pro-sys consultants

PDFinclude PRO

Version 1

Documentation

Prepared by:
PRO-SYS Consultants Ltd.
Gordon Campbell
July 6, 2005

PRO-SYS Document – PDFinclude PRO Documentation

Any questions regarding this document should be forwarded to:

PRO-SYS Consultants Ltd.
7220 Montana Road
Richmond, BC, CANADA
V7C 4K3

Phone: 1-604-377-5050 or 1-403-606-0799

E-mail: sleung@epro-sys.com or gcampbell@epro-sys.com

Document Change Log

As documents are revised, important information is added or removed. Enter changes in this table so readers know what has changed.

Refer to this table to make sure you haven't missed the latest information.

Date yy/mm/dd	**Changed By**	**Description of Change**	**Reason for Change**
05/07/06	G Campbell	Initial Release	Initial Release

This is the document change log for Project: PDFinclude PRO Documentation.

Table of Contents

1.Introduction

PDFinclude PRO is a Progress ® include file developed by PRO-SYS Consultants Ltd (PRO-SYS) to allow Progress developers to output reports in Adobe PDF file format without having to use third-party tools or utilities.

PDFinclude PRO is a stand-alone component that defines a toolset of Progress functions and procedures that aid in the output of a PDF file directly from within the Progress 4GL.

PDF files can be read by PDF viewers such as Adobe Reader and PDFView.

The PDF file format is fast becoming an industry-standard for electronic document viewing. The PDF format allows for documents that can be accessed by a broad range of hardware and software devices (including PDAs and Internet browsers).

PDFinclude PRO utilizes Progress code that is compliant with versions 9 and 10 of the Progress 4GL.

Some of the benefits of using PDFinclude PRO are:

Consistency	By using the PDF file format, you can present a representation of the output file in a consistent manner. Use the same PDF file across multiple devices, platforms and/or software packages.
Easy Implementation	Since PDFinclude PRO is written in Progress 4GL the ability to implement into your existing (or new) code is easy and seamless. No external calls are required to third-party products (such as a Perl script). PDFinclude PRO can also be used with Progress versions 9 and 10 on UNIX and/or Windows platforms.
Secure Report Output	The PDF file format is a secure format that cannot be easily changed therefore sensitive reports can be distributed to users without the worry of them changing reported material.
Configurable	By offering a multitude of different Progress-written PDF related functions, you can easily configure and determine the outputs content, object placement and look-and-feel.
Report Viewer Supplied	Since PDF is becoming an industry-standard file format, many document viewers already exist (such as Adobe's Acrobat Reader). This reduces your development effort by not having to create a separate Report Viewer.

2.PDFinclude PRO

2.1Overview

As previously mentioned, PDFinclude PRO is written in Progress and consists of a set of functions. These functions are used in combination to create an output file in PDF format. The list of procedures and functions and their uses can be viewed in Section 3 and Section 4.

Due to the complexity of the PDF specification and our requirement of having PDF easy to implement, building a PDF document with PDFinclude PRO involves two different planes … or Spaces as they are called within the PDF specification. The two spaces are the Text Space and the Graphic Space. Each of the spaces are defined further in preceding sections but the following items outline the basic space functions.

Text Space: The Text Space allows you to manipulate and control the where and how text will appear on the document. The Text Space starts on the top-left hand corner of the page and moves across and downward.

In fact, this space doesn't actually exist in the PDF specification. It was created by PRO-SYS to mimic how reports are currently developed using the Progress 4GL.

Procedures such as pdf_text_to, pdf_text_at use the Text Space. These procedures use a mimicked Row and Column coordinate system. And these are really only useful if you use a Fixed width font (such as Courier).

Graphic Space: The Graphic Space allows you to manipulate and control the drawing of non-textual object such as JPEG Images, Rectangles, Lines, etc. The Graphic Space start at the bottom-left corner of the page and move up and across.

Text placement can also be used performed on the Graphic Space by using procedures such as pdf_text_xy, pdf_text_align. These procedures allow you to specifically use X and Y coordinates and are very useful when dealing with proportional fonts (such as Arial).

By manipulating and controlling the Spaces, you can create elegant documents that are directly derived from within your Progress application.

Before using of the following procedures or functions, you must add "{ pdf_inc.i }" to your Progress program.

The pdf_inc.i include file also accepts an argument. The argument allows you to specify whether you want to add the pdf_inc.p as a SUPER procedure or not. The following table outlines the possible options.

Argument	**Sample**	**Description**
blank	{ pdf_inc.i }	This will run pdf_inc.p as a SUPER procedure and associate the procedure handle with the SESSION handle.

		This is the default action.
THIS-PROCEDURE	{ pdf_inc.i “THIS-PROCEDURE” }	This will run pdf_inc.p as a SUPER procedure and associate the procedure handle with the THIS-PROCEDURE handle.
Anything Else	{ pdf_inc.i “NOT SUPER” }	This will run pdf_inc.p as a PERSISTENT procedure and will not create the procedure as SUPER. If this option is used then the procedure calls must have the following added to each call – IN h_PDFinc. For example, you originally had: *RUN pdf_new (“Spdf”).* now with this option you must change your call to: *RUN pdf_new IN h_PDFinc (“Spdf”).*

3.Procedure List

This section defines the procedures available within PDFinclude PRO. For each procedure you are given the required parameters, expected values, sample calls, and a detailed description of how (and when) the procedure should be used.

pdf_bookmark

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	Title as Character	- Bookmark Description
	Parent as Integer	- 0 or valid Bookmark number
	Expand as Logical	- Expand this bookmark (if parent) – Yes/No
	OUTPUT:	
	Bookmark as Integer	- How thick to draw the circle border (stroke)
Sample Call:	RUN pdf_bookmark (“Spdf”,”Customer 0001”,0,No,OUPUT vBookmark)	

Description:

This procedure allows you to add a bookmark to the PDF document. This will in turn add a marker that will appear in the Bookmarks section when viewing the PDF document (via Reader).

This allows for easier navigation of the document when viewing.

pdf_circle

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	X Coord as Integer	- Graphic X Coordinate
	Y Coord as Integer	- Graphic Y Coordinate
	Radius as Integer	- Radius of Circle
	Weight as Decimal	- How thick to draw the circle border (stroke)
Sample Call:	RUN pdf_circle (“Spdf”,10,10,100,0.5)	

Description:

This procedure will draw a circle at the specific X/Y coordinate. The X/Y coordinate represent the center point of the circle and also become the new Graphic X/Y points after drawing the circle.

This procedure uses the colours set by the pdf_stroke_color and pdf_stroke_fill.

pdf_close

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
Sample Call:	RUN pdf_close (“Spdf”)	

Description:

This procedure closes the identified PDF stream and generates the PDF document that was identified in the pdf_new procedure.

pdf_close_path

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
Sample Call:	RUN pdf_close_path ("Spdf")	

Description:

This procedure output a "B" element. This will basically stroke and fill anything that has been drawn on the graphic plane. Not usually required but in certain circumstances may be needed.

pdf_curve

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	X1 Coord as Integer	- Graphic X1 Coordinate
	Y1 Coord as Integer	- Graphic Y1 Coordinate
	X2 Coord as Integer	- Graphic X2 Coordinate
	Y2 Coord as Integer	- Graphic Y2 Coordinate
	X3 Coord as Integer	- Graphic X3 Coordinate
	Y3 Coord as Integer	- Graphic Y3 Coordinate
	Weight as Decimal	- How thick to draw the circle border (stroke)
Sample Call:	RUN pdf_curve ("Spdf",10,10,100,200,200,100,0.5)	

Description:

This procedure add a Bézier curve is added from the current Graphic X/Y Location to X3/Y3 using X1/Y1 and X2/Y2 as the control points. The X3/Y3 of the curve becomes the new Graphic X/Y Location.

This procedure uses the colours set by the pdf_stroke_color and pdf_stroke_fill.

pdf_Encrypt

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Master Password as Character	- The PDF document's Master Password
	User Password as Character	- The PDF document's User Password
	Access List as Character	- List identifying Access Rights
	Encryption Key as Integer	- 40 or 128 Bit Encryption Key Length
	Encryption Mode as Character	- COM, SHARED or OS
	Silent Encryption as Logical	- TRUE OR FALSE
Sample Call:	RUN pdf_Encrypt ("Spdf","gord1","gord","",128,"COM",TRUE)	

Description:	

This procedure allows you to encrypt a generated PDF document. The call to this procedure can occur anywhere between the appropriate pdf_new and pdf_close procedures.

This procedure call integrates with PDFpsp.w (another external procedure). PDFpsp.w uses a commercial product called Pretty Safe PDF (PSP) from PDFlib GmbH. The PSP product can be downloaded from www.pdflib.com. PRO-SYS has is a reseller of all PDFlib GmbH products and we offer PEG members a discount. For more information go to www.epro-sys.com. The discounts will only be applied by PRO-SYS. PDFlib GmbH will not honour the discounts, so orders must be directed to PRO-SYS Consultants Ltd.

The following table outlines the parameters in further detail:

Parameter	Description
Stream Name	Must be a valid Stream Name as created in the procedure – see calls to ‘pdf_new’ to identify valid stream names.
Master Password	The Master Password must be a maximum of 32 characters. A ‘blank’ password is valid. The Master Password allows you to associate a password to protect the documents security settings. This password must be entered to access the Document Security. This password also allows entry to the documents contents.
User Password	The User Password must be a maximum of 32 characters. A ‘blank’ password is valid. The User Password allows you to associate a password to protect the documents contents. This password must be entered to access the Document.
Access List	The Access list can be ‘blank’ or a comma-delimited list of access prevention rights. If ‘blank’, the document has full access rights (default). If ‘non-blank’, then only specific items will be allowed when the document is viewed. The following table outlines the valid entries for the comma-delimited list:

Code	Description
noprint	Prevent Printing of the File
nohiresprint	Prevent High Resolution Printing
nomodify	Prevent Adding Form Fields & Other Changes
nocopy	Prevent Copying and Extracting Text or Graphics
noannots	Prevent Adding or Changing Comments or Form Fields
noforms	Prevent Filling of Form Fields
noaccessible	Prevent Extraction of Text or Graphics
noassemble	Prevent Inserting/Deleting/Rotating Page, Bookmarks

Parameter	Description
Encryption Key Length	This value can either be 40 or 128. This value represents the length of the Encryption Key (in bits). 128 Bit encryption is currently the strongest encryption available.
Encryption Mode	PSP can be run via a few different options. The options available vary by Operating System. The following table outlines the values and the Operating System that they can be used with.

Mode	OS	Description
COM	Windows	Uses the COM version of PSP. Accessed via a COM-Handle.
SHARED	Windows, *NIX	Uses the DLL (for Windows) or Shared Library (for *NIX). Accessed via Functions calls defined in PSP.i and PSPbind.p.

pdf_fill_text

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Form FieldName as Character	- Name of a valid Font
	Field Text as Character	- Integer value of the Character to change
	Field Options as Character	- Postscript Character name
Sample Call:	RUN pdf_fill_text (“Spdf”, “InvoiceNumber”,”00001”,””)	

Description:

This procedure is used in conjunction with the pdf_open_pdf and pdf_use_pdf_page procedures.

