 34
GraBox ... 35
GraLine ... 36
GraMarker ... 37
GraPathBegin ... 37
GraPathEnd ... 38
GraPathOutline ... 38
GraPathFill ... 38
TextBox ... 38
XppPDF.ch ... 39
Class Variables ... **40**
2 View & Print ... **42**
Intro ... **42**
View ... **42**
Print ... **44**
RenderToPS ... **44**
RenderToBMP ... **45**
Class Variables ... **45**
Index **0**

1 Xbase++ PDF Class

Created by Softsupply Informatica

Rua Alagoas, 48
01242-000
São Paulo, SP
Brazil
Tel (5511) 3159-1997
Fax (5511) 3255-5224
Email : softsupply@terra.com.br

Contact : Edgar Borger

2 Overview

This class was created to fill the need of Xbase++ users around the world, as noted by the many threads existing in the Xbase News Group about creating PDF output from Xbase++ applications. Several solutions have been presented, but none was a complete one, that could be deployed to customer installations without further steps, like PDF printer drivers, and so on.

This is a very simple class with very little effort in terms of programming, of course implementing it will require code changes, but the benefits are worth it.

An example of a simple program would be :

```
PROCEDURE MAIN

   pdf := xbpPdf():New()                    // Creates the PDF object

   pdf:Create( “Hello.Pdf” )                // Tell the class the file name

   pdf:NewPage()                            // Page size (Clipper style)

   pdf:Font( '10.Courier New' )             // Choose font

   pdf:Text( 1, 1, "Hello World !" )        // write text
```

```
pdf:EndDoc()                    // Close file and write to disk

return
```

As seen here, I use lines and columns in the old Clipper style, also known as TopDown witch should make things easier to users of TD or Express libraries.

2.1 Installing

To install Xbase PDF Class, unzip and copy the DLL files to your working path, select the DLLs from subfolder 180, 182, 190, 190Sl1 or 200 for the correct Xbase version, and xpppdf52.dll, xpppdf53.dll and ot4xb.dll from the main zip directory.
Also copy xppdf51.lib from the subfolder 180, 182, 190, 190Sl1 or 200 to the source directory.
All together you will need 4 DLL, XppPdf51, XppPdf52, XppPdf53 and Ot4Xb available at run time, for the class to work properly.

This file will print a pink "DEMO" diagonal line on every page, this will not show when you get the registered key number.

2.2 Changes

Changes for Version 5.0

- Support for Xbase++ V2.0
- New Renderer engine with improved view and print quality
- Faster processing
- Fixed several minor issues
- New Method TextArc

Changes for Version 4.1

- New management of fonts, the FontDef call is no longer needed, it is still available for backward compatibility reasons.

- Enhanced printing speed and file size.

- Smaller distribution footprint, the XppPDF2.dll is no longer needed, only XppPDF1 and XppPDF3 are now required.

- Fixed problem with margins, top margins are ignored if you use bottom/up coordinates, and are now correctly processed for top/down coordinates systems.

- Fixed problem for password protected files, could not be opened for viewing and/or printing

- New ANSI/DOS translation table for Scandinavian characters.

- New Page Scaling option in the Print Method, options are :
  0 = None
  1 = Fit to paper
  2 = Shrink large pages

Changes for Version 4.0

- New methods to emulate Xbase++ Gra.. functions. See Xbase Graxx Methods.

- New methods now enable Viewing and Printing PDF files, without the use of any external resources.

- New Rendering methods, will render a PDF page to any Xbase++ presentation space or bitmap object.

- Ability to integrate with TD library by Clayton Jones, for version 7 and above, a new button will be available in the printer class, allowing the generated report to be saved as a PDF file with options for Image based copy or elements based copy.

- New XPPPDF.CH preprocessor file, that will automatically translate all Xbase++ Graxx calls to the equivalent XbpPDF methods. See GraDemo.

Changes for Version 3.3

- Improved image handling, including the option to draw a XbpBitmap object.
- Ability to integrate with Top-Down library by Clayton Jones for version 7.0 and above. An optional PDF button will be available in the Print Preview window, allowing the generated report to be saved as a PDF file. PDFs can also be created programmatically without using the Preview window.
- New Coordinate system 100 = Clipper style (lines,columns) Bottom Up
- New option for Shape() Method, with the possibility to fill a shape with a specific color instead of Gray.

Changes for Version 3.2

- Translation routines for DOS/Windows characters conversion in German and Central Europe languages.

Changes for Version 3.1

- New method PageMode for viewing the created document, using your registered PDF reader.
- Translation routines for DOS/Windows characters conversion in Polish and Portuguese languages.
- Encoding options now stays permanent until changed explicitly by Font call
- Better support for embedded fonts, using AFM and PFB files in the FontDef call.
- Minor corrections for box, line and text colors.
- Improved error handling routine

Changes for Version 3

- Underscore is now supported for any font, allowing the underlining of text.

- Enhanced Font support, with the new method FontDef that allows to include any existing True Type font in your document.

- Better justification control with the use of the *width* parameter in the mText method

- Smaller DLLs, with the enhanced font support, the need of Xpppdf3.DLL and Xpppdf4.DLL are no longer needed, so only Xpppdf1 and Xpppdf2 are now present, using less disk space, and easier implementation.

- Faster execution, the process of creating the PDF has been optimized, consuming less time.

- PDF Compression, the use of compression permits the creation a smaller pdf's.

- TWIPS and MILLIMETERS as well as Top/Down variations are now supported as a coordinate system.

0 = CLIPPER	Line, Column, Top Down (Clipper style)
1 = GRA_PU_PIXEL	Pixels, Bottom Up
2 = GRA_PU_LOMETRIC	0.1 Millimeters, Bottom Up
3 = GRA_PU_HIMETRIC	0.01 Millimeters, Bottom Up
4 = GRA_PU_LOENGLISH	0.01 Inches, Bottom Up
5 = GRA_PU_HIENGLISH	0.001 Inches, Bottom Up
6 = GRA_PU_TWIPS	1/1440 Inches, Bottom Up
7 = MILLIMETERS	1 Millimeters, Bottom Up
101 = PIXEL TOP/DOWN	Pixels, Top Down
102 = LOMETRIC TOP/DOWN	0.1 Millimeters, Top Down
103 = HIMETRIC TOP/DOWN	0.01 Millimeters, Top Down
104 = LOENGLISH TOP/DOWN	0.1 Inches, Top Down
105 = HIENGLISH TOP/DOWN	0.01 Inches, Top Down
106 = TWIPS TOP/DOWN	1/1440 Inches, Top Down
107 = MILLIMETERS TOP/DOWN	1 Millimeters, Top Down

- Outline now permits the creation of markers in your document, creating a document outline, that will assist the user of the final document with a better navigation within large (several pages) documents.

- Enhanced documentation with better explanation of methods parameters, as well as more examples on each method.

- Enhanced error routine, that will provide better debugging information in case of error during

execution.

2.3 Acknowledgements

Many thanks to (in alphabetical order) :

Adolf Hebelka

Clayton Jones

Guenter Beyes

Miro Martinidis

Pablo Botella

Randy Howe

Takeshi Kanno

Who helped to develop and test several routines from our class.

The distribution file also uses Ot4Xb.dll library, developed by Pablo Botella, that can be found at http://www.xbwin.com where you have the complete library and docs.

Special thanks to Pablo, and Clayton for the contribution in the development of the class, without your help this would not be possible.

2.4 Demo

This is a sample program that creates the "Demo.PDF" that is included in this package, by analyzing it, you will be able to see how simple its is, and at the same time how powerful this new class is.

```
#include "dll.ch"
#include "gra.ch"
#include "xbp.ch"
#include "common.ch"
#include "xbpdev.ch"
```

```
PROCEDURE MAIN

        fl := 'Demo.Pdf'
        use parts

        pdf := xbppdf():New()
        dc  := pdf:Create( fl )
        pg  := pdf:newpage(XBPPRN_FORM_LETTER)

        pdf:Box( 1, 1, 60, 78 )
        pdf:Image( 2, 2, 5, 77, "Logo.JPG" )
        pdf:LineWidth( 2 )
        pdf:Box( 6, 2, 10, 77, 10)
        pdf:TextColor( GRA_CLR_BLUE )
        pdf:font( '14.Courier New Bold' )
        pdf:Text( 7, 10, "Parts Price List")
        pdf:Text( 7, 60, dtoc(date()) )

        pdf:LineWidth( 1 )
        pdf:Box( 12, 2, 50, 77 )
        pdf:Line( 12, 13, 50, 13 )
        pdf:Line( 12, 54, 50, 54 )
        pdf:Line( 12, 65, 50, 65 )
        pdf:TextColor( GRA_CLR_BLACK )
        pdf:font( '10.Courier New' )
        pdf:text( 12.4,  3, "Prt Nmbr" )
        pdf:text( 12.4, 14, "Description" )
        pdf:text( 12.4, 55, " Available" )
        pdf:text( 12.4, 66, "     Price" )
        pdf:Line( 14, 2, 14, 77 )

        go top
        line := 14
        do while .not. eof()
            pdf:Text( line, 3, number )
            pdf:Text( line, 14, descrip )
            pdf:Text( line, 57, transform( qt, '###,###') )
            pdf:Text( line, 66, transform( price, '###,###.##') )
            line = line + 1
            skip
        enddo


        pdf:BoxColor( GRA_CLR_RED )
        pdf:Box( 52, 6, 57, 73, 25 )
```

```
            pdf:TextColor( GRA_CLR_DARKBLUE )
            pdf:font( '20.Courier New' )
            pdf:Text( 52.5,  9, 'Demo Created by Softsupply Informatica' )
            pdf:TextColor( GRA_CLR_BLACK )
            pdf:font( '8.Courier New Italic' )
            pdf:Text( 54,  9, 'Contact :' )
            pdf:Text( 54, 44, 'eMail :' )
            pdf:Text( 55,  9, 'Phone   :' )
            pdf:Text( 55, 44, 'Fax   :' )
            pdf:font( '12.Courier New' )
            pdf:Text( 54, 15, 'Edgar Borger' )
            pdf:Text( 55, 15, '(5511) 3159-1997' )
            pdf:Text( 55, 49, '(5511) 3255-5224' )
            pdf:TextColor( GRA_CLR_BLUE )
            pdf:Text( 54, 49, 'eborger@terra.com.br' )

            pdf:enddoc(.t.)
            close data
  RETURN
```