If you’ve opened and existing PDF document and then used a page within your new document then you may be able to publish data to any Adobe Form Fields that appear within that used page. If no Form Fields are available, or you publish a field name that doesn’t exist, then no data will be added to your form.

The last parameter “options”, allows you to perform some additional processing if the form field is found. The two options currently available are:

Align=LEFT|CENTER|RIGHT
Multiline

The ‘Align’ option allows you to programmatically set how you want the field contents aligned when the field value is displayed.

The ‘MultiLine’ option allows you to tell the program to fit as many lines of text into the specified region of the Form Field as possible.

For more information on this functionality, see the section called ‘Introducing Complex Forms’.

pdf_Font_Diff

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Name as Character	- Name of a valid Font
	Character Number as Integer	- Integer value of the Character to change
	Postscript Name as Integer	- Postscript Character name
Sample Call:	RUN pdf_font_diff (“Spdf”, 190,”/nsuperior”)	

Description:

This procedure allows you to dynamically change the character mapping for a given character in a specified Font Name. This is useful when you have previously created text files that include high ASCII character values that are used for graphic line drawing. For example, you use ASCII character 196 to ‘draw’ a horizontal line. When you look at the actual text file you won’t see a horizontal line you will see a funny looking binary character. Using this procedure, you can redefine how PDFinclude handles that character … you can have it remapped to a horizontal line (Postscript name is SF100000).

pdf_GetBestFont

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Name as Character	- Name of a valid Font
	I/O Text as Character	- Integer value of the Character to change
	I/O FontSize as Decimal change	- Integer value of the Character to
	Smallest Size as Decimal	- Postscript Character name
	Chop Text Name as Logical	- Postscript Character name
	From X as Decimal	- Postscript Character name
	To Y as Decimal	- Postscript Character name
Sample Call:	RUN pdf_GetBestFont (“Spdf”, “Arial”, INPUT-OUTPUT cText, INPUT-OUTPUT dFontSize, 1.0, No, 100.0, 200.0)	

Description:

This procedure calculates the best font size to use to insert text into a given range along the X axis - tests in 0.5 point size increments. You can specify whether or not you want the text truncated or not (within the given range).

pdf_insert_page

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Page Number as Integer	- Non-zero Integer value of the page to insert
	Where as Character	- Insert the page Before or After the page #
Sample Call:	RUN pdf_insert_page (“Spdf”,1,”AFTER”)	

Description:

This procedure allows you to insert a page anywhere in the stream while you are still building the document.

This is useful if you wanted to add a table of contents to your document. You could build your document as is but track the TOC elements then at the end of your document generation (and before running pdf_close) insert the TOC page as Page 1.

pdf_line

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	From Column Point as Integer	- A non-zero integer value representing the

		lines columnar starting point
	From Row Point as Integer	- A non-zero integer value representing the lines row starting point
	To Column Point as Integer	- A non-zero integer value representing the lines columnar ending point
	To Row Point as Integer	- A non-zero integer value representing the lines row ending point
	Line Weight as Decimal	- A non-zero decimal value representing the width that the line should be drawn with (a higher number indicates a thicker line)
Sample Call:	RUN pdf_line (“Spdf”,10,10,10,600,2,0.5)	

Description:

This procedure allows you to draw a line from a given starting point to a specified end point. You can determine the color of the line by using the pdf_stroke_color procedure. You can also change the appearance of the line by using the pdf_set_dash procedure.

pdf_line_dec

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	From Column Point as Decimal	- A non-zero decimal value representing the lines columnar starting point
	From Row Point as Decimal	- A non-zero decimal value representing the lines row starting point
	To Column Point as Decimal	- A non-zero decimal value representing the lines columnar ending point
	To Row Point as Decimal	- A non-zero decimal value representing the lines row ending point
	Line Weight as Integer	- A non-zero decimal value representing the width that the line should be drawn with (a higher number indicates a thicker line)
Sample Call:	RUN pdf_line_dec (“Spdf”,10.0,10.0,10.0,600.0,2.0,0.5)	

Description:

This procedure allows you to draw a line from a given starting point to a specified end point. You can determine the color of the line by using the pdf_stroke_color procedure. You can also change the appearance of the line by using the pdf_set_dash procedure.

This differs from pdf_line in that it uses decimal values for the X/Y coordinates. This allows for more exact positioning/drawing of the line.

pdf_link

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	From X as Integer	- A non-zero integer value representing the X axis starting point
	From Y as Integer	- A non-zero integer value representing the Y axis starting point

	Link Width as Integer	- How wide should the link boundry be?
	Link Height as Integer	- How high should the link boundry be?
	Link Text as Character	- Text to appear when link is hovered over
	Text Red as Integer	- Red value of RGB Colour
	Text Green as Integer	- Green value of RGB Colour
	Text Blue as Integer	- Blue value of RGB Colour
	Link Border as Integer	- Non-negative integer for the border width
	Link Style as Character	- Display style of link
Sample Call:	RUN pdf_line ("Spdf",10,10,10,600,2)	

Description:

This procedure allows you to place a rectangular link boundry into a PDF document. This can be useful when used with linking a Customer ID to another document that lists all Customer Invoices.
The link boundry could also be placed around an image (such as a logo image) that once clicked, could redirect someone to your corporate webpage.

Possible Style values are:
N – No highlighting (blank)
I – Invert the contents of the link
O – Invert the links border
P – Display a 'down' (or Pushed) appearance

pdf_load_font

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Name as Character	- Unique Font Name (no spaces)
	Font File as Character	- Location of TTF Font File (uses PROPATH)
	Font AFM File as Character	- Location of AFM file (uses PROPATH)
	Font DIF File as Character	- Location of DIF file (uses PROPATH)
Sample Call:	RUN pdf_load_font ("Spdf","Code39","c:\windows\fonts\Code39.ttf",".\code39.afm","")	

Description:

This procedure allows you to load and embed external fonts for later use. You can 'use' a font by using the pdf_set_font procedure.

The DIF file is optional and is only used if you want to remap some of the characters. The DIF file should have a format similar to the following:

```
65 /SF100000
66 /SF110000
67 /SF010000
68 /SF030000
69 /SF040000
70 /SF080000
71 /SF090000
72 /SF060000
73 /SF070000
74 /SF050000
75 /SF430000
76 /SF240000
77 /SF510000
78 /SF520000
79 /SF390000
80 /SF220000
81 /SF210000
82 /SF250000
83 /SF500000
84 /SF490000
85 /SF380000
86 /SF280000
87 /SF270000
88 /SF260000
89 /SF360000
90 /SF370000
```

The first entry represents the character to replace and the second entry represents the Postscript name of character to replace it with. The Postscript character names can be found at: http://partners.adobe.com/asn/tech/type/opentype/appendices/wgl4.jsp

pdf_load_image

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Image Name as Character	- Unique Font Name (no spaces)
Sample Call:	RUN pdf_load_image ("Spdf","ProSysLogo","logo.jpg")	

Description:

This procedure allows you to load and embed and external JEPG image. You can 'use' the image by using the pdf_place_image procedure. Currently, only JPEG images can be loaded using this procedure.

pdf_load_xml

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new

	XML Name as Character - XML File Name
Sample Call:	RUN pdf_load_image ("Spdf","c:\temp\mydata.xml")

Description:

This procedure allows you to load an XML file. The Node Data can then be accessed by referencing the TT_pdf_xml TEMP-TABLE. Then any of the data values etc can be output to your PDF document using the appropriate procedures (eg: pdf_text).

pdf_markup

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	Text as Character	- Text to appear in markup content
	Title as Character	- Small title for the Markup content
	Style as Character	- Predefined Style set **(CASE-SENSITIVE)**
	X1 as Decimal	- Lower Left X Coordinate
	Y1 as Decimal	- Lower Left Y Coordinate
	X2 as Decimal	- Lower Right X Coordinate
	Y2 as Decimal	- Lower Right Y Coordinate
	X3 as Decimal	- Upper Left X Coordinate
	Y3 as Decimal	- Upper Left Y Coordinate
	X4 as Decimal	- Upper Right X Coordinate
	Y4 as Decimal	- Upper Right Y Coordinate
	Red as Decimal	- decimal value representing Red colour value
	Green as Decimal	- decimal value representing Green colour value
	Blue as Decimal	- decimal value representing Blue colour value
Sample Call:	RUN pdf_markup("Spdf","My Markup Text","My Markup Title","Highlight",10.0,10.0,20,0,30.0,10.0,100.0,20.0,100.01.0,0.0,0.0)	

Description:

This procedure allows you to add a Style at a specific location (actually an area determined by the X/Y coordinates) on the current PDF page.

The Markup can contain as much as 32K of associated text.

The possible Markup Style are:
Highlight **(default)**
Underline
Squiggly
StrikeOut

pdf_merge_stream

Parameters:	Stream From as Character	- Stream name to copy
	Stream To as Character	- Stream name to copy to
	Copies as Integer	- Number of times to copy

Sample Call:	RUN pdf_merge_stream (“Spdf”,”Sxxx”,1)

Description:

This procedure allows you to copy one stream into another stream. But the two streams must be generated at the same time (that is, within the same process).

This is useful if you want to create separate documents (say customer invoices) but then have a summary document that includes all separate documents.

pdf_move_to

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	To Column Point as Integer	- A non-zero integer value representing a columnar location that you want to move the Graphic State cursor to
	To Row Point as Integer	- A non-zero integer value representing a row location that you want to move the Graphic State cursor to
Sample Call:	RUN pdf_line (“Spdf”,10,10)	

Description:

This procedure allows you to dynamically change Graphic State locations within the PDF stream.

pdf_new

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	File Name as Character	- The name of the PDF file to generate (as it will be stored on the Operating System)
Sample Call:	RUN pdf_new (“Spdf”,”c:\sports2000\sample.pdf”)	

Description:

This procedure initiates a new PDF stream to create with PDFinclude. The stream name must be unique and is used by most all other PDFinclude functions and procedures. By adding a stream name to each of the functions and procedures, we allow you to create multiple (unlimited) PDF documents from within a single Progress procedure.

When creating a new stream, this procedure loads the PDF Base 14 fonts (a defined set of fonts including Courier, Helvetica, etc) plus it includes setup of the following default parameters:

Parameter Name	**Default Value**
Orientation	Portrait
PaperType	LETTER
PageWidth	612
PageHeight	792
Render	0
TextX	0
TextY	0
Font	Courier
FontSize	10
GraphicX	0
GraphicY	0
VerticalSpace	10
LeftMargin	10
TopMargin	50

pdf_new_page

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
Sample Call:	RUN pdf_new_page (“Spdf”)	

Description:

This procedure initiates a creates a new PDF page to write output to. It also resets the current TextX, TextY, GraphicX and GraphicY parameters.

Note: The Progress default paging is not used, as is the Header, Footer, Page-Top, Page-Bottom, Page-Number, and Line-Counter options. Using PDFinclude requires that you control all output.

pdf_new_page2

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Orientation as Character	- Landscape/Portrait
Sample Call:	RUN pdf_new_page2 (“Spdf”,”Landscape”)	

Description:

This procedure initiates a creates a new PDF page to write output to. It also resets the current TextX, TextY, GraphicX and GraphicY parameters. Plus you can specify what the orientation should be for the new page.

This allows you to intermix portrait and landscape pages within the same PDF document.

pdf_note

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	Text as Character	- Text to appear in note
	Title as Character	- Small title for the note header
	Icon as Character	- Predefined icon set
	LLX as Decimal	- Lower Left X Coordinate
	LLY as Decimal	- Lower Left Y Coordinate
	URX as Decimal	- Upper Right X Coordinate
	URY as Decimal	- Upper Right Y Coordinate
	Red as Decimal	- decimal value representing Red colour value
	Green as Decimal	- decimal value representing Green colour value
	Blue as Decimal	- decimal value representing Blue colour value
Sample Call:	RUN pdf_note(“Spdf”,”My Note Text”,”My Note Title”,”Note”,10.0,10.0,20,0,30.0,1.0,0.0,0.0)	

Description:

This procedure allows you to add an annotation (or note) at a specific location (actually an area determined by the LL and UR X/Y coordinates) on the current PDF page.

The note can contain as much as 32K of text. This is useful if you have a large product description that won’t fit on the printed document but you’d like to have inserted in the viewable document.

The possible icon types are:
Note (default)
Comments
Insert
Key
Help
NewParagraph
Paragraph

The icons may differ based on viewer being used.

pdf_open_pdf

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	PDF Name as Character	- PDF File Name and Path (must exist)
	PDF ID as Character	- Unique PDF Identifier (eg: Invoice)

Sample Call: RUN pdf_open_pdf (“Spdf”,”c:\temp\myInvoice.pdf”,”INV”)

Description:

This procedure allows you to open a pre-existing PDF document. This procedure parses the identified PDF file and determines information about that file.

This procedure basically makes the contents of the existing PDF document available for use within your new document (see procedure pdf_use_pdf_page).

Very useful if you have an existing blank form (say an Invoice form) that you want to use as the basis for your page. You can then overlay data (either dynamically or statically) onto the form.

For more information on this, see the section called ‘Introducing Complex Forms’.

pdf_place_image

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Image Name as Character	- Valid Image name as pre-loaded with pdf_load_image function
	Graphic Column as Integer	- Non-zero integer value identifying the columnar location to begin drawing the image
	Graphic Row as Integer	- Non-zero integer value identifying the row location to begin drawing the image
	Width as Integer	- Non-zero integer value representing the width to render the image at
	Height as Integer	- Non-zero integer value representing the height to render the image at
Sample Call:	RUN pdf_place_image (“Spdf”, “ProSysLogo”,200,16)	

Description:

This procedure places a previously loaded image onto the current PDF page at the specified location.

This differs from pdf_place_image2 in that it originally drew the image above any text elements therefore obscuring the text.

NOTE: The functionality of PDFinclude has changed so that the placement of elements (text or graphic) are drawn onto the document as you call them therefore the usage of pdf_place_image2 is not really required but has been maintained for backward compatibility.

pdf_place_image2

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Image Name as Character	- Valid Image name as pre-loaded with pdf_load_image function
	Graphic Column as Integer	- Non-zero integer value identifying the columnar location to begin drawing the image

	Graphic Row as Integer	- Non-zero integer value identifying the row location to begin drawing the image
	Width as Integer	- Non-zero integer value representing the width to render the image at
	Height as Integer	- Non-zero integer value representing the height to render the image at
Sample Call:	RUN pdf_place_image (“Spdf”, “ProSysLogo”,200,16)	

Description:

This procedure places a previously loaded image onto the current PDF page at the specified location.

This differs from pdf_place_image in that it originally drew the image below any text elements (so it was useful for using an image as a watermark).

NOTE: The functionality of PDFinclude has changed so that the placement of elements (text or graphic) are drawn onto the document as you call them therefore the usage of pdf_place_image2 is not really required but has been maintained for backward compatibility.

pdf_rect

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	From Column as Integer	- Non-zero integer value identifying the columnar location to begin drawing the rectangle
	From Row as Integer	- Non-zero integer value identifying the row location to begin drawing the rectangle
	Width as Integer	- Non-zero integer value representing the width to render the rectangle at
	Height as Integer	- Non-zero integer value representing the height to render the rectangle at
	Weight as Decimal	- Value representing how thick a line to draw
Sample Call:	RUN pdf_rect (“Spdf”, 10,10,200,16,0.5)	

Description:

This procedure draws a rectangle onto the current PDF page at the specified location using the specified width and height.

This call uses both the Fill and Stroke colours when building the rectangle.

pdf_rect2

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	From Column as Integer	- Non-zero integer value identifying the columnar location to begin drawing the rectangle
	From Row as Integer	- Non-zero integer value identifying the row

		location to begin drawing the rectangle
	Width as Integer	- Non-zero integer value representing the width to render the rectangle at
	Height as Integer	- Non-zero integer value representing the height to render the rectangle at
	Weight as Decimal	- Value representing how thick a line to draw
Sample Call:	RUN pdf_rect2 (“Spdf”, 10,10,200,16,0.5)	

Description:

This procedure draws a rectangle onto the current PDF page at the specified location using the specified width and height.

This call uses only the Stroke colours when building the rectangle. This is useful if you want to draw a rectangle but want to see the information (such as an image or text) below the rectangle.

pdf_ReplaceTxt

Parameters:	Stream From as Character	- Stream name to copy
	MergeNbr as Character	- Stream name to copy to
	From Text as Character	- Number of times to copy
	To Text as Character	- Number of times to copy
Sample Call:	RUN pdf_ReplaceTxt (“Spdf”,1,”Invoice”,”Invoice – Copy”)	

Description:

This procedure is only used in conjunction with the pdf_merge_stream procedure. It allows you to change text within the stream content.

pdf_reset_all

Parameters:	None
Sample Call:	RUN pdf_reset_all.

Description:

This procedure clears all PDF content for all defined PDF streams.

pdf_reset_stream

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
Sample Call:	RUN pdf_reset_all.	

Description:

This procedure clears all PDF content for the named PDF stream.

pdf_rgb

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Target Procedure Name as Character	- Must be one of: pdf_text_color, pdf_stroke_color, pdf_stroke_fill
	RGB Color code as Character	- Must be either a six digit Hex RGB color code, preceded by ‘0x’, or ‘#’, or a nine digit decimal RGB color code, of the format RRRGGGBBB.
Sample Call:	RUN pdf_rgb (“Spdf”, “pdf_text_color”, “0x006466”) /* Hex Teal Green*/ RUN pdf_rgb (“Spdf”, “pdf_text_color”, “000100102”) /* Decimal Teal Green*/	

Description:

This procedure accepts a six digit Hex RGB color code (preceded by ‘0x’, or ‘#’, format RRGGBB), or a Nine digit decimal RGB color code (format RRRGGGBBB) and converts it into the three PDF RGB colors. It then runs the Target Procedure, passing in these three PDF colors.

pdf_set_BottomMargin

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Margin as Integer	- An integer value for the top of the Margin
Sample Call:	RUN pdf_set_BottomMargin (“Spdf”, 80)	

Description:

This procedure allows you to set the height of the Bottom Margin. As soon as the Text Y location goes greater than the Bottom Margin then a new PDF page is created. This is useful for formatting a document as it ensures that you don’t run text to the bottom of the PDF page.

This procedure is also used in conjunction with the PageFooter functionality (explained elsewhere). That is, if a PageFooter is defined and the Text Y location is greater than the Bottom Margin, then the PageFooter is automatically output to the PDF document.

pdf_set_dash

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Points On as Integer	- An integer value representing how many points to display
	Points Off as Integer	- An integer value representing how many

	points not to display
Sample Call:	RUN pdf_set_dash (“Spdf”, 1,0)

Description:

This procedure allows you to take a solid line (or rectangle) and adjust it to look like a dashed line. The On/Off parameter options allows you to define the appearance of the dashed line. You may want to have one short line appear (eg: On=1) then have a fairly large space where the line doesn’t appear at all (eg: Off=10).

pdf_set_FillBlue

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Decimal	- Value to set Blue option on RGB Colours
Sample Call:	RUN pdf_set_FillBlue (“Spdf”,1.0)	

Description:

This procedure allows you to individually change the Fill Blue value of the current RGB colour scheme. That is, if you had previously set the Fill to Black (eg: Red=0, Blue=0, Green=0) then you can easily change it to use blue fill by running the sample call (as above). Then you can easily set it back to black by using ‘RUN pdf_set_FillBlue (“Spdf”,0.0)’.

Possible Values: Any realistic decimal value between 0.0 and 1.0.

pdf_set_FillGreen

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Decimal	- Value to set Green option on RGB Colours
Sample Call:	RUN pdf_set_FillGreen (“Spdf”,1.0)	

Description:

This procedure allows you to individually change the Fill Green value of the current RGB colour scheme. That is, if you had previously set the fill to Black (eg: Red=0, Blue=0, Green=0) then you can easily change it to use green fill by running the sample call (as above). Then you can easily set it back to black by using ‘RUN pdf_set_FillGreen (“Spdf”,0.0).

Possible Values: Any realistic decimal value between 0.0 and 1.0.

pdf_set_FillRed

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Decimal	- Value to set Red option on RGB Colours

Sample Call:	RUN pdf_set_FillRed (“Spdf”,1.0)

Description:

This procedure allows you to individually change the Red Fill value of the current RGB colour scheme. That is, if you had previously set the fill to Black (eg: Red=0, Blue=0, Green=0) then you can easily change it to use red fill by running the sample call (as above). Then you can easily set it back to black by using ‘RUN pdf_set_FillRed (“Spdf”,0.0)’.

Possible Values: Any realistic decimal value between 0.0 and 1.0.

pdf_set_font

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Name as Character	- The name of a valid Font (either one of the PDF 14 Base fonts or a font previously loaded by the pdf_load_font function)
	Font Size as Decimal	- The size to display the font with
Sample Call:	RUN pdf_setfont (“Spdf”, “Code39”,10)	

Description:

This procedure allows you to set how the following text will be displayed. All text following this procedure will appear using the set font until the Font is set back to another font.

The following table defines the names of the PDF Base14 Fonts.

Note: Only 12 are defined right now. We still need to add the WingDings and Symbol Fonts.