2.5 GraDemo

This sample program HELLO.PRG, illustrates the use of the XPPPDF.CH preprocessor file to assist in the conversion of existing reports to use the XbpPDF class and create PDF files.

```
#include "xbp.ch"
#include "gra.ch"
#include "common.ch"
#include "xbpdev.ch"
#include "appevent.ch"
#include "xpppdf.ch"
#pragma  library("xppPDF1.lib")


PROCEDURE MAIN

aSize := { 590, 780 }
aPos  := { 0, 50 }
oDlg := xbpDialog():New( AppDesktop(), , aPos, {612, 822} )
oDlg:Create()

oPs := oDlg:DrawingArea:LockPS()
oFont := xbpFont():New( oPs )
oFont:Create( '24.Times New Roman' )

oPdf := xbppdf():New()
oPdf:Create( "gra.pdf" )
```

```
oPdf:NewPage( , , , , ,1 )

Report( oPdf )
oPdf:EndDoc(.t.)

Report( oPs )

nEvent := 0
do while nEvent <> xbeP_Close
        nEvent := AppEvent( @mp1, @mp2, @obr )
        obr:handleEvent( nEvent, mp1, mp2 )
Enddo

Return

Function Report( oPs )
LOCAL aAttra

        aAttra := array(GRA_AA_COUNT)

        aAttra[GRA_AA_COLOR] := GRA_CLR_WHITE
        GraSetAttrArea( oPs, aAttra )
        GraBox( oPs, {10,10}, aSize, GRA_OUTLINEFILL)

        aAttra[GRA_AA_COLOR] := GRA_CLR_YELLOW
        GraSetAttrArea( oPs, aAttra )
        GraBox( oPs, {100, 100}, {512, 692}, GRA_OUTLINEFILL)

        GraSetFont( oPs, oFont )
        GraSetColor( oPs, GRA_CLR_BLUE )
        GraStringAt( oPs, {230, 400}, "Hello world!" )

RETURN(.t.)

//  This function ReportPDF bellow, is equivalent to function Report above with a PDF object as a
parameter, XbpPDF class will analyze the object
//  parameter, and if it is a XbpPDF object will call the appropriate class methods instead of normal
Xbase++ functions.
//
// Note that if you call the method directly, you should not pass the PDF object as a parameter.

Function ReportPDF( )
LOCAL aAttra

        aAttra := array(GRA_AA_COUNT)

        aAttra[GRA_AA_COLOR] := GRA_CLR_WHITE
        oPdf:GraSetAttrArea( aAttra )
        oPdf:GraBox( {10,10}, aSize, GRA_OUTLINEFILL)

        aAttra[GRA_AA_COLOR] := GRA_CLR_YELLOW
        oPdf:GraSetAttrArea( aAttra )
        oPdf:GraBox( {100, 100}, {512, 692}, GRA_OUTLINEFILL)
```

```
        oPdf:GraSetFont( oFont )
        oPdf:GraSetColor( GRA_CLR_BLUE )
        oPdf:GraStringAt( {230, 400}, "Hello world!" )

RETURN(.t.)
```

2.6 Charts

This is a sample program that creates the "Charts.PDF" that is included in this package, by analyzing it, you will be able to see how simple its is, and at the same time how powerful this new class is.

```
PROCEDURE MAIN

        fl := 'Charts.Pdf
        pdf := xbppdf():New()
        dc  := pdf:Create( fl )
        pg  := pdf:newpage(XBPPRN_FORM_LETTER)

        pdf:Font('12.Courier New Bold')
        pdf:Text(0.5,2,"Pie Chart Sample")
        data := {5,6,1,3,8,9,4,6,1,7}
        titu := {'1','22','333','4444','55555','666666','7777777','8888888','99999','000'}
        pdf:PieChart(15,30,5,data,titu,"Pie Chart Sample" )

        pdf:Font('12.Courier New Bold')
        pdf:Text(24.5,2,"Line Chart Sample")
        data := { {5,6,1,3,8,9,4,6,1,7}, {8,3,7,10,6,8,2,1,3,4} }
        titu := {'A','B','C','D','E','F','G','H','I','J'}
        pdf:LineChart(44,10,14,50,data,titu,"Line Chart Sample", {GRA_CLR_BLUE,
        GRA_CLR_RED} )

        pdf:Font('12.Courier New Bold')
        pdf:Text(45.5,2,"Bar Chart Sample")
        data := {5,6,0,3,8,9,4,6,1,7}
        titu := {'A','B','C','D','E','F','G','H','I','J'}
        pdf:BarChart(65,10,14,55,data,titu,"Bar Chart Sample", GRA_CLR_BLUE,
        GRA_CLR_RED )

        pdf:enddoc(.t.)

RETURN
```

2.7 Notes

1. All color references are in the GraSetColor format and can be used from GRA.CH, additional colors can be created and used with GraMakeRGBColor() function.

2. Page Size and orientation can be used from XBPDEV.CH

3. Lines and Columns for all functions are according to the chosen coordinate system, in the Clipper/ Text style (Coordinate System 0), the 0,0 point is top leftmost position, all others are Xbase style, meaning that the 0,0 point is at the bottom leftmost position of the page.

---

3 Class Methods

Available class methods.

3.1 PDF Creation

Class Methods to create PDF files.

3.1.1 Document

Those are the document handling methods.

3.1.1.1 New

- **New(** *cKey* **)** => Creates the pdf object. No parameters are needed, and returns the new created object.

cKey is the numeric key as a string, to remove the "DEMO" restriction

---

3.1.1.2 Create

- **Create(** *filename, oPrt* **)** => Declares the name of the output file, if a file with that name already exists on your disk, it will be overwritten by the new one. This method is mandatory.

  Filename (character) : is the full file name to be created.

  oPrt (object) : is an optional Xbase printer object (XbpPrinter) to be used with the :Print() method, if none is given, then a printer dialog will be presented during the printing of the PDF file.

  Example : oPdf:Create(“Sample.PDF”)

3.1.1.3 NewPage

- **NewPage(** *[page size], [orientation], [lines], [columns], [user size], [Coord System], [left margin], [top margin]* **)** => starts a new page, specifying the page size, orientation and the coordinates system used.

Page size (numeric) : is a page format reference as in XBPDEV.CH. The page size defaults to XBPPRN_FORMAT_LETTER, valid page sizes are :

Letter	230 x 275	8 1/2 x 11
LetterSmall	230 x 279	8 1/2 x 11
Tabloid	279 x 432	11 x 17
Ledger	432 x 279	17 x 11
Legal	203 x 356	8 1/2 x 14
Statement	127 x 203	5 1/2 x 8 1/2
Executive	184 x 254	7 1/4 x 10 1/2
A3	297 x 420	
A4	210 x 297	
A4 Small	210 x 297	
A5	148 x 210	
B4	250 x 354	
B5	182 x 257	
Folio	203 x 330	8 1/2 x 13
Quarto	215 x 275	
10 x 14 in	254 x 356	10 x 14
11 x 17 in	279 x 432	11 x 17
Note	203 x 279	8 1/2 x 11
Envelope #9	98 x 225	3 7/8 x 8 7/8
Envelope #10	105 x 229	4 1/8 x 9 1/2
Envelope #11	102 x 264	4 1/2 x 10 3/8
Envelope #12	102 x 179	4 3/4 x 11
Envelope #14	127 x 279	5 x 11 1/2
C size sheet	432 x 558	17 x 22
D size sheet	558 x 864	22 x 34
E size sheet	864 x 1118	34 x 44
DL	110 x 220	
C5	162 x 229	
C3	324 x 458	
C4	229 x 324	
C6	114 x 162	
C65	114 x 229	
B4	250 x 353	
B5	176 x 250	
B6	176 x 125	

Envelope Italy	110 x 230	
Monarch	98 x 191	3 7/8 x 7 1/2
Envelope Personal	92 x 165	3 5/8 x 6 1/2
US Standard Fanfold	378 x 279	14 7/8 x 11
German Standard Fanfold	216 x 305	8 1/2 x 12
German Legal Fanfold	216 x 330	8 1/2 x 13

Orientation (numeric) : is the page orientation, portrait or landscape, and defaults to XBPPRN_ORIENT_PORTRAIT.

Lines (numeric) : is the number of lines in the page, this number is related to the chosen coordinate system (see bellow). Default is 66 lines per page.

Columns (numeric) : the number of character columns in each line, this number is also related to the coordinate system. Default is 80 characters per line.

User size (array) : is a two element array that specifies page dimensions, first element is horizontal size, and second element is vertical size. This parameter is only used for *page size* XBPPRN_FORMAT_USER, and the measures should be in 1/72 inches. A page with 3 x 5 inches should be informed as { 216, 360 }.