Font Name	**Description**
Courier	Fixed-Width Courier Font
Courier-Bold	Bolded Fixed-Width Courier Font
Courier-Oblique	Italicized Fixed-Width Courier Font
Courier-BoldOblique	Bolded and Italicized Proportional-Width Helvetica Font
Helvetica	Proportional-Width Helvetica Font
Helvetica-Bold	Bolded Proportional-Width Helvetica Font
Helvetica-Oblique	Italicized Proportional-Width Helvetica Font
Helvetica-BoldOblique	Bolded and Italicized Proportional-Width Helvetica Font
Times	Proportional-Width Times Roman Font
Times-Bold	Bolded Proportional-Width Times Roman Font
Times-Italic	Italicized Proportional-Width Times Roman Font
Times-BoldItalic	Bolded and Italicized Proportional-Width Times Roman Font

pdf_set_GraphicX

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Integer	- Value to set the Graphic X coordinate to
Sample Call:	RUN pdf_set_GraphicX(“Spdf”,300)	

Description:
This procedure allows you to programmatically change the X coordinate of the Graphic drawing plane. Similar to pdf_move_to but you only get to set the X coordinate in this procedure. **Possible Values:** Any value greater than or equal to 1 and less than or equal to the Page Width.

pdf_set_GraphicY

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Integer	- Value to set the Graphic Y coordinate to
Sample Call:	RUN pdf_set_GraphicY("Spdf",300)	

Description:
This procedure allows you to programmatically change the Y coordinate of the Graphic drawing plane. Similar to pdf_move_to but you only get to set the Y coordinate in this procedure. **Possible Values:** Any value greater than or equal to 1 and less than or equal to the Page Height.

pdf_set_info

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Attribute Name as Character	- The name of a valid attribute that can be set
	Attribute Value as Character	- The value you want to assign to the specified Attribute
Sample Call:	RUN pdf_set_info ("Spdf","Author","Gordon Campbell")	

Description:

This procedure allows you to set values that help you define attributes about the PDF document itself. The following table outlines the settable attributes:

Attribute	Description
Author	Who created the document
Producer	What is the primary producer of the document (eg: PDFinclude)
Creator	What created the document (eg: The name of your procedure)
Keywords	Keywords that will help identify the contents of the PDF document
Subject	Any phrase that would describe the contents of the document eg: Accounts Payable
Title	Any phrase that would describe the contents of the document eg: Vendor Listing

pdf_set_LeftMargin

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Margin Value as Character	- The value you want to assign to the Left Margin
Sample Call:	RUN pdf_set_LeftMargin (“Spdf”,10)	

Description:

This procedure allows you to set the number of points that the document will be indented from the left-hand edge of the pages boundaries (identified by Page Height and Page Width).

pdf_set_Orientation

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Orientation as Character	- How you would like the page oriented
Sample Call:	RUN pdf_set_Orientation (“Spdf”,”Landscape”)	

Description:

This procedure allows you to define how the page should be displayed … upright (Portrait) or lengthwise (landscape). This value can be set before or after the Page Height or Page Width has been set but must be set before the call to the pdf_new_page procedure.

Possible Values: Portrait or Landscape (Portrait is the default)

pdf_set_PageHeight

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Page Height as Integer	- How tall is the page?

Sample Call:	RUN pdf_set_PageHeight (“Spdf”,792)

Description:

This procedure allows you to define how tall (or wide if using Landscape orientation) the page should be. This value can be set before or after the Page Width or Orientation has been set but must be set before the call to the pdf_new_page procedure. As soon as this procedure is used, the Paper Type is set to “CUSTOM”.

Possible Values: Any realistic integer value greater than zero

pdf_set_PageRotate

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Rotation Angle as Integer	- By what degree to rotate the contents
Sample Call:	RUN pdf_set_PageRotate(“Spdf”,90)	

Description:

This procedure allows you to determine how the contents of the page should be output to the page. That is, keeping the same page size (say Letter) and orientation (say Portrait), you can change how the contents should appear on the page. If you rotate the page 180 degrees, then the content will appear up-side-down on the Letter/Portrait page.

Possible Values: 0,90,180,270

pdf_set_PageWidth

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Page Width as Integer	- How wide is the page?
Sample Call:	RUN pdf_set_PageHeight (“Spdf”,612)	

Description:

This procedure allows you to define how wide (or tall if using Landscape orientation) the page should be. This value can be set before or after the Page Height or Orientation has been set but must be set before the call to the pdf_new_page function. As soon as this procedure is used, the Paper Type is set to “CUSTOM”.

Possible Values: Any realistic integer value greater than zero

pdf_set_PaperType

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Page Type as Character	- Enter a valid Paper Type

Sample Call: RUN pdf_set_PageType (“Spdf”,”LETTER”)

Description:

This procedure allows you to select Paper Types that have predefined height and widths. The following table outlines the valid Paper Types and their associate page dimensions.

Paper Type	Width	Height
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421
B5	501	709
LETTER	612	792
LEGAL	612	1008
LEDGER	1224	792

The Widths and Heights identified in the preceding table are based on the Portrait orientation. If you were to change the Orientation to ‘Landscape’ then the Width and Height columns would be reversed.

pdf_set_parameter

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Parameter Name as Character	- name of parameter to set
	Parameter Value as Character	- value to set parameter to

Sample Call: RUN pdf_set_parameter (“Spdf”,”Compress”,”True”)

Description:

This procedure allows you to set parameters associated with the specified PDF stream. The following table outlines the valid parameter names and potential values (please note that with using 40-bit encryption only certain parameters are useful or work).

Parameter Name	Description
Compress	The parameter allows you to specify that you want to compress the data streams. Compression occurs before encryption. **Possible Values**: TRUE or FALSE
Encrypt	This parameter allows you to encrypt your data streams. Encryption does not change the size of the data streams. **Possible Values**: TRUE or FALSE
UserPassword	This parameter allows you to add a User password to the document. This password must be entered whenever the document is opened. **Possible Values**: Any alpha-numeric character combination. Max 32 characters.
MasterPassword	This parameter allows you to add a Master password to the document. This password must be entered when trying to modify the security settings. **Possible Values**: Any alpha-numeric character combination. Max 32 characters.
EncryptKey	This parameter allows you to specify the level of encryption. Currently only handles 40-bit Encryption and is automatically set when the 'Encrypt' parameter is set to 'TRUE'. **Possible Values**: 40 (128 bit will be the other possible value in future versions)
AllowPrint	This parameter allows you to specify whether the PDF document is printable or not. **Possible Values**: TRUE or FALSE
AllowCopy	This parameter allows you to specify whether elements (such as text or Graphics) can be copied or not. **Possible Values**: TRUE or FALSE
AllowModify	This parameter allows you to specify whether elements (such as Form objects) can be modified or not. **Possible Values**: TRUE or FALSE
AllowAnnots	This parameter allows you to specify whether Annotations are allowed or not. **Possible Values**: TRUE or FALSE
AllowForms	This parameter allows you to specify whether Forms processing can be performed or not. **Possible Values**: TRUE or FALSE
AllowExtract	This parameter allows you to specify whether Extraction can be performed or not. **Possible Values**: TRUE or FALSE
AllowAssembly	This parameter allows you to specify whether Assembly can be performed or not. **Possible Values**: TRUE or FALSE

pdf_set_TextBlue

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Decimal	- Value to set Blue option on RGB Colours
Sample Call:	RUN pdf_set_TextBlue (“Spdf”,1.0)	

Description:

This procedure allows you to individually change the Blue value of the current RGB colour scheme. That is, if you had previously set the font to Black (eg: Red=0, Blue=0, Green=0) then you can easily change it to use blue text by running the sample call (as above). Then you can easily set it back to black by using ‘RUN pdf_set_TextBlue (“Spdf”,0.0)’.

Possible Values: Any realistic decimal value between 0.0 and 1.0.

pdf_set_TextGreen

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Decimal	- Value to set Green option on RGB Colours
Sample Call:	RUN pdf_set_TextGreen (“Spdf”,1.0)	

Description:

This procedure allows you to individually change the Green value of the current RGB colour scheme. That is, if you had previously set the font to Black (eg: Red=0, Blue=0, Green=0) then you can easily change it to use green text by running the sample call (as above). Then you can easily set it back to black by using ‘RUN pdf_set_TextGreen (“Spdf”,0.0).

Possible Values: Any realistic decimal value between 0.0 and 1.0.

pdf_set_TextRed

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Decimal	- Value to set Red option on RGB Colours
Sample Call:	RUN pdf_set_TextRed (“Spdf”,1.0)	

Description:

This procedure allows you to individually change the Red value of the current RGB colour scheme. That is, if you had previously set the font to Black (eg: Red=0, Blue=0, Green=0) then you can easily change it to use red text by running the sample call (as above). Then you can easily set it back to black by using ‘RUN pdf_set_TextRed (“Spdf”,0.0)’.

Possible Values: Any realistic decimal value between 0.0 and 1.0.

pdf_set_TextX

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Integer	- Value to set the Text X coordinate to
Sample Call:	RUN pdf_set_TextX(“Spdf”,300)	

Description:

This procedure allows you to programmatically change the X coordinate of the Text drawing plane. This is useful if you want to continue using the text drawing procedures (such as pdf_text, pdf_text_to, pdf_text_at etc) but want to change the X location within your program.

Possible Values: Any value greater than or equal to 1 and less than or equal to the Page Width.

pdf_set_TextY

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Integer	- Value to set the Text Y coordinate to
Sample Call:	RUN pdf_set_TextY(“Spdf”,300)	

Description:

This procedure allows you to programmatically change the Y coordinate of the Text drawing plane. This is useful if you want to continue using the text drawing procedures (such as pdf_text, pdf_text_to, pdf_text_at etc) but want to change the Y location within your program.

Possible Values: Any value greater than or equal to 1 and less than or equal to the Page Height.

pdf_set_tool_parameter

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Tool Name as Character	- Valid Tool Name (as defined in pdf_tool_add)
	Parameter as Character	- Valid Tool Parameter
	Column as Integer	- integer value (usage varies on tool)
	Value as Character	- Parameter Value
Sample Call:	RUN pdf_set_TextY(“Spdf”,”CustList”,” HeaderFont",0,"Helvetica-Bold")	

Description:

This procedure allows you to set a parameter that is allowed for a given tool.

For more information on what parameters are available for specific tools, see the section called 'Implementing Tools'.

pdf_set_TopMargin

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Margin Value as Character	- The value you want to assign to the Top Margin
Sample Call:	RUN pdf_set_TopMargin ("Spdf",50)	

Description:

This procedure allows you to set the number of points that the document will be indented from the top edge of the pages boundries (identified by Page Height and Page Width)

pdf_set_VerticalSpace

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Vertical Space as Integer	- The value you want to assign to Vertical Space
Sample Call:	RUN pdf_set_VerticalSpace ("Spdf",10)	

Description:

This procedure allows you to set the number of points that represents the vertical spacing of lines. A value of 12 is typical for Portrait oriented documents while 10 is fairly common for Landscape oriented documents.

pdf_skip

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
Sample Call:	RUN pdf_skip ("Spdf")	

Description:

This procedure will skip to the next line in the Text Space. This also updates the current TextX and TextY attributes for the stream.

pdf_skipn

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Number of Lines as Integer	- Number of lines to skip
Sample Call:	RUN pdf_skipn (“Spdf”,2)	

Description:

This procedure will skip *n* number of lines in the Text Space. This also updates the current TextX and TextY attributes for the stream. This saves you calling pdf_skip multiple times.

pdf_stamp

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	Text as Character	- Text to appear in note
	Title as Character	- Small title for the note header
	Stamp as Character	- Predefined icon set
	LLX as Decimal	- Lower Left X Coordinate
	LLY as Decimal	- Lower Left Y Coordinate
	URX as Decimal	- Upper Right X Coordinate
	URY as Decimal	- Upper Right Y Coordinate
	Red as Decimal	- decimal value representing Red colour value
	Green as Decimal	- decimal value representing Green colour value
	Blue as Decimal	- decimal value representing Blue colour value
Sample Call:	RUN pdf_stamp(“Spdf”,”My Stamp Text”,”My Stamp Title”,”Approved”,10.0,10.0,20,0,30.0,1.0,0.0,0.0)	

Description:

This procedure allows you to add a Stamp at a specific location (actually an area determined by the LL and UR X/Y coordinates) on the current PDF page.