Coordinate system (numeric): the coordinates system specifies the type of coordinates (line, column) passed to the class, they can be :

0 = CLIPPER	Line, Column, Top Down (Clipper style)
1 = GRA_PU_PIXEL	Pixels, Bottom Up
2 = GRA_PU_LOMETRIC	0.1 Millimeters, Bottom Up
3 = GRA_PU_HIMETRIC	0.01 Millimeters, Bottom Up
4 = GRA_PU_LOENGLISH	0.01 Inches, Bottom Up
5 = GRA_PU_HIENGLISH	0.001 Inches, Bottom Up
6 = GRA_PU_TWIPS	1/1440 Inches, Bottom Up
7 = MILLIMETERS	1 Millimeters, Bottom Up
101 = PIXEL TOP/DOWN	Pixels, Top Down
102 = LOMETRIC TOP/DOWN	0.1 Millimeters, Top Down
103 = HIMETRIC TOP/DOWN	0.01 Millimeters, Top Down
104 = LOENGLISH TOP/DOWN	0.1 Inches, Top Down
105 = HIENGLISH TOP/DOWN	0.01 Inches, Top Down
106 = TWIPS TOP/DOWN	1/1440 Inches, Top Down
107 = MILLIMETERS TOP/DOWN	1 Millimeters, Top Down

left margin (numeric) : left margin for the document, default is one character.

top margin (numeric) : top margin for the document, default is one line. The top margin is ignored if you are using a bottom/up coordinate system.

Examples : oPdf:NewPage(XBPPRN_FORMAT_A4) will create a page in the A4 format, oriented portrait, with 66 lines and 80 columns, using line/column coordinate system.

oPdf:NewPage(XBPPRN_FORMAT_LETTER, , 660, 800) will create portrait oriented page, letter size, with standard line/column coordinate system, but with 660 lines per page and 800 characters per line, creating a smaller line/character feed, that could provide better precision in positioning elements on the page.

oPdf:NewPage(XBPPRN_FORMAT_LETTER, , , , 3) this page will have letter size, but will be using HIMETRIC coordinate system.

oPdf:NewPage(XBPPRN_FORMAT_USER, , 30, 70, { 504, 360 }) this page will be sized at 5 x 7 inches providing 30 lines with 70 characters in each line, notice that the horizontal dimension (504) is larger that the vertical (360), indicating that the paper will be at a landscape position.

3.1.1.4 EndDoc

- **EndDoc(** *lView* **)** => finish all processing, and saves the file to disk.

l*View (logical)* : View is a logical parameter (.t. or .f.) to view the created PDF (using registered PDF reader)

Instead of using the registered PDF reader, you can use the :View() method to display the created PDF file, leave this parameter as False (default) and use the :View() method just after the :EndDoc() call, or change the content of :lView to True (oPdf:lView := .t.) before to call to :EndDoc()

3.1.1.5 Outline

- **Outline(** *line, parent, title* **)** => will create an outline (markers) of the document giving a title for each page at the outline.

Line (numeric) : is the vertical position within the page.

Parent (numeric) : is the parent for this title in the outline tree. The first time used will create a root parent, any outline linked to the root should be specified as NIL

Title(character) : is the title that will show at the outline.

Returns : This method will return the created outline, that can be used as future parent for other outline titles.

Example : pg1 := oPdf:Outline(1, NIL, “Page 1”)
Ou1 := oPdf:Outline(5, pg1, "Outline 1")

3.1.1.6 PageMode

- **PageMode(** *mode* **)** => defines page viewing mode for PDF Reader.

Mode (numerical) : viewing mode (using registered PDF reader) can be as follows :

0 (none)	Use default PDF reader mode
1 (outline)	Show document Outline page
2 (Thumbnail)	Show pages thumbnails
3 (Full Screen)	Show document in Full Screen mode

this method can be called anywhere between Create() and EndDoc() calls.

3.1.1.7 Password

- **Password(** *ownerpass, userpass* **)** => defines passwords for PDF Reader.

Ownerpass (character) : defines a owner password for the PDF file. The owner password can not be null.

Userpass (character) : defines a user password for the PDF file. The user password can be any length, including null, but if present, must be different from the owner password.

If passwords are defined for the file, the PDF will be encrypted.

This method can be called anywhere between Create() and EndDoc() calls.

3.1.1.8 Encryption

- **Encryption(** *mode, keylen* **)** => defines encryption mode for PDF Reader.

Mode (numeric) : defines encryption mode, valid parameters are 2 for 40 bit encryption, and 3 for 128 bit encryption.

Keylen (numeric) : defines key length for encryption, if mode is 2 (40 bit) key length must be 5, using mode 3 (128 bit) key length can be 5 for 40 bit encryption, or 16 for 128 bit encryption.

This method can be called anywhere between Create() and EndDoc() calls.

3.1.1.9 Permission

- **Permission(** *level* **)** => defines permission levels for PDF Reader.

  Level (numeric) : defines the level of permissions for the user, valid values are :

value	description
0 ENABLE READ	user can read the document.
4 ENABLE PRINT	user can print the document.
8 ENABLE EDIT_ALL	user can edit the contents of the document other than annotations, form fields.
16 ENABLE COPY	user can copy the text and the graphics of the document.
32 ENABLE EDIT	user can add or modify the annotations and form fields of the document.

  Different values can be combined by adding values, oPdf:Permission(4+8+16+32) will give full permission to the user, and is the default.

  This method can be called anywhere between Create() and EndDoc() calls.

3.1.1.10 NumPages

- **NumPages( )** => returns the number of pages in the document.

  This method can be called anywhere between Create() and EndDoc() calls.

3.1.1.11 GoToPage

- **GoToPage(** *n* **)** => jumps to page *n*

  jumps to page *n*, whre n is any number between 1 and the total number of pages in the document, to get the number of pages see the NumPages() method.

  This method can be called anywhere between Create() and EndDoc() calls.

3.1.2 Text and Fonts

Those are text methods, with them you can load any kind of text into your document.

3.1.2.1 Font

- **Font(** *[compound fontname], [Encoding], [Embed], [translate], [lfixed], [nScale]* **)** => specify the wanted font according to Xbase syntax, nominal point size followed by dot followed by font name. If none provided defaults to “10.Courier New”, with WinAnsiEncoding. Basic fonts are included in the Xbase PDF Class, but with version 3, any **True Type** font can be used to create your document, (see FontDef bellow), also any size can be used, there is no limitation to maximum size or the number of fonts used in one document. The existing fonts can be combined with Bold and or Italic, and “Underscore” can be added to any combination, providing

underlining of text.

Compound fontname (character) : is the font compound name, point size, followed by font family name. Valid fonts are Arial, Courier New, Times New Roman, Helvetica, Tahoma, Verdana, Symbols and ZapfDingbats fonts, combined to Bold, Italic or Bold Italic. Underscore can be added to any of the available fonts, like Font("10.Arial Underscore") or Font("12.Helvetica Bold Underscore"). Other fonts might be used using the **FontDef** method.

Available fonts are (see Fonts.PDF) :

Arial	a e i o u A E I O U - 1 2 3 4 5
Arial Bold	**a e i o u A E I O U - 1 2 3 4 5**
Arial Bold Italic	***a e i o u A E I O U - 1 2 3 4 5***
Arial Italic	*a e i o u A E I O U - 1 2 3 4 5*
Courier New	a e i o u A E I O U - 1 2 3 4 5
Courier New Bold	**a e i o u A E I O U - 1 2 3 4 5**
Courier New Bold Italic	***a e i o u A E I O U - 1 2 3 4 5***
Courier New Italic	*a e i o u A E I O U - 1 2 3 4 5*
Helvetica	a e i o u A E I O U - 1 2 3 4 5
Helvetica Bold	**a e i o u A E I O U - 1 2 3 4 5**
Helvetica Bold Italic	***a e i o u A E I O U - 1 2 3 4 5***
Helvetica Italic	*a e i o u A E I O U - 1 2 3 4 5*
Symbol	α ε ι ο υ Α Ε Ι Ο Υ – 1 2 3 4 5
Tahoma	a e i o u A E I O U - 1 2 3 4 5
Tahoma Bold	**a e i o u A E I O U - 1 2 3 4 5**
Times New Roman	a e i o u A E I O U - 1 2 3 4 5
Times New Roman Bold Italic	***a e i o u A E I O U - 1 2 3 4 5***
Times New Roman Italic	*a e i o u A E I O U - 1 2 3 4 5*
Times New Roman Bold	**a e i o u A E I O U - 1 2 3 4 5**
Verdana	a e i o u A E I O U - 1 2 3 4 5
Verdana Bold	**a e i o u A E I O U - 1 2 3 4 5**
Verdana Bold Italic	***a e i o u A E I O U - 1 2 3 4 5***
Verdana Italic	*a e i o u A E I O U - 1 2 3 4 5*
ZapfDingbats	a e i o u A E I O U - 1 2 3 4 5

Encoding (character) : This parameter can be used to specify different encoding for the chosen font, one of the following values should be used :

Encoding	**Comments**
WinAnsiEncoding	Normal Windows encoding
MacRomanEncoding	Mac Roman encoding
PDFDocEncoding	PDF Doc encoding

ISO8859_2	(Latin 2) Slavic languages of Central Europe
ISO8859_3	(Latin 3) Esperanto, Maltese
ISO8859_4	(Latin 4) Estonian, Baltic Languages, Greelandic
ISO8859_5	Bulgarian, Russian, Serbian
ISO8859_6	Arabic
ISO8859_7	Modern Greek
ISO8859_8	Hebrew
ISO8859_9	(Latin 5) Western Europe, Turkish
ISO8859_10	(Latin 6) Nordic languages
ISO8859_11	Thai Language
ISO8859_13	(Latin 7) Baltic languages
ISO8859_14	(Latin 8) Celtic
ISO8859_15	(Latin 9) Adds EURO as well as French, Finish to Latin 1
ISO8859_16	(Latin 10) Hungarian, Polish, Romanian, Slovenian
CP1250	Central Europe
CP1251	Cyrillic
CP1253	Greek
CP1254	Turkish
CP1255	Hebrew
CP1256	Arabic
CP1257	Baltic
CP1258	Vietnam

Embed (logical) : specify if the font file is to be embedded in the document, or not. Embedding the font makes the document larger in size, but assures that the font will be respected, if not embedded, the reader may chose a similar font for displaying the document.