The stamp can contain as much as 32K of associated text.

The possible Stamp types are:
Approved
Experimental
NotApproved
AsIs
Expired
NotForPublicRelease
Confidential
Final
Sold
Departmental
ForComment
TopSecret
Draft **(default)**
ForPublicRelease

The stamps icon may differ based on viewer being used.

pdf_stroke_color

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Red as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Red component of an RGB color value
	Green as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Green component of an RGB color value
	Blue as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Blue component of an RGB color value
Sample Call:	RUN pdf_stroke_color (“Spdf”,1.0,0.0,0.0)	

Description:

This procedure allows you to change the color PDF objects that are stroked. These include stroked text (see pdf_text_render), lines, and rectangle borders.

pdf_stroke_fill

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Red as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Red

		component of an RGB color value
	Green as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Green component of an RGB color value
	Blue as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Blue component of an RGB color value
Sample Call:	RUN pdf_stroke_fill (“Spdf”,1.0,0.0,0.0)	

Description:

This procedure allows you to change the color PDF objects that are filled. These include filled text (see pdf_text_render), and rectangles.

pdf_text

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
Sample Call:	RUN pdf_text (“Spdf”,”Hello World”)	

Description:

This procedure allows you to place the passed text string at the current TextX and TextY Text Space points.

pdf_text_align

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Align as Character	- LEFT,CENTER,RIGHT
	Column as Integer	- Any non-zero integer value
	Row as Integer	- Any non-zero integer value
Sample Call:	RUN pdf_text_align (“Spdf”,”Hello World”,”CENTER”, 60, 100)	

Description:

This procedure allows you to place the passed text string at a specific Row/Column (X/Y) Coordinate. How the text is placed at the coordinate depends on the Alignment.

LEFT – starts placing the text the passed Row/Column coordinate
CENTER – centers the text string over the Row/Column coordinate
RIGHT – places the end of the text string at the Row/Column coordinate

pdf_text_at

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Column as Integer	- Any non-zero integer value
Sample Call:	RUN pdf_text_at ("Spdf","Hello World",60)	

Description:

This procedure allows you to place the passed text string at column 60 (determined by current PointSize * passed Column Value) on the current Text row (TextX).

Note: The column value entered here is determined by the actual row that you would like to see the text appear beginning at. That is, assuming a PointSize of 10, instead of entering 600 to have the text begin at Column 60, just enter 60.

pdf_text_boxed_xy

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Column as Integer	- Any non-zero integer value
	Row as Integer	- Any non-zero value
	Width as Integer	
	Height as Integer	
	Justify as Character	- Currently Unused
	Weight as Integer	- Weight of Boxes line
Sample Call:	RUN pdf_text_boxed_xy ("Spdf","Hello World",10,100,15,105,"LEFT",1)	

Description:

This procedure allows you to specifically place the passed text string at the passed Row and Column. A box will be placed around the text.

Note: The column and row values passed to this function should represent the actual PDF points. That is , unlike pdf_text_at, if you want to have the text appear at point 600 then enter 600 (not 60).

pdf_text_center

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Column as Integer	- Any non-zero integer value
	Rowas Integer	- Any non-zero integer value
Sample Call:	RUN pdf_text ("Spdf","This is Centered Text",50,100)	

Description:

This centers the passed text on the given X,Y point.

pdf_text_char

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Integer	- Any value between 0 and 377
Sample Call:	RUN pdf_text_char (“Spdf”,63)	

Description:

This procedure outputs the character using the specified character code. This is useful if you want to change your font to ZapfDingbats then output a check mark (✓) into your text stream.

pdf_text_charxy

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Value as Character	- Any character code
	Column as Integer	- Any value between 1 and Page Height
	Row as Integer	- Any value between 1 and Page Width
Sample Call:	RUN pdf_text_charxy (“Spdf”,”063”,100,100)	

Description:

This procedure outputs the character using the specified character code at the specific location. This is useful if you want to change your font to ZapfDingbats then output a check mark (✓) onto your graphic plane.

pdf_text_color

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Red as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Red component of an RGB color value
	Green as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Green component of an RGB color value
	Blue as Decimal	- A decimal value not less than 0 and no greater than 1. Represents the Blue component of an RGB color value
Sample Call:	RUN pdf_textcolor (“Spdf”,1.0,0.0,0.0)	

Description:

This procedure allows you to set the color that the following text displays should appear in.

pdf_text_render

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Render Type as Integer	- An integer value representing how to render the text
Sample Call:	RUN pdf_text_render ("Spdf", 0)	

Description:

This procedure allows you to define how the text will be rendered. There are four possible values and they are outlined below.

Render Type	**Description**
0	Fill Text
1	Stroke Text
2	Fill, then Stroke Text
3	Neither Fill nor Stroke Text (invisible)

pdf_text_rotate

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Angle as Integer	- Angle to rotate and display text in
Sample Call:	RUN pdf_skip ("Spdf", 90)	

Description:

This procedure will set the rotation angle for displaying text in. This must be run before you display the text you want rotated.

Possible Values are:
0 – 0 Degrees (default)
90 – 90 Degrees
180 – 180 Degrees
270 – 270 Degrees

pdf_text_to

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Column as Integer	- Any non-zero value
Sample Call:	RUN pdf_text_to ("Spdf","Hello World",30)	

Description:

This procedure allows you to place the passed text string right-aligned to the specified column (determined by current PointSize * passed Column Value) on the current Text row (TextX).

Note: The column value entered here is determined by the actual row that you would like to see the text appear beginning at. That is, assuming a PointSize of 10, instead of entering 600 to have the text begin at Column 60, just enter 60.

pdf_text_xy

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Column as Integer	- Any non-zero value
	Row as Integer	- Any non-zero value
Sample Call:	RUN pdf_text_xy (“Spdf”,”Hello World”,10,100)	

Description:

This procedure allows you to specifically place the passed text string at the passed Row and Column.

Note: The column and row values passed to this procedure should represent the actual PDF points. That is , unlike pdf_text_at, if you want to have the text appear at point 600 then enter 600 (not 60).

pdf_text_xy_dec

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Any string value
	Column as Decimal	- Any non-zero value
	Row as Decimal	- Any non-zero value
Sample Call:	RUN pdf_text_xy_dec (“Spdf”,”Hello World”,10.0,100.00)	

Description:

This procedure allows you to specifically place the passed text string at the passed Row and Column.

This procedure is similar in functionality to pdf_text_xy except that it accepts decimal values for the Row/Column (or X/Y) graphic coordinates. This allows for more accurate placement of the text element.

pdf_tool_add

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Tool Name as Character	- Any string value
	Tool Type as Character	- valid tool type
	Tool Data as Handle	- Handle to TEMP-TABLE

Sample Call: RUN pdf_tool_add (“Spdf”,”CustList”,”TABLE”,TT_Customer)

Description:

This links to the pdftool.p procedure. The pdftool.p procedure has some predefined ‘tools’ that allow for simplified documents. The tool types available are:

TABLE	This allows you to easily create a columnar report that will span multiple pages.
CALENDAR	This allows you to easily create a calendar for a specific year/month. A calendar is associated with a specific page (can’t span pages) and you can have multiple calendars on a single page. You can also specify notes for a specific day within the calendar.
MATRIX	This allows you to easily create a columnar table that does not span multiple pages. A Matrix is associated with a single page and you can have multiple matrices on a single page. You control what data is entered in each cell of the matrix.

pdf_use_pdf_page

Parameters:	Stream as Character	- Stream name as identified in pdf_new
	PDF ID as Character	- Valid PDF ID as created in pdf_open_pdf
	Page as Integer	- Number of page to use from PDF ID

Sample Call: RUN pdf_use_pdf_page (“Spdf”, “INV”,1)

Description:

This procedure allows you to include a page from a pre-existing PDF document into your newly generated document.

This command can be called once per page. In other words, you can only include an existing page once per new page. But you can still use the pre-existing page as many times as you wish (on each page of your new document or not).

By using a page from a pre-existing document, it also enables you access to any Adobe Form Fields that have been embedded into the pre-existing document page. You can then write data to those field placement by using the pdf_fill_text procedure.

For more information on using pre-existing PDF documents, see the section called ‘Introducing Complex Forms’.

pdf_Watermark

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Text value to display
	Font as Character	- Name of Font to display Text in
	FontSize as Integer	- How large the text should appear
	Red as Decimal	- Value to set Red option for text Colour
	Blue as Decimal	- Value to set Blue option for text Colour

	Green as Decimal	- Value to set Green option for text Colour
	X as Integer	- X location to display text at
	Y as Integer	- Y location to display text at

Sample Call:

```
RUN pdf_set_WaterMark
        ("Spdf",
         "Sample Report",
         "Courier",
         16,
         1.0,
         0.0,
         0,0,
         100,
         500)
```

Description:

This procedure allows you to place a text watermark into the PDF document. The watermark will appear only on the Page where it is issued, so if you want to have it appear on all pages ensure that the command is reissued whenever a new page is issued.

pdf_Wrap_Text

Parameters:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Text value to display
	From Column as Integer	- Column at which to start Text output
	To Column as Integer	- Column to stop Text Output
	Alignment as Character	- how to align the text – left or right aligned?
	Max Y as Integer	- OUTPUT variable

Sample Call:

```
RUN pdf_set_TextRed
        ("Spdf",
         "This is a lot of text that could include line breaks etc",
         10,
         20,
         "left",
         OUTPUT CurrentY).
```

Description:

This procedure allows you to place text into the PDF document within a from/to column location. If the text is longer than the from/to location then it will automatically wrap the text to the next line and place the remaining text within the same from/to location as the previous line. This is useful when printing character fields that have been input via an Editor widget.

4.Function List

This section defines the functions available within PDFinclude PRO. For each function you are given the required arguments, expected or valid values, sample calls, and a detailed description of how (and when) the function should be used.

GetXMLNodeValue

Arguments:	Parent Name as Character	- Parent Node
	Node Name as Character	- Node Name
Returns:	Character	
Sample Call:	GetXMLNodeValue (“Invoice”,”InvoiceNumber”)	

Description:

This function returns the value of a unique Parent/Child Node combination from the TT_pdf_xml TEMP-TABLE (which was created using the pdf_load_xml procedure).

pdf_Angle

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_Angle (“Spdf”)	

Description:

This function returns the currently set angle that is being used for the Text Rotation.

pdf_BottomMargin

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_BottomMargin (“Spdf”)	

Description:

This function returns the currently set value of the PDF pages Bottom Margin.

pdf_FillBlue

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Decimal	
Sample Call:	pdf_FillBlue (“Spdf”)	

Description:
This function allows you to determine what the current Blue value of the current FILL RGB setting is.

pdf_FillGreen

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Decimal	
Sample Call:	pdf_FillGreen (“Spdf”)	

Description:
This function allows you to determine what the current Green value of the current FILL RGB setting is.

pdf_FillRed

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Decimal	
Sample Call:	pdf_FillRed (“Spdf”)	

Description:
This function allows you to determine what the current Red value of the current FILL RGB setting is.

pdf_Font

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Character	
Sample Call:	pdf_Font (“Spdf”)	

Description:

This function returns the currently selected font that is being used to display text.

pdf_Font_Loaded

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Name as Character	- Font Name
Returns:	Logical	
Sample Call:	pdf_Font_Loaded (“Spdf”,”Arial”)	

Description:

This function lets you determine whether a font has already been loaded or not.

pdf_FontType

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Name as Character	- Font Name
Returns:	Character	
Sample Call:	pdf_Font_Loaded (“Spdf”,”Arial”)	

Description:

This function returns what type of Font this is … FIXED or Proportional.

pdf_get_info

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Attribute Name as Character	- One of the valid Attributes (see below)
Returns:	Character	
Sample Call:	pdf_get_info (“Spdf”)	

Description:

This function returns the value of the specified Document Information attribute.