Translate (character) : specify a translation routine from DOS characters to Windows characters, according to national accented characters, valid options are : PTB for Portuguese, POL for Central European and GRM for German characters

lFixed (logical) : specify if the character width should have a fixed size, that will be calculated according to font/page size, or if FALSE will use proportional character width according to font/page size.

nScale (numerical) : sets the scale for the font, 100 (default) is the normal scale, and it can be enlarged or reduced to fit the text in the desired space.

Examples : oPdf:Font(“10.Arial”) will print with Arial fonts using 10 point size characters like
=> abcdefABCDEF12345.

oPdf:Font(“16.Tahoma Bold Underscore”, “CP1250”) will use 16 size characters in Tahoma font, bold and underlined, special characters will be used from the

central European alphabet, like => **abcdefABCDEF12345ašåý**

3.1.2.2 FontDef

- **FontDef(** *family name, file name, embed* **)** => this powerful method enables you to use any font installed in your system. By defining the font family name, and its file container, you can use any existing font, for non Latin characters, bar codes, symbols, etc.

  Family name (character) : is the font compound name that you will be using latter to create your document.

  File name (character) : is the name of the file for that specified font. This file must be in the %SystemRoot%\Fonts folder or at the current working directory and must be a either a TTF format file, or a AFM/PFB file. If the file is not found at usage time, font will default to Courier New.

  Embed (logical) : indicates if the font should be embedded (incorporated) to the document, if TRUE, the TTF file will be embedded to the PDF file, making it larger in size, but with a better font definition, if FALSE, the font will no be incorporated to the file, making it smaller, but eventually having a substitute font if the same font is not installed at viewing time.

  Examples : oPdf:FontDef("3-9 Barcode", "FREE3OF9X.TTF") this will add the barcode TTF font to your pdf, enabling you to print barcode data.
  oPdf:FontDef("Avant Garde Book", "avgardn.ttf") will allow you to use the Avant Garde font in your document.

  Observation : you must define all the variations that you want to use in a font, there are separate files for Bold, Italic and Bold Italic fonts, and they must be defined one by one to XbpPDF.

3.1.2.3 Text

- **Text(** *line, column, string* **)** => prints a string of text on the page, at line and column position that are calculated according to page size.

  Line (numeric) : is the vertical position where the text will be outputted, this position must be according to the coordinate system chosen for your document (see NewPage above). If *line* is NIL, will continue to print at current line

  Column (numeric) : is the horizontal position within the page for the text, according to the

coordinate system specified in the NewPage method. If *column* is NIL, will print at the next character position.

String (character) : is the string of characters to be outputted at the specified position, if the string does not fit on the line, it will be lost, no line wrapping is done with this method, for this use the mText method.

Examples : oPdf:Text(5, 1, "this is a sample") with print "this is a sample" on line 5, column 1, assuming the Line/Column coordinate system is in use.

3.1.2.4 mText

- **Mtext(** *init line, init col, text, horizontal alignment, width* **)** => Multi line text, automatically fits the given text into the specified coordinates, aligning the text horizontally relative to *init col*.

Init line (numeric) : specifies the vertical position according to the coordinate system for the document, where text output will begin.

Init col (numeric) : specifies the horizontal position for the text output. This parameters is used by XbpPDF according to the *horizontal alignment* chosen. On *left* alignment (standard) and on *justified* the left margin of the text is aligned to the specified column, on *right* alignment the end of the text lines are aligned to the column, and on *center* the text is centralized on the column position.

Text (characters) : is the text to be outputted at the document, this text could be several lines of text, separated by NL ($0D0A) characters, or in the case of *justified* alignment one long string (there is no limit). Text should conform to memo fields formats.

Horizontal alignment (numeric) : is the type of alignment to be performed by XbpPDF on adding the text to your document.

Possible values for horizontal alignment are :	Left	=> 0 (Default)
	Right	=> 1
	Center	=> 2
	Justified	=> 3

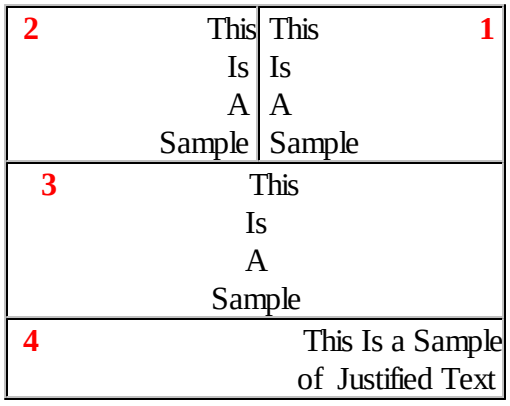
Width (numeric) : on *justified* the optional parameter *width* specifies the line width, if this parameter is not present, the width will be of the longest line present at the text.

Examples : for this example the text will be "This is a sample" (one word on each line)

```
Text := "This" + nl + "is" + nl + "a" + "Sample"
oPdf:Box( 10, 0, 18, 20 )
```

```
oPdf:Line( 14, 0, 14, 20 )
oPdf:Line( 10, 10, 14. 10 )

oPdf:mText( 10, 10, text, 0 )          (1)
oPdf:mText( 10, 10, text, 1 )          (2)
oPdf:mText( 14, 10, text, 2 )          (3)
oPdf:mText( 18, 10, text, 3, 16 )      (4)
```



Notice that all mText calls use column 10, so the positioning of the text varies according to the chosen alignment.

Text position is calculated according to font/page size specifications, and also according to fixed or proportional character spacing, proportional character spacing may not align text correctly, so if you use right or center alignment, please check your character width settings, this setting can be changed either in the oPdf:Font() call, or changing the oPdf:lFixed var contents.

3.1.2.5 rText

- **RText(** *line, column, string, angle* **)** => prints a string of text on the page, rotating to the specified *angle*, at line and column position that are calculated according to page size.

  Line (numeric) : Vertical position of text according to coordinate system.

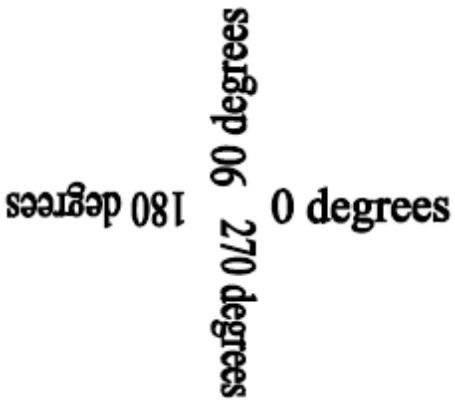
  Column (numeric) : Horizontal position of text according to coordinate system.

  String (character) : Text to be included in the document.

  Angle (numeric) : Angle of rotation, ranging from 0 to 360, starting at the normal text position and rotating counter clockwise back to an horizontal position.

  Examples : oPdf:rText(15, 15, “normal”, 0)

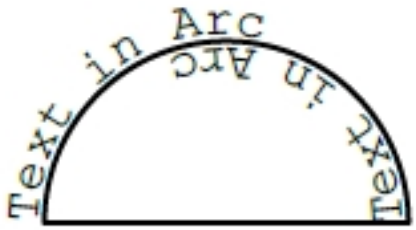
```
oPdf:rText( 15, 25, “90 degrees”, 90)
oPdf:rText( 15, 25, “180 degrees”, 180)
oPdf:rText( 15, 25, “270 degrees”, 270)
```



3.1.2.6 TextArc

- **TextArc(** *nXpos, nYpos, nRadius, nAngle, cText, nOpt* **)** => prints a string of text on the page, in a arc acording to the specified *radius* and *angle*, at Xpos and Ypos position that are calculated according to page size.

nXpos (numeric) :	Horizontal position of arc center, according to coordinate system.
nXpos (numeric) :	Vertical position of arc center according to coordinate system.
nRadius (numeric) :	Radius of arc according to coordinate system.
nAngle (numeric) :	Starting angle to start the text string, ranging from 0 to 360, starting at the normal text position and rotating counter clockwise back to an horizontal position.
cText (character) :	Text to be included in the document.
nOpt (numeric) :	0 => Draw the text outside the arc in a clockwise direction. 1 => Draw the text inside the arc in a anti-clockwise direction.
Examples :	oPdf:GraArc({60, 30}, 5, , 0, 180) oPdf:TextArc(60, 30, 5.2, 180, "Text in Arc", 0) oPdf:TextArc(60, 30, 4.8, 0, "Text in Arc", 1)



3.1.2.7 TextColor

- **TextColor(** [*color]* **)** => specify the color to the following text.

  color (numeric) : informs the color for the following text, constants from GRA.CH can be used in the form of GRA_CLR_… or any other RGB number (created with GraMakeRGBColor()) can be used. Default is GRA_CLR_BLACK.

3.1.2.8 ChineseFonts

ChineseFonts([en] **)** => enable the use of chinese MingLiU font.

this command will make available the MingLiU chinese font with variations of bold and italic, the usage should be pdf:Font("anysize.MingLiU"), or "MingLiU,Bold", "MingLiU,Italic" and "MingLiU,BoldItalic" using "," (comma) instead of space between the font name and the variation.

3.1.3 Drawing

This group of methods can be used to load drawing and images to your document.

3.1.3.1 Line

- **Line(** *init line, init col, final line, final col* **)** => draws a line from initial line, initial column position to final line, final column position.

  Init line (numeric) : Vertical position of line start according to coordinate system.

  Init col (numeric) : Horizontal position to start drawing the line.

  Final line (numeric) : Vertical position of line ending.

Final col (numeric) : Horizontal position of line ending.

3.1.3.2 Box

- **Box(** *init line, init col, final line, final col, [fill percent]* **)** => draws a box (rectangle) from initial line, initial column position to final line, final column position, and optionally fills the create shape with gray according to fill percent value.

  Init line (numeric) : Vertical position of box corner according to coordinate system.

  Init col (numeric) : Horizontal position of box corner to start drawing the box.

  Final line (numeric) : Vertical position of opposite box corner.

  Final col (numeric) : Horizontal position of opposite box corner.

  Fill percent (numeric) : numeric value in the range of 0 to 100, indicating the amount of gray that should be used to fill the box, the higher the number the darker will be the filling, a value between 5 to 20 will provide a gray area as a background, suitable for foreground text, or other data. Default for fill percent is 0 (none).

3.1.3.3 Shape

- **Shape(** *points array, line color, [fill percent], [fill color]* **)** => creates a geometric shape, defined by and array of points {{l1,c1},{l2,c2}…..}, drawn in the specified color (default is black), and optionally fills the create shape with gray according to fill percent value.

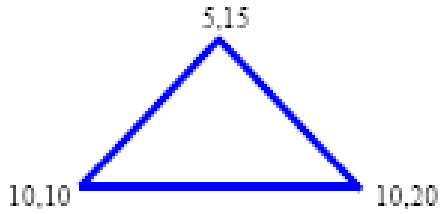
  Points array (array) : An array of numbers pairs, giving vertical and horizontal positions, for every point in the shape, the quantity of pairs is unlimited, allowing the creation of complex figures of any kind.

  Line color (numeric) : informs the color of the surrounding line for the shape, constants from GRA.CH can be used in the form of GRA_CLR_… or any other RGB number (created with GraMakeRGBColor()) can be used.

  Fill percent (numeric) : numeric value in the range of 0 to 100, indicating the amount of gray that should be used to fill the shape, the higher the number the darker will be the filling, a value between 5 to 20 will provide a gray area as a background, suitable for foreground text, or other data. Default for fill percent is 0 (none).

Fill color (numeric) : sets the color to be used to fill the shape, to use this option, you should set the the *Fill percent* to -1 (minus one), and the shape will be filled with the specified color instead of gray.

Example : oPdf:Shape({ {10, 10}, {5, 15}, {10, 20} }, GRA_CLR_BLUE) will create a blue triangle at the specified points.



3.1.3.4 Image

- **Image(** *init line, init col, final line, final col, image name, [ratio], compress* **)** => draws an image from initial line, initial column position to final line, final column position. Ratio indicates if the original ratio of the image is to be kept, or if the image should fill the entire area. Image can be JPEG, TIFF, BMP, PNG or GIF, file, and extension is mandatory.

  Init line (numeric) : Vertical position of image corner according to coordinate system, default is 0.

  Init col (numeric) : Horizontal position of image corner to start drawing the box, default is 0.

  Final line (numeric) : Vertical position of opposite image corner, default is vertical page size (the hole page).

  Final col (numeric) : Horizontal position of opposite image corner, default is horizontal page size (the hole page).

  Image name (character/Bitmap Object) : Name of the image file to be included in the document, if the file is in a different folder as the current one, the full path should be given. Also the file type (JPEG, BMP or PNG) should be included in the name. As an alternative and XbpBitmap object can be passed as parameter, allowing an easy conversion from a XbpPresSpace into a PDF file.

  Ratio (logical) : Ratio is true or false (.t. or .f.) indicating if the original ratio of the image proportion is to be kept, or if the image should fill the entire area. If ratio is true, then *initial line* and *initial col* are used to calculate the positioning of the image, and the height will be

calculated proportional to the width of the image. If the image is passed as an XbpBitmap object then ratio TRUE will display the image in the whole page.

Compress (numeric) : is a number from 0 to 100, indicating the compression to be done on the image, lower numbers results in better quality, but a larger PDF file, larger numbers results in less quality, but smaller file sizes.

Example : oPdf:Image(5, 5, 20, 60, "Image.JPG") will draw the contents of Image.JPG into line 5, column 5 thru line 20, column 60

oPDF:Image(, , , , oBmp, .t.) will draw the contents of the Bitmap object oBmp into the whole page.

Note : This new implementation will allow the integration of XbasePDF class and TD library (version 7 and above) developed by Clayton Jones. TD is a wonderful tool for Xbase++ developers, and can be found at http://www.cjcom.net.

3.1.3.5 LineColor

- **LineColor(** *[color]* **)** => specify the color to the following lines.

Line color (numeric) : informs the color of the following lines, constants from GRA.CH can be used in the form of GRA_CLR_… or any other RGB number (created with GraMakeRGBColor()) can be used. Default is GRA_CLR_BLACK.

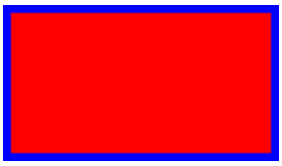
3.1.3.6 BoxColor

- **BoxColor(** *[color],[fill color]* **)** => specify the colors to the following box. First parameter is color for the box lines, and second parameter is color for filling the box (background).

color (numeric) : informs the color drawing the following boxes, constants from GRA.CH can be used in the form of GRA_CLR_… or any other RGB number (created with GraMakeRGBColor()) can be used. Default is GRA_CLR_BLACK.

Fill color (numeric) : informs the color for the filling of boxes, constants from GRA.CH can be used in the form of GRA_CLR_… or any other RGB number (created with GraMakeRGBColor()) can be used. Default is GRA_CLR_WHITE (no filling).

Examples : oPdf:BoxColor(GRA_CLR_BLUE, GRA_CLR_RED) followed by a oPdf:Box() call, will draw the a blue box with a red inside.



3.1.3.7 LineWidth

- **LineWidth(** *[width]* **)** => the width of the following lines.

  Width (numeric) : the width for the lines, starting with 1 and up, the higher the number, the thicker the line. Default is 1.

3.1.4 Charts

Those are Chart drawing methods. They can be used to create simple charts in your document, and are build using Box Lines, Shapes and Text methods.

3.1.4.1 BarChart

- **BarChart(** *init line, init col, height, width, data, data titles, chart title, color1, color2* **)** => draws a bar chart, at position *init line, init col* with size *height*, *width*, using the values in *data* array, for each bar, a title in *data titles* is displayed. Bars are drawn using *color1* for outline, and *color2* for filling. A title can be displayed at the top/center of the chart using the *chart title* parameter. See a sample at Charts.PDF

  Init line (numeric) : Vertical position of chart corner according to coordinate system.

  Init col (numeric) : Horizontal position of corner to start drawing the chart.

  Height (numeric) : Height of box containing the chart.

  Width (numeric) : Width of box containing the chart.

  Data (array) : An array of data values to be charted, one bar will be drawn for each element of the array, automatic scaling will be calculated according to lowest/highest value present in the array.

  Data titles (array) : An array containing titles for each element in the data array, those titles will be displayed under each bar, so the quantity of characters in each title should not be long, so as to fit under the bar.

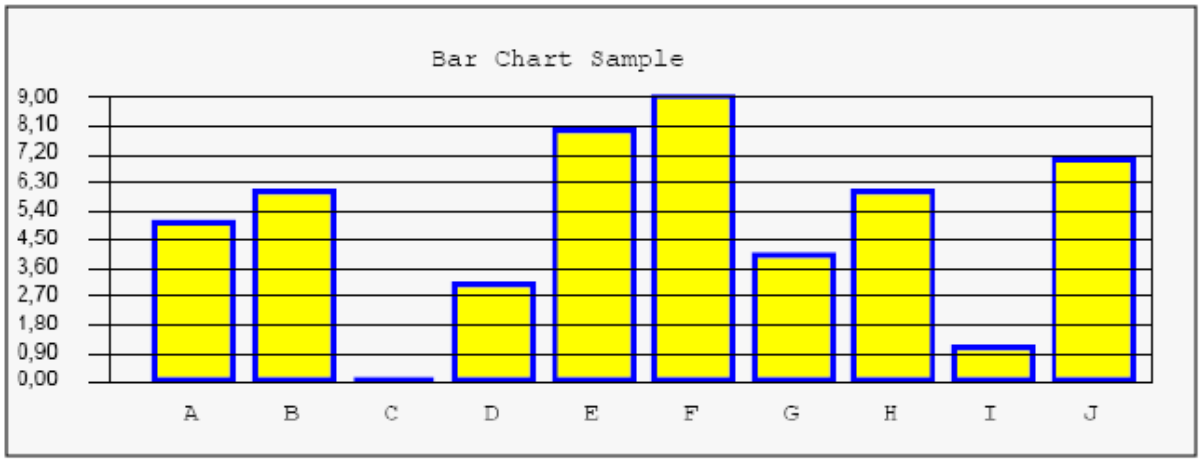
  Chart title (character) : Title of the chart, will be displayed on the center of the chart top

row.