Possible attributes are listed in the table below:

Attribute	Description
Author	Who created the document
Creator	What created the document (eg: PDFinclude or the name of your procedure)
Keywords	Keywords that will help identify the contents of the PDF document
Subject	Any phrase that would describe the contents of the document eg: Accounts Payable
Title	Any phrase that would describe the contents of the document eg: Vendor Listing

pdf_GetNumFittingChars

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Text value to check
	From X as Integer	- Starting X Coordinate
	To X as Integer	- Ending X Coordinate
Returns:	Integer	
Sample Call:	pdf_get_NumFittingChars (“Spdf”, “This is my text example”, 10, 20).	

Description:

This function returns the index of the last character that will fit into the specified width. It does this by summing each characters AFM width (as specified in the tt_pdf_font.font_width array) and comparing with the required width (converted into these same units).

pdf_get_parameter

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Parameter Name as Character	- One of the valid parameter names
Returns:	Character	
Sample Call:	pdf_get_parameter (“Spdf”,”Encrypt”)	

Description:

This function returns the value of the specified Document parameter. For a list of valid document parameters see the ‘pdf_set_parameter’ procedure.

pdf_get_tool_parameter

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Tool Name as Character	- Tool Name
	Parameter Name as Character	- Valid Tool Parameter Name
	Tool Column as Integer	- Column ID
Returns:	Character	
Sample Call:	pdf_get_tool_parameter (“Spdf”,”CustList”,”HeaderFont”,0)	

Description:

This function returns the value of the specified Tool parameter. For a list of valid Tool parameters see the section on ‘Implementing Tools’.

pdf_get_wrap_length

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Text as Character	- Text value to check
	Width as Integer	- Maximum width required
Returns:	Integer	
Sample Call:	pdf_get_NumFittingChars (“Spdf”, “This is my text example”, 10).	

Description:

You can use the function to see how long a piece of text WOULD be if you were to wrap it.

pdf_GraphicX

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_GraphicX ("Spdf")	

Description:

This function returns the value of the current X (Column) location of the Graphic State.

pdf_GraphicY

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_GraphicY ("Spdf")	

Description:

This function returns the value of the current Y (Column) location of the Graphic State.

pdf_ImageDim

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Image Name as Character	- Image name as defined with a pdf_load_image
	Dimenstion Type as Character	- Either HEIGHT or WIDTH
Returns:	Integer	
Sample Call:	pdf_ImageDim ("Spdf","Logo","HEIGHT")	

Description:

This function returns the value of the Image Dimension that was requested with Dimension Type input parameter. This is useful because when you load the image you don't know the Image Width or Height. With this function you can determine either.

pdf_LeftMargin

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_LeftMargin ("Spdf")	

Description:

This function returns the value of the Left Margin.

pdf_Orientation

Arguments:	Stream Name as Character - Stream name as identified in pdf_new
Returns:	Character
Sample Call:	pdf_Orientation (“Spdf”)

Description:

This function returns the value of the Orientation parameter. The Orientation parameter is used to define how the page appears.

Possible Values: Portrait or Landscape

pdf_Page

Arguments:	Stream Name as Character - Stream name as identified in pdf_new
Returns:	Integer
Sample Call:	pdf_Page (“Spdf”)

Description:

This function returns the value of the current Page parameter. The Page parameter is incremented after every call to a pdf_new_page.

pdf_PageFooter

Arguments:	Stream Name as Character - Stream name as identified in pdf_new Proc Handle as Handle - Handle to ‘called from’ procedure Footer Name as Character - Footer procedure name from calling procedure
Returns:	Logical
Sample Call:	pdf_PageFooter (“Spdf”, THIS-PROCEDURE:HANDLE, “PageFooter”)

Description:

This function notified PDFinclude that the calling report procedure has a Page Footer procedure associated with it. If this has been set (with a non-blank value) then whenever a text element is to be displayed below the Bottom Margin (set by pdf_set_BottomMarging) this procedure (eg: PageFooter) will be called.

pdf_PageHeader

Arguments:	Stream Name as Character - Stream name as identified in pdf_new

	Proc Handle as Handle - Handle to 'called from' procedure Header Name as Character - Header procedure name from calling procedure
Returns:	Logical
Sample Call:	pdf_PageHeader ("Spdf", THIS-PROCEDURE:HANDLE, "PageHeader")

Description:

This function notified PDFinclude that the calling report procedure has a Page Header procedure associated with it. If this has been set (with a non-blank value) then on each call to pdf_new_page this procedure (eg: PageHeader) will be called.

pdf_PageHeight

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_PageHeight ("Spdf")	

Description:

This function returns the value of the current Page Height parameter. The Page Height parameter is used to determine the page boundaries. The Page Height parameter can either be set directly by using pdf_set_PageHeight or indirectly by calling pdf_PaperType.

pdf_PageRotate

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_PageRotate ("Spdf")	

Description:

This function returns the value of the current PageRotate parameter. Can only be set to 0,90,180, or 270.

pdf_PageWidth

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_PageWidth ("Spdf")	

Description:

This function returns the value of the current Page Width parameter. The Page Width parameter is used to determine the page boundaries. The Page Width parameter can either be set directly by using pdf_set_PageWidth or indirectly by calling pdf_PaperType.

pdf_PaperType

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Character	
Sample Call:	pdf_PaperType (“Spdf”)	

Description:

This function returns the value of the current Paper Type parameter. The Paper Type parameter is used to determine the page boundaries. The Paper Type parameter is set by calling pdf_set_parameter and selecting a valid Paper Type or by calling pdf_set_PageHeight or pdf_set_PageWidth. Calling either of the two latter functions will set the Paper Type to ‘CUSTOM’.

Valid Paper Types are:

Paper Type	**Width**	**Height**
A0	2380	3368
A1	1684	2380
A2	1190	1684
A3	842	1190
A4	595	842
A5	421	595
A6	297	421
B5	501	709
LETTER	612	792
LEGAL	612	1008
LEDGER	1224	792

The Widths and Heights identified in the preceding table are based on the Portrait orientation. If you were to change the Orientation to ‘Landscape’ then the Width and Height columns would be reversed.

pdf_PointSize

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_PointSize (“Spdf”)	

Description:

This function returns the value of the current Point Size parameter. The Point Size parameter is used to determine how large a text object will be drawn on the PDF page. The Point Size is set by calling the pdf_set_Font function.

pdf_Render

Arguments:	Stream Name as Character - Stream name as identified in pdf_new
Returns:	Integer
Sample Call:	pdf_Render ("Spdf")

Description:

This function returns the value of the current Text Rendering parameter. The Test Rendering parameter is used to determine how the text will be drawn on the PDF page. The Text Rendering parameter is set by calling the pdf_Text_Render function.

The following table outline the possible values for the Text Rendering parameter:

Render Type	**Description**
0	Fill Text
1	Stroke Text
2	Fill, then Stroke Text
3	Neither Fill nor Stroke Text (invisible)

pdf_text_fontwidth

Arguments:	Stream Name as Character - Stream name as identified in pdf_new Font Name as Character - The Font Tag name Text Value as Character - The text you want to determine the width of
Returns:	Decimal
Sample Call:	pdf_text_widthdec2 ("Spdf","Arial","End of Report")

Description:

This function is similar in functionality to the pdf_text_widthdec2 function except that you only pass in the Font. The Font Size is determined by using the current Point Size (as set by pdf_set_font).

pdf_text_width

Arguments:	Stream Name as Character - Stream name as identified in pdf_new Text Value as Character - The text you want to determine the width of
Returns:	Integer
Sample Call:	pdf_text_width ("Spdf","End of Report")

Description:

This function allows you to determine the length (in points) of the passed text string using the current Font and PointSize. The width is calculated as follows:

*width = (LENGTH(text value) * individual character lengths / 1000) * pdf_PointSize(PDF Stream)*

The individual character lengths may differ for each character within a given font. The accumulation of the total character widths divided by 1000 then multiplied by the current font Point Size. This function is useful when used in conjunction with the pdf_text_boxed_xy function (allowing you to determine how wide the box around the text should be).

pdf_text_widthdec

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Text Value as Character	- The text you want to determine the width of
Returns:	Decimal	
Sample Call:	pdf_text_widthdec (“Spdf”,”End of Report”)	

Description:

This function is similar in functionality to the pdf_text_width function except that it returns the value in decimal format. This allows for more exact determination of the text width.

pdf_text_widthdec2

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
	Font Tag as Character	- The Font Tag name
	Font Size as Decimal	- Stream name as identified in pdf_new
	Text Value as Character	- The text you want to determine the width of
Returns:	Decimal	
Sample Call:	pdf_text_widthdec2 (“Spdf”,”\Arial”,8.0,”End of Report”)	

Description:

This function is similar in functionality to the pdf_text_widthdec function except that you pass in the Font and Font Size you want to use to determine the Text width.

pdf_TextBlue

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Decimal	
Sample Call:	pdf_TextBlue (“Spdf”)	

Description:

This function allows you to determine what the current Blue value of the current RGB setting is.

pdf_TextGreen

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Decimal	
Sample Call:	pdf_TextGreen ("Spdf")	

Description:
This function allows you to determine what the current Green value of the current RGB setting is.

pdf_TextRed

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Decimal	
Sample Call:	pdf_TextRed ("Spdf")	

Description:
This function allows you to determine what the current Red value of the current RGB setting is.

pdf_TextX

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_TextX("Spdf")	

Description:

This function returns the current X (Column) location within the Text Space.

pdf_TextY

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_TextY ("Spdf")	

Description:

This function returns the current Y (Row) location within the Text Space.

pdf_TotalPages

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Character	

Sample Call:	pdf_TotalPages (“Spdf”)

Description:

This function allows you to include the Total Number of Pages for the document directly into your report. For instance, if you wanted to have 'Page n of x' as your Page footer then you can implement this with pdf_TotalPages(streamname).

pdf_TopMargin

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_TopMargin (“Spdf”)	

Description:

This function returns the current Top Margin point setting.

pdf_VerticalSpace

Arguments:	Stream Name as Character	- Stream name as identified in pdf_new
Returns:	Integer	
Sample Call:	pdf_VerticalSpace (“Spdf”)	

Description:

This function returns the current point size for the Vertical Space. The Vertical Space is used to define the spacing between lines.