Color1 (numeric) : Color for the outline of the drawn bars. Conforms to color definition in GRA.CH or any valid RGB color.

Color2 (numeric) : Color for the filling of the drawn bars. Conforms to color definition in GRA.CH or any valid RGB color.

Example : oPdf:BarChart(65, 10, 14, 55, data, titu, "Bar Chart Sample", GRA_CLR_BLUE, GRA_CLR_YELLOW)
this will draw a bar chart with blue outlined bars and yellow fillings. See **Demo.PDF** for more details.



3.1.4.2 LineChart

- **LineChart(** *init line, init col, height, width, data, data titles, chart title, colors* **)** => draws a line chart, at position *init line, init col* with size *height, width*, using the values in *data* array, for each position, a title in *data titles* is displayed. Multiple lines can be drawn, each one will have its color from the *colors* array. A title can be displayed at the top/center of the chart using the *chart title* parameter. See a sample at Charts.PDF

  Init line (numeric) : Vertical position of chart corner according to coordinate system.

  Init col (numeric) : Horizontal position of corner to start drawing the chart.

  Height (numeric) : Height of box containing the chart.

  Width (numeric) : Width of box containing the chart.

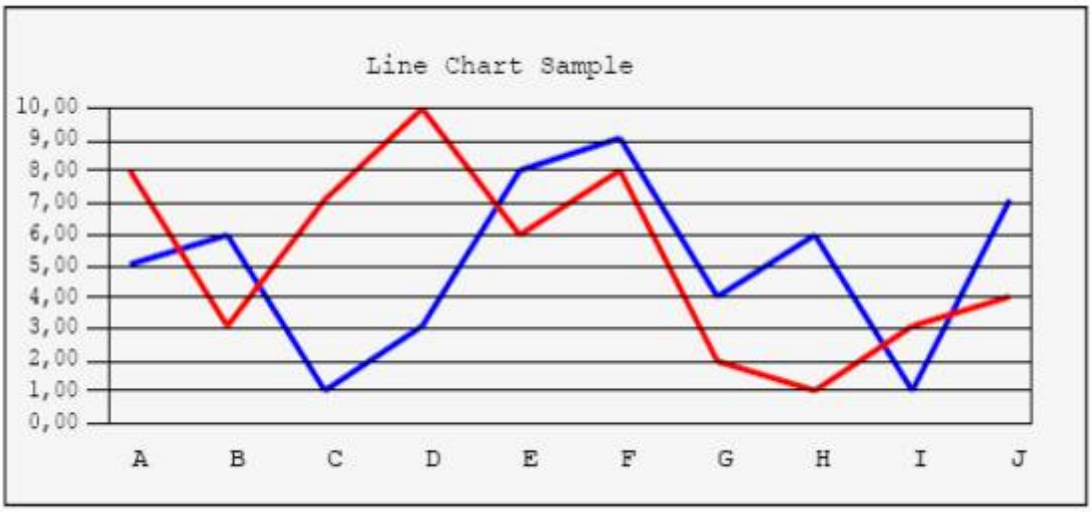
  Data (array) : An array of arrays. Each sub array contains data values to be charted, one line will be drawn for each element of the sub array, using different colors for each line, automatic scaling will be calculated according to lowest/highest value present in all the array.

  Data titles (array) : An array containing titles for each sub array in the data array, those titles will be displayed at the bottom of the chart.

  Chart title (character) : Title of the chart, will be displayed on the center of the chart top row.

  Colors (array) : An array of colors for the lines of the chart, should have as many elements as sub arrays in the data array. Conforms to color definition in GRA.CH or any valid RGB color.

  Example : oPdf:LineChart(44, 10, 14, 50, data, titu, "Line Chart Sample", { GRA_CLR_BLUE, GRA_CLR_RED })
  this will draw a line chart with blue and red lines. See **Demo.PDF** for more details.


Line Chart Sample
10,00
9,00
8,00
7,00
6,00
5,00
4,00
3,00
2,00
1,00
0,00
A
B
C
D
E
F
G
H
I
J

3.1.4.3 PieChart

- **PieChart(** *init line, init col, radius, data, data titles, chart title* **)** => draws a pie chart, at position *init line, init col* with size *radius*, using the values in *data* array, for each position, a title in *data titles* is displayed. Each pie "slice" will have its color. A title can be displayed at the top/ center of the chart using the *chart* title parameter. See a sample at Charts.PDF

  Init line (numeric) : Vertical position of chart corner according to coordinate system.

  Init col (numeric) : Horizontal position of corner to start drawing the chart.

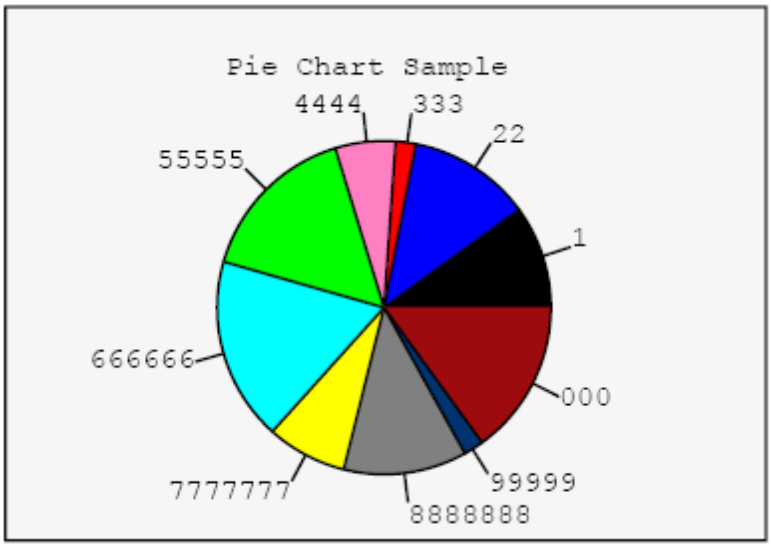
  Radius (numeric) : Height of box containing the chart.

  Data (array) : An array of data values to be charted, one pie slice will be drawn for each element of the array, using different colors for each slice.

  Data titles (array) : An array containing titles for each element in the data array, those titles will be displayed next to each slice in the chart.

  Chart title (character) : Title of the chart, will be displayed on the center of the chart top row.

  Example : oPdf:PieChart(15, 30, 5, data, titu, "Pie Chart Sample")
  This will draw a pie chart. **See Demo.PDF** for more details.



3.1.5 Xbase Graxx Methods

XbasePDF class methods that emulate Xbase++ functions.

3.1.5.1 GraSetAttrArea

- **GraSetAttrArea(** *aAttributes* **)** => Sets attributes for area filling. The array is the same as in Xbase++, and the elements considered are :

Array element	Description
GRA_AA_COLOR	Foreground color
GRA_AA_BACKCOLOR	Background color
GRA_AA_BGMIXMODE	Color mix attribute for background
GRA_AA_SYMBOL	Fill pattern

3.1.5.2 GraSetAttrLine

- **GraSetAttrLine(** *aAttributes* **)** => Sets attributes for drawing lines. The array is the same as in Xbase++, and the elements considered are :

Array element	Description
GRA_AL_COLOR	Foreground color
GRA_AL_WIDTH	Line width
GRA_AL_TYPE	Line type

3.1.5.3 GraSetAttrMarker

- **GraSetAttrMarker(** *aAttributes* **)** => Sets attributes for drawing markers. The array is the same as in Xbase++, and the elements considered are :

Array element	Description
GRA_AM_COLOR	Foreground color
GRA_AM_SYMBOL	Symbol for marker
GRA_AM_BOX	Dimension of a marker in x and y direction

3.1.5.4 GraSetAttrString

- **GraSetAttrString(** *aAttributes* **)** => Sets attributes for drawing strings with GraStringAt(). The array is the same as in Xbase++, and the elements considered are :

Array element	Description
GRA_AS_COLOR	Foreground color
GRA_AS_ANGLE	Angle at which characters are output

3.1.5.5 GraSetColor

- **GraSetColor(** *nForegroundColor, nBackgroundColor* **)** => Sets colors for Foreground and Background drawing.

3.1.5.6 GraSetFont

- **GraSetFont(** *oFont/cFont* **)** => Sets the font for subsequent writing with GraStringAt(). The font can be specified as a XbpFont object, or the font compound name.

3.1.5.7 GraStringAt

- **GraStringAt(** *aPoint, cText* **)** => Writes the cText string on the aPoint in the page. aPoint is an array of x,y coordinates, according to the chosen coordinate system (see the NewPage() Method)

3.1.5.8 GraPos

- **GraPos(** *aPoint* **)** => Moves the pen to aPoint in the page. aPoint is an array of x,y coordinates, according to the chosen coordinate system (see the NewPage() Method)

3.1.5.9 GraQueryTextBox

- **GraQueryTextBox(** *cText* **)** => The function GraQueryTextBox() returns a two dimensional array of five elements. Each element contains a subarray with two elements which are the relative coordinates for five points. The first four elements designate the corner points for a parallelogram which represents the boundaries of the specified character string <cString> . The fifth element is the pen position at which the pen is found after the character string is drawn with GraStringAt():

```
aPoints := { { nXLeft , nYTop    }, ;          // upper left corner
             { nXLeft , nYBottom }, ;          // lower left corner
             { nXRight, nYTop    }, ;          // upper right corner
             { nXRight, nYBottom }, ;          // lower right corner
             { nXPen  , nYPen    } }           // pen position
```

The points designate the boundaries if the string is displayed at the origin of the coordinate system (the point {0,0}). To calculate the actual coordinates for a box surrounding a string, add the x and y positions at which <cString> is drawn.