5.Internal Procedures and Functions

This section outlines the other procedures and functions that are used within PDFinclude PRO. These procedure and functions should not be used within programs as they may cause issues when developing a PDF document.

Name	Type	Description
Compress	Procedure	Defines External Compress Procedure in the Zlib DLL or Shared Library
UnCompress	Procedure	Defines External UnCompress Procedure in the Zlib DLL or Shared Library
First_Marker	Function	Used in determination of Image Dimensions
Hex	Function	Used in determination of Image Dimensions
Nextbyte	Function	Used in determination of Image Dimensions
Next2Bytes	Function	Used in determination of Image Dimensions
Next_Marker	Function	Used in determination of Image Dimensions
ObjectSequence	Function	Used to increment the PDF Object Counter. The objects represent different dictionaries within the PDF document. The start and end bytes of each Object must be stored so that the XREF dictionary can be built correctly.
Pdf_Catalog	Procedure	This procedure creates the PDF Catalog Object.
Pdf_Content	Procedure	This procedure is the main content generator of the PDF document.
Pdf_Definition	Procedure	This procedure creates the Page Dictionaries (and Annotations) for the PDF document.
pdf_error	Procedure	This procedure is used to track errors that have occurred during the building of the PDF document. Sample errors include incorrectly defined parameter options or incorrect stream names. Any errors are displayed to the default Progress stream and the PDF document is not generated.
pdf_set_Angle	Procedure	This procedure stores the value of the Angle as defined in pdf_Text_Rotation so that it can be retrieved via pdf_Angle.
Pdf_Encoding	Procedure	This procedure creates the PDF Encoding Object.
Pdf_Fonts	Procedure	This procedure creates the appropriate Font Dictionaries for Base 14 fonts.
Pdf_FontType	Function	This function returns whether the current font is FIXED or VARIABLE (Proportional)
Pdf_Get_Image_WH	Procedure	Used in determination of Image Dimensions
Pdf_Header	Procedure	Output the beginning of the PDF Document plus the Document Information Object (Object #1)
Pdf_inc_ContentSequenc e	Function	This function increments the Object Sequence (Object Number).
Pdf_init_param	Procedure	This procedure initializes the PDF document for a given stream.
Pdf_Length	Procedure	This procedure writes the Length Object for another associated Object. Example, a Page object would have an associated Length Object.
Pdf_ListPages	Procedure	This procedure writes the Page Object Dictionary.
Pdf_LoadBase14	Procedure	This procedure loads the Base 14 fonts, for a PDF

		stream, so that the font can be used when generating the PDF (eg: when using pdf_Set_font)
Pdf_Load_Fonts	Procedure	This procedure writes the Font Dictionaries for the TTF Fonts that were embedded via pdf_load_font.
Pdf_Load_Images	Procedure	This procedure writes the Image Dictionaries for the JPEG Images that were embedded via pdf_load_image.
Pdf_Load_Links	Procedure	This procedure writes the Annotation Dictionaries for the links that were included via pdf_link.
Pdf_LoadParseAFMFile	Procedure	This procedure parses the AFM file specified in a pdf_Load_font call.
Pdf_Replace_Text	Procedure	This procedure replaces any special characters in the content that would result in an invalid PDF being generated.
Pdf_Resources	Procedure	This procedure writes the Resource Object Dictionary.
pdf_set_GraphicX	Procedure	This procedure sets the current graphic plane X axis location whenever a graphic element is added to the PDF document.
pdf_set_GraphicY	Procedure	This procedure sets the current graphic plane Y axis location whenever a graphic element is added to the PDF document.
pdf_set_page	Procedure	This procedure increment the PDF page number whenever a new page is required.
Pdf_Xref	Procedure	This procedure writes the XREF Object Dictionary.
Process_SOF	Function	Used in determination of Image Dimensions
ReleasePDFencryptlib	Procedure	Releases the Encryption DLL or Shared Library
Release Zlib	Procedure	Releases the Zlib DLL or Shared Library
Skip_Variable	Function	Used in determination of Image Dimensions

6.Component List

This section lists and describes the components that make-up the PDFinclude PRO package.

Component	Description
pdf_inc.i	Main include file. Creates initial GLOBAL parameters (and TEMP-TABLES used by other PDFinclude components) via pdfglobal.i. Calls main pdf_inc.p procedure FORWARD declares Function declarations (via pdf_func.i).
pdf_inc.p	Main PDFinclude procedure. Defines all functions and procedures available within PDFinclude.
pdf_func.i	FORWARD declares Function declarations
pdfglobal.i	Defines GLOBAL parameters and TEMP-TABLES
pdftool.p	Tool Library
pdfextract.p	PDF Parser component that allows for inclusion of existing PDF documents into PDFinclude generated documents.
pdfencrypt.p	PDF Encryption component (40-Bit Encryption)
pdf_pre.i	Preprocessor definitions. The definitions maintained in this include file are used throughout PDFinclude PRO.

7.How to Implement Page Headers and Footers

Page Headers and Footers allow you to define code blocks that will appear whenever a page is generated. The code blocks allow you to include both text and graphic elements into your PDF document.

Page Headers appear whenever a call to pdf_new_page is issued. The Page Header is generated before anything else can be included in the PDF content of the page, ensuring that the Page Header content actually appears at the top of the page … unless you've used an element that uses specific X/Y positioning (eg: pdf_text_xy or pdf_place_image), then that element can be placed anywhere on the current page (useful if you wanted a watermark to appear on a page).

Page Footers appear only if you have set the Bottom Margin of the page (using pdf_set_BottomMargin). If any text elements (except X/Y positional text elements) are going to be displayed below the Bottom Margin setting, then the Page Footer code block is run and a call to pdf_new_page is issued .. to automatically increment the PDF paging. Ensure that when you call pdf_set_BottomMargin you set the value to something that will allow your Footer to appear correctly. That is, if you set the value to low (say 10) you may not get a page footer, or if your footer included graphic elements those elements may overlay some of the text.

To create Page Footers and Page Headers for a report, you need to create code blocks (procedures) in your program that will generate the appropriate output. For example, the following procedure illustrates a PageFooter:

```
PROCEDURE PageFooter:
/*------------------------------------------------------------------------------
  Purpose:  Procedure to Print Page Footer -- on all pages.
------------------------------------------------------------------------------*/
  /* Display a Sample Watermark on every page */
  RUN pdf_watermark ("Spdf","Customer List","Courier-Bold",34,.87,.87,.87,175,500).

  RUN pdf_skip ("Spdf").
  RUN pdf_set_dash ("Spdf",1,0).
  RUN pdf_line  ("Spdf", 0, pdf_TextY("Spdf") - 5, pdf_PageWidth("Spdf") - 20 ,
pdf_TextY("Spdf") - 5, 1).
  RUN pdf_skip ("Spdf").
  RUN pdf_skip ("Spdf").
  RUN pdf_text_to  ("Spdf",  "Page: "
                        + STRING(pdf_page("Spdf"))
                        + " of " + pdf_TotalPages("Spdf"), 97).

  vlines = 1. /* Restart our count for the linking */

END. /* PageFooter */
```

But this code won't be run unless we communicate this procedure to PDFinclude PRO. To do that you need to make the appropriate call. The two calls (functions) that notify PDfinclude PRO that you have included headers and footers in your report are:

```
pdf_PageFooter (<stream name>,THIS-PROCEDURE:HANDLE,  <Procedure Name>).
pdf_PageHeader (<stream name>,THIS-PROCEDURE:HANDLE, <Procedure Name>).
```

So, for our footer example, we would need to include the following function call (after referencing pdf_inc.i) in our program.

pdf_PageFooter (“Spdf”,THIS-PROCEDURE:HANDLE,"PageFooter").

This function call lets PDFinclude know that we have a footer procedure in our calling program.

8.How to Implement Compression

PDFinclude PRO uses the Zlib Compression Library (Zlib) to implement Flate compression. Zlib is a free open-source project and the binaries (or source) for many operating systems can be found at www.zlib.net.

Steps to implementing compression:

1. Download and install the appropriate binary from www.zlib.net

2. Open the file pdf_pre.i Progress include file in the PDFinclude PRO download. Change the **zlib** pre-processor to point to the appropriate DLL or Shared Library object.

   Eg: For Windows installs the zlib pre-processor would have the value of zlib1.dll defined like:

   ```
   &GLOBAL-DEFINE zlib zlib1.dll
   ```

3. Add the following command to each of the PDF documents that you wish to compress:

   RUN pdf_set_parameter(<stream name>,"Compress","TRUE").

   This statement can be run anytime between the calls to 'pdf_new' and 'pdf_close'. If you happen to run the statement multiple times before 'pdf_close', then the last call will be used to determine whether compression is to be used or not.

4. Run your code and you will see a reduction in your PDF filesize.

PDFinclude PRO does compress images but since they are already compressed, you may find that they aren't further reduced by very much.

9.How to Implement Encryption

The Encryption of the data stream is performed after compression … if compression is used that is.

PDFinclude PRO uses external binaries (a DLL or Shared Library object) to perform the encryption.

For Windows, a DLL (procryptlib.dll) is included in the download.

If you are on a *NIX OS then you will need to compile the rc4.c object into a Shared Library

Steps to implementing encryption:

1. Copy pdfencrypt.p to the same location as pdf_inc.p etc

2. Edit pdf_pre.i and modify the pre-processor **MD5LIB** to point to the appropriate md5 routine.

On Windows:	use the md5.exe binary included in the download
On *NIX:	point to the appropriate *NIX command (eg: /usr/lib/md5sum)

3. Edit pdf_pre.i and modify the pre-processor **pdfencryptlib** to point to the appropriate DLL or Shared Library object (this will contain the external function 'endecrypt')

4. Add the following command to each of the PDF documents that you wish to encrypt:

   RUN pdf_set_parameter(<stream name>,"Encrypt","TRUE").

   This statement can be run anytime between the calls to 'pdf_new' and 'pdf_close'. If you happen to run the statement multiple times before 'pdf_close', then the last call will be used to determine whether encryption is to be used or not.

   You can also set the **Allow***, **UserPassword** and/or **MasterPassword** document parameters to determine the permissions (see 'pdf_set_parameter' procedure documentation for more info) associated with the PDF document.

5. Run your procedures and see the encrypted (or protected) PDF document.

Currently PDFinclude PRO currently only utilizes 40-Bit encryption.

10.Implementing Tools

Tools are pre-built components that allow you to quickly and easily create specific content types (or tool types).

To add a tool to a PDFinclude PRO document you need call the ‘pdf_add_tool’ procedure.

Syntax: RUN pdf_add_tool(<Stream>,<ToolType>,<ToolID>).

Tool Parameters are used to control the output and look-and-feel of the Tool. To set parameters (listed in the following sections) for a tool you need to call the ‘pdf_set_tool_parameter’ procedure.