3.1.5.10 GraArc

- **GraArc(** *[<aCenter>], <nRadius>, [<aEllipse>], [<nStartAngle>], [<nSweepAngle>]* **)** => Draws an Arc.Circles, arcs and ellipses are drawn using the function GraArc(). By default a circle is drawn as an outline with the color and the line width set by the function GraSetAttrLine(). The fill pattern and fill color for a filled circle can be specified with GraSetAttrArea(). The default for the fill pattern is the #define constant GRA_SYM_SOLID.

  Parameters are :

  <aCenter>:= { <nX0>, <nY0> }

  <aCenter> is an array of two elements which determines the coordinates for the center point of the circle. The first element <nX0> specifies the x coordinate and the second element <nY0> specifies the y coordinate. The unit for the coordinates depends on the coordinate system defined for the presentation space. If no presentation space is specified, values for the coordinates are given in pixels, which is the unit for a window. The origin for the coordinates (the point {0,0}) is at the lower left. If <aCenter> is not specified, the center point of the circle is the current pen position.

  <nRadius>

  <nRadius> is a positive integer which indicates the radius of the circle.

  <aEllipse> := { <nX1>, <nY1>, <nX2>, <nY2> }

  The parameter <aEllipse> is needed to draw an ellipse. It is an array with four elements which designates the end points of both ellipse axes. The coordinates of both endpoints are calculated by the following equations (see sections "Description" and "Example"):

```
nXhorizontal := nX0 + nRadius * nX1  // ellipse axis in the
nYhorizontal := nY0 + nRadius * nY1  // x direction (horizontal)
nXvertical   := nX0 + nRadius * nX2  // ellipse axis in the
nYvertical   := nY0 + nRadius * nY2  // y direction (vertical)
```

The default value of <aEllipse> is {1,0,0,1}.

<nStartAngle>

<nStartAngle> determines the starting point from which an arc is drawn. It is a positive integer which indicates the number of degrees the starting point of the arc deviates in the counter clockwise direction from horizontal.

<nSweepAngle>

<nSweepAngle> is a positive integer which indicates the opening angle for the arc in degrees.

3.1.5.11 GraBox

- **GraBox(** *[<aLeftBottom>], <aRightTop>, [<nFill>], [<nHRadius>], [<nVRadius>]* **)** => Draws a rectangle.Rectangles are drawn with the function GraBox(). By default, output is an outline, with the color and line width set by the function GraSetAttrLine(). For a filled rectangle, the fill pattern and the fill color are determined by GraSetAttrArea(). The default pattern corresponds to the #define constant GRA_SYM_SOLID.

Parameters are :

<aLeftBottom> := { <nX1>, <nY1> }

<aLeftBottom> is an array of two elements which determines the coordinates for the lower left corner of the rectangle. The first element <nX1> designates the x coordinate and the second element <nY1> the y coordinate. The unit for the coordinates depends on the coordinate system defined for the presentation space. If no presentation space is specified, the values for the coordinates are given in pixels, which is the unit for a window. The origin for the coordinates (point {0,0}) is at the lower left. If <aLeftBottom> is not specified, the lower left corner of the rectangle is the current pen position.

<aRightTop> := { <nX2>, <nY2> }

<aRightTop> is an array of two elements which contains the coordinates for the upper right corner of the rectangle.

<nFill>

The optional argument <nFill> specifies whether the rectangle is drawn as an outline or is filled. The following table shows the #define constants from the GRA.CH file to use for

<nFill> :

Fill attributes for GraBox()

Constant	Description
GRA_FILL	Fills rectangle
GRA_OUTLINE *)	Only draws rectangle border
GRA_OUTLINEFILL	Draws rectangle border and fills

*) Default value

<nHRadius>

The parameters <nHRadius> and <nVRadius> determine how much to round the corners of the rectangle. Both are positive integers which, taken together, determine the distance (radius of curvature) from a corner to the middle of the rectangle. This distance provides the measure from which the corners are rounded. <nHRadius> indicates the horizontal distance. When <nHRadius> is equal to <nVRadius> , the corners are rounded by a 90ø arc.

<nVRadius>

<nVRadius> determines the vertical curvature radius for rounding the corners. For rounded corners, both parameters <nHRadius> and <nVRadius> must be greater than 0.

3.1.5.12 GraLine

- **GraLine(** *[<aStartPoint>], <aEndPoint>* **)** => The function GraLine() is a graphic primitive which draws a line in a presentation space. In order to draw a line, two points must be specified. If the first point <aStartPoint> is not specified, the line begins at the current pen position. Various line types are available (solid, dashed, dotted, etc.). Line types must be selected before output of the line by calling the function GraSetAttrLine(). After GraLine() returns, the pen is positioned at <aEndPoint> .

  Parameters Are

  <aStartPoint> := { <nXStart>, <nYStart> }

  <aStartPoint> is an array of two elements which determines the coordinates for the origin of the line. The first element <nXStart> specifies the x coordinate and the second element <nYStart> specifies the y coordinate. The unit for the coordinates depends on the coordinate system defined for the presentation space. If no presentation space is defined, the values for the coordinates are given in pixels, which is the unit for a window. The origin for the coordinates (the point {0,0}) is at the lower left. If <aStartPoint> is not specified, the line begins at the current pen position.

  <aEndPoint> := { <nXEnd>, <nYEnd> }

  <aEndPoint> is an array of two elements which contains the coordinates for the end point of the line.

3.1.5.13 GraMarker

- **GraMarker(** *[<aPoint>]* **)** => The function GraMarker() marks a single point at the position <aPoint> . Various marker types are available (cross, circle, square etc.). They must be selected before the output of a marker by calling the function GraSetAttrMarker(). The default marker is the #define constant GRA_MARKSYM_CROSS. The function GraMarker() does not change the pen position.

  Parameters are :

  <aPoint>:= { <nX>, <nY> }

  <aPoint> is an array of two elements which determines the coordinates for the point at which the marker is drawn. The first element <nX> specifies the x coordinate and the second element <nY> specifies the y coordinate. The unit for the coordinates depends on the coordinate system defined for the presentation space. If no presentation space is specified, the values for the coordinates are specified in pixels, which is the unit for a window. The origin for the coordinates (the point {0,0}) is the lower left. If <aPoint> is not specified, the marker is drawn at the current pen position.

3.1.5.14 GraPathBegin

- **GraPathBegin( )** => The function GraPathBegin() initializes a graphic path. A path is formed with graphic primitives like GraLine(), GraArc(), GraSpline() or GraStringAt() and must be closed with the function GraPathEnd(). The graphic primitives executed between GraPathBegin() and GraPathEnd() define the border for a path which can be formed in a variety of ways. During the path definition no output occurs. The defined path is made visible using the functions GraPathFill() and GraPathOutline() after calling the function GraPathEnd().

  A graphic path is always used when complex structured areas are outlined or filled. An example of such a complex area is a character string whose letters are filled with a pattern. A graphic path can consist of any number of enclosed areas (each letter in a string is an area, all the letters form the graphic path which is filled or outlined). If characters are used in the definition of a graphics path, they must be provided by a vector font because bitmap fonts do not use graphic primitives to display characters.

  The functions GraPathBegin() and GraPathEnd() cannot be nested. After a call to GraPathBegin(), a call to GraPathEnd() must occur so that graphic primitives can again be used for normal drawing rather than being interpreted as part of a path definition. The graphic path is made visible using a path operation such as GraPathFill() or GraPathOutline(). For each presentation space, only a single graphic path is supported, which is discarded as soon as a path operation like "Fill" or "Outline" is executed.

3.1.5.15 GraPathEnd

- **GraPathEnd(** *[<lCloseFigure>]* **)** => The function GraPathEnd() ends the definition of a graphic path. It must always be executed after a call of GraPathBegin() (see GraPathBegin()) and before the execution of a path operation. Path operations are executed by the functions GraPathFill(), GraPathClip() and GraPathOutline(). After a path operation occurs, the path definition is discarded.

  Parameters are :

  <lCloseFigure>

  <lCloseFigure> is a logical value specifying whether the drawing defined as a graphical path is to be closed. The default value is .F. (false), meaning the path includes only the defined graphic primitives. If <lCloseFigure> equals .T. (true), the starting point of the first graphic primitive is connected to the end point of the last graphic primitive.

3.1.5.16 GraPathOutline

- **GraOutline( )** => The function GraPathOutline() outlines a graphic path defined by GraPathBegin() / GraPathEnd(). After the call to GraPathOutline(), the definition of the path is discarded.

3.1.5.17 GraPathFill

- **GraPathFill(** *[<nFillMode>]* **)** => The function GraPathFill() fills a graphic path defined by GraPathBegin() / GraPathEnd() with a pattern. The fill pattern must have been previously selected using the function GraSetAttrArea(). After the call to GraPathFill() the definition of the path is discarded.

  Parameters are :

  <nFillMode>

  <nFillMode> specifies the mode for filling. This is significant only with complex paths. Complex paths consist of several, mutual overlapping areas. For this parameter, one of the two #define constants GRA_PATH_FILL_ALTERNATE or GRA_PATH_FILL_WINDING are used. The default value is GRA_PATH_FILL_ALTERNATE.

3.1.5.18 TextBox

- **TextBox(** *[<aStart>], [<aEnd>], <cText>, [<lBox>], [nLType]* **)** => The method TextBox() writes a string to the PDF and then draws a box around it.

Parameters are :

<aStart>
<aStart> specifies the starting position for the text, it will also be the lower left corner of the box.

<aEnd>
<aEnd> specifies the upper right corner for the box

<cText>
<cText> is the string of characters to be placed in the PDF.

<lBox>
<lBox> is a logic value (True or False) to draw or not the box around the text.

<nLType>
<nLType> is the line type to be used, valid line types are GRA_AL values from GraSetAttrLine

---

3.1.5.19 XppPDF.ch

- XppPDF.ch is a preprocessor file that will (almost) automatically convert all your existing reports to generate PDF files.

  Including this preprocessor file at compile time, all your Gra... calls will be translated to an internal XbpPDF routine that will test for the type of object being passed as the first parameter. If this objec is a PDF object, than a PDF file will be created, if it is a Xbase++ object like a XbpPresSpace or a XbpPrinter, then normal processing will continue. With this approach minimal changes have to be made to your existing reports in order to use the XbasePDF class.

An example of how to use this new feature :

```
#include "xpppdf.ch"

PROCEDURE MAIN

        aSize := AppDeskTop():CurrentSize()
        aPos  := AppDeskTop():Currentpos()
        aSize[2] := aSize[2] - 45
        oDlg := xbpDialog():new(,,aPos,aSize)
        oDlg:Create()

        oPs := oDlg:DrawingArea:LockPS()
        oFont := xbpFont():New( oPs )
        oFont:Create('12.Times New Roman')
```

```
        oPdf := xbppdf():New()
        oPdf:Create("gra.pdf")
        oPdf:NewPage( , , , , ,1 )

        Report( oPdf )
        opdf:enddoc(.t.)

        Report( oPs )

        nEvent := 0
        do while nEvent <> xbeP_Close
                nEvent := AppEvent( @mp1, @mp2, @obr )
                obr:handleEvent( nEvent, mp1, mp2 )
        Enddo

Return

FUNCTION Report( ops )

        aAttra := array(GRA_AA_COUNT)

        aAttra[GRA_AA_COLOR] := GRA_CLR_WHITE
        GraSetAttrArea( ops, aAttra )
        GraBox( ops, {0,0}, aSize, GRA_OUTLINEFILL)

        aAttra[GRA_AA_COLOR] := GRA_CLR_YELLOW
        GraSetAttrArea( ops, aAttra )
        GraBox( ops, {80,100}, {220,300}, GRA_OUTLINEFILL)

        GraSetFont( ops, oFont )
        GraSetColor( ops, GRA_CLR_BLUE )
        GraStringAt( ops, {100,200}, "Hello world!" )

Return(.t.)
```

As you can see creating two different objects,a XbpPDF object (oPdf) and a PresSpace (oPs) you can call the same function (Report) and the preprocessor together with the XbpPDF class, will do the rest. The exact same report that is seen on the screen, is saved as a PDF file, and programming efforts is minimal.

3.1.6 Class Variables

oDoc *(object)* main pdf object.

oPage *(object)* current page object.

cName *(character)* file name for the pdf.

aTam *(array)*	two elements array, for page width and height in 1/72 inches.
nCoordSystem *(numeric)*	coordinate system in use.
cFontn *(character)*	current font name.
nFonts *(numeric)*	current font size.
aFonts *(array)*	used fonts in this document.
aAvFonts *(array)*	available true type fonts and their file names.
oOutline *(object)*	root parent for outline.
nAlt *(numeric)*	height of line.
nLrg *(numeric)*	character width.
nSlin *(numeric)*	last line number, current vertical position.
nScol *(numeric)*	last column number, current horizontal position.
nPag *(numeric)*	page number.
nLmg *(numeric)*	left margin.
nTmg *(numeric)*	top margin.
lUscore *(logic)*	underscore text.
nFcor *(numeric)*	filling color for boxes and shapes.
nScor *(numeric)*	stroke color for lines, boxes and shapes.
nTcor *(numeric)*	text color.
lFixed *(logical)*	fixed or proportional character spacing.
lDebug *(logical)*	creates a PDF.LOG file with all calls to the class
lView *(logical)*	uses the View() method to open the PDF (:EndDoc)

3.2 View & Print

Class Methods used to View and/or Print existing PDF files.

3.2.1 Intro

This class was created to fill a need to Xbase++ users around the world, as noted by the many threads existing in the Xbase News Group about viewing and printing PDF files from Xbase++ applications.

This is a very simple class with very little effort in terms of programming, of course implementing it will require code changes, but the benefits are worth it.

An example of a simple program would be :

```
PROCEDURE MAIN

   pdf := XbpPDF():New()                  // Initializes the PDF object

   pdf:Create( )                          // Creates the ViP object

   pdf:View( "Demo.PDF" )                 // Views the loaded PDF on screen

   pdf:Print( "Demo.PDF" )                // Prints the file to any windows printer

   pdf:Destroy()                          // Terminates and release resources

return
```

3.2.2 View

- **View(** *cFileName, [nZoom], [lMaximize], [oParent], [oOwner], [aPos], [aSize], [lModal], [lNoPrint], [nIPag]* **)** => Show's the previous loaded file on the screen.

  cFileName (character) : full filename of PDF file (including path, if not in current folder)

  nZoom (numerical) : the initial zoom setting for viewing the pdf, any number higher than 0, 100 is 100 %, default is 100%.

lMaximize (logical) : show in full screen mode (default), or normal window

oParent (object) : is the parent dialog for the preview window, default is AppDesktop

oOwner (object) : is the owner dialog for the preview window, default is AppWindow

aPos (array) : position for the preview window, default is owner center

aSize (array) : Size of preview window, default is owner size

lModal (logic) : Opens the preview window in Modal mode, in this case oParent must be AppDesktop

lNoPrint (logic) : Don't display the "Print" button in the preview window

nIPag (numeric) : Initial page number to display, defaults to 1.

View window navigation :

-  or [-] : Zoom out
-  or [+] : Zoom in
-  or [H] : Zoom to Height
-  or [W] : Zoom to Width
-  or [F] : Zoom to Fit
-  or [1] : First Page
-  or [<] : Previous Page
-  or [>] : Next Page
-  or [999] : Last Page
-  : Displays 1 page per screen
-  : Displays 2 pages per screen, divided vertically (side by side)

-  : Displays 2 pages per screen, divided horizontally (top and bottom)
-  or [P] : Print
-  or [ESC] : Exit View window

3.2.3 Print

- **Print(** *cFileName, [nFirst] [,nLast], [nFit]* **)** => Print's the named file to any Windows Printer device.

cFileName (character) : full filename of PDF file (including path, if not in current folder) to be printed.

nFirst (numeric) : number of first page to be printed, default is 1.

nLast (numeric) : number of last page to be printed, default is number of pages in pdf file.

nFit (numeric) : page scaling, options are :
0 = no scaling
1 = Fit to Page
2 = Shrink larger pages
(Default is 1 = Fit to Page)

Obs: The PDF will be printed to the printer object specified in the :Create() method. If none, then a Printer Dialog will be shown to selecet the desired printer. All printer specifications like page size, orientation, number of copies, paper bins, collate, duplex etc will be obtained from that object.

3.2.4 RenderToPS

- **RenderToPS(** *cFileName, oPspace ,nRes, nPag* **)** => Render's the named file to any Xbase Presentation Space.This method can be used to create your own viewer, to print PDF files, or any other desired usage.

cFileName (character) : full filename of PDF file (including path, if not in current folder) to be rendered.

oPspace (object) : Xbase++ presentation space object, it can be linked to a screen dialog or to a printer object

nRes (numeric) : Resolution (DPI) for the rendering, default is 150. Any resolution above 300 DPI can cause an excessive amount of memory usage.

nPag (numeric) : Page number to render, any number between 1 and the total number of pages in the PDF file. Default is 1.

3.2.5 RenderToBMP

- **RenderToBMP(** *cFileName, oBmp ,nRes, nPag* **)** => Render's the named file to any Xbase Bitmap object.This method can be used to create your own viewer, to print PDF files, or even to change the contents of an existing PDF, adding or erasing elements, saving the rendered page as an image (BMP, JPEG, etc) files or any other desired usage.

cFileName (character) : full filename of PDF file (including path, if not in current folder) to be rendered.

oBmp (object) : Xbase++ bitmap object.

nRes (numeric) : Resolution (DPI) for the rendering, default is 150. Any resolution above 300 DPI can cause an excessive amount of memory usage.

nPag (numeric) : Page number to render, any number between 1 and the total number of pages in the PDF file. Default is 1.

3.2.6 Class Variables

lPrinted *(logical)*	True or False if the file was printed, either by the :Print method, or by the Print button in the View screen.
cZoom *(character)*	zoom constant name. Default is "Zoom"
cPage *(character)*	page constant name. Default is "Page"
cTbar *(character)*	configures Tool Bar position, can be either "Top" or "Left". Default is "Top"
cTitle *(character)*	contains the title for the View window.
nQpag *(numeric)*	contains the number of pages in the PDF.
lTip *(logical)*	True or False controls the display of ToolTips in the View window

---