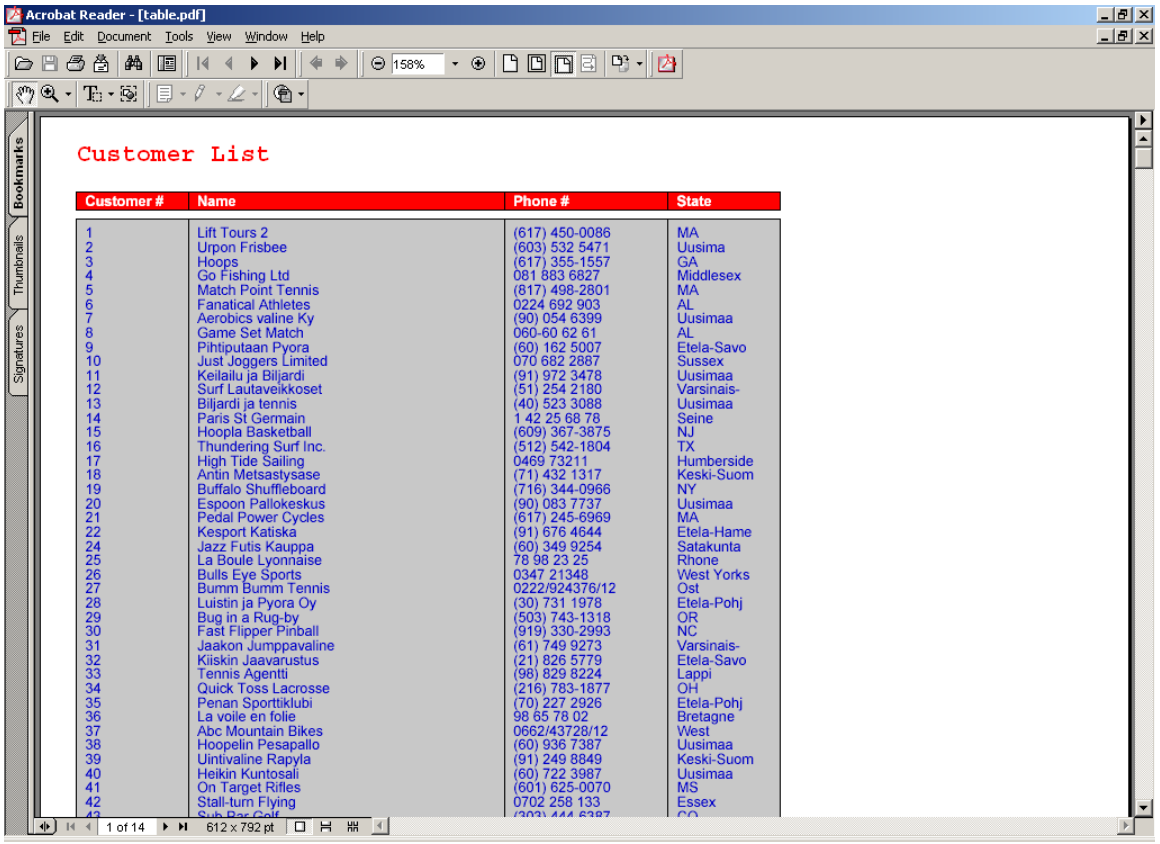
Syntax:

RUN pdf_set_tool_parameter(<Stream>,<ToolID>,<ParamName>,<Indicator>,<ParamValue>)

This section outlines the tools that are available via the pdftool.p procedure.

10.1TABLE Tool

This Table tool allows you to generate a multi-page columnar report. For example, helps to produce a Customer Listing (as illustrated below):


Acrobat Reader - [table.pdf]
File Edit Document Tools View Window Help
158%
Bookmarks
Thumbnails
Signatures
1 of 14
612 x 792 pt

Customer List

Customer #	Name	Phone #	State
1	Lift Tours 2	(617) 450-0086	MA
2	Urpon Frisbee	(603) 532 5471	Uusima
3	Hoops	(617) 355-1557	GA
4	Go Fishing Ltd	081 883 6827	Middlesex
5	Match Point Tennis	(817) 498-2801	MA
6	Fanatical Athletes	0224 692 903	AL
7	Aerobics valine Ky	(90) 054 6399	Uusimaa
8	Game Set Match	060-60 62 61	AL
9	Pihtiputaan Pyora	(60) 162 5007	Etela-Savo
10	Just Joggers Limited	070 682 2887	Sussex
11	Keilailu ja Biljardi	(91) 972 3478	Uusimaa
12	Surf Lautaveikkoset	(51) 254 2180	Varsinais-
13	Biljardi ja tennis	(40) 523 3088	Uusimaa
14	Paris St Germain	1 42 25 68 78	Seine
15	Hoopla Basketball	(609) 367-3875	NJ
16	Thundering Surf Inc.	(512) 542-1804	TX
17	High Tide Sailing	0469 73211	Humberside
18	Antin Metsastysase	(71) 432 1317	Keski-Suom
19	Buffalo Shuffleboard	(716) 344-0966	NY
20	Espoon Pallokeskus	(90) 083 7737	Uusimaa
21	Pedal Power Cycles	(617) 245-6969	MA
22	Kesport Katiska	(91) 676 4644	Etela-Hame
24	Jazz Futis Kauppa	(60) 349 9254	Satakunta
25	La Boule Lyonnaise	78 98 23 25	Rhone
26	Bulls Eye Sports	0347 21348	West Yorks
27	Bumm Bumm Tennis	0222/924376/12	Ost
28	Luistin ja Pyora Oy	(30) 731 1978	Etela-Pohj
29	Bug in a Rug-by	(503) 743-1318	OR
30	Fast Flipper Pinball	(919) 330-2993	NC
31	Jaakon Jumppavaline	(61) 749 9273	Varsinais-
32	Kiiskin Jaavarustus	(21) 826 5779	Etela-Savo
33	Tennis Agentti	(98) 829 8224	Lappi
34	Quick Toss Lacrosse	(216) 783-1877	OH
35	Penan Sporttiklubi	(70) 227 2926	Etela-Pohj
36	La voile en folie	98 65 78 02	Bretagne
37	Abc Mountain Bikes	0662/43728/12	West
38	Hoopelin Pesapallo	(60) 936 7387	Uusimaa
39	Uintivaline Rapyla	(91) 249 8849	Keski-Suom
40	Heikin Kuntosali	(60) 722 3987	Uusimaa
41	On Target Rifles	(601) 625-0070	MS
42	Stall-turn Flying	0702 258 133	Essex
43	Sub Par Golf	(303) 444 6387	CO

To use a Table you must supply a data set (TEMP-TABLE Handle) when running the pdf_tool_add procedure.

The Table tool has a number of parameters that can be set. Those parameters are outlined below.

Parameter Name	**Description**
ColumnHeader	For each column, you must specify the header (or ‘column-label’ in Progress-speak). Use the ‘Indicator’ value to specify which column should have which label. eg: RUN pdf_set_tool_parameter(“Spdf”, “TBL”,”ColumnHeader”,**1**,”Customer #”). In this case the **1** tells pdftool.p that the column label for the first column should be “Customer #”.
ColumnPadding	This represents the space between the line and the start of the column text. It is set for the table (not per cell or column) therefore the indicator must be zero (0).
ColumnWidth	For each column you need to specify the width (in points). Since this is a column specific parameter, you must use the Indicator to specify which column you are referring to. If no width is specified then pdftool.p uses the Field’s format to determine the width.
ColumnX	For each column you need to specify the starting point of the column. Since this is a column specific parameter, you must use the Indicator to specify which column you are referring to.
DetailBGColor	This represents the background colour for each of the detail lines in the columnar output (eg: the gray background in the table sample). Since this is a table specific parameter the Indicator must be zero (0).
DetailFont	This represents the font used for each of the detail lines in the columnar output. Since this is a table specific parameter the Indicator must be zero (0).
DetailFontSize	This represents the font size used for each of the detail lines in the columnar output. Since this is a table specific parameter the Indicator must be zero (0).
DetailTextColor	This represents the colour used to display the text in, for each of the detail lines in the columnar output.

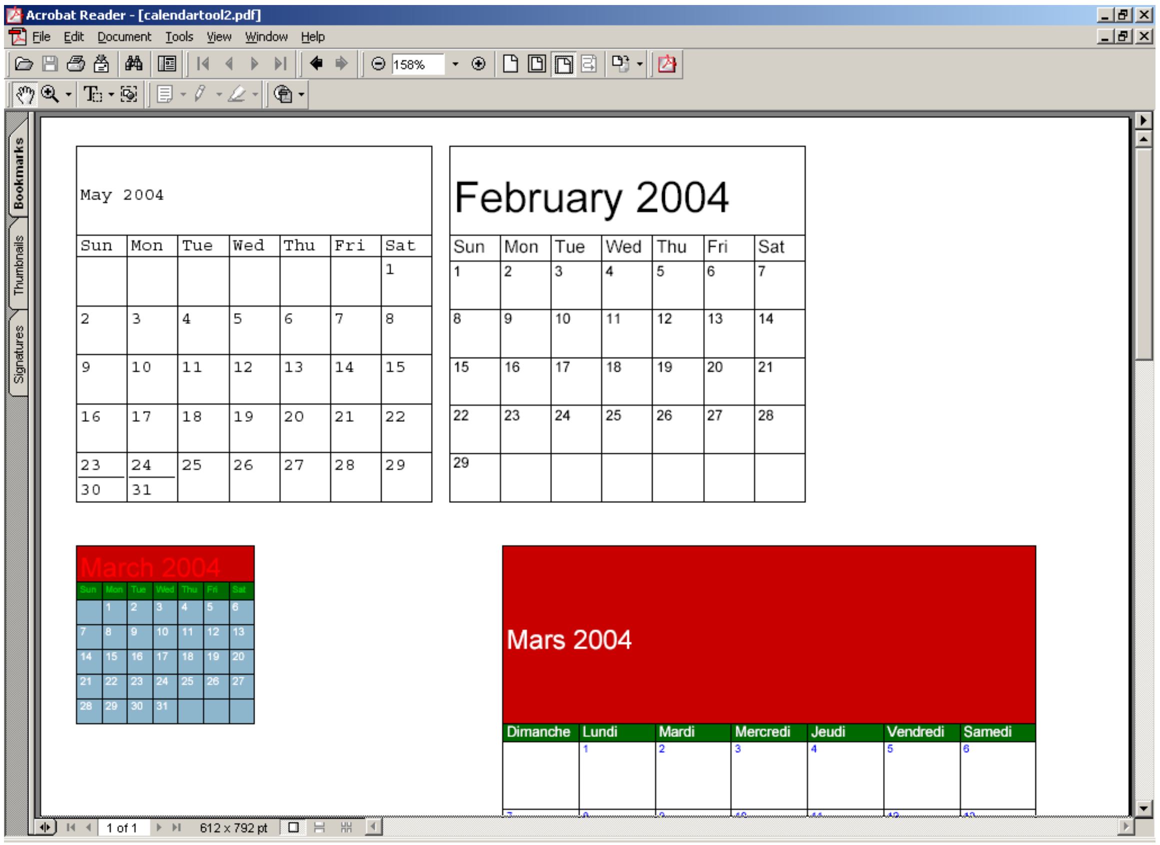
	Since this is a table specific parameter the Indicator must be zero (0).
HeaderBGColor	This represents the background colour for column headers (eg: the red background in the table sample). Since this is a table specific parameter the Indicator must be zero (0).
HeaderFont	This represents the font used for each of the column headers. Since this is a table specific parameter the Indicator must be zero (0).
HeaderFontSize	This represents the font size used for each of the column headers. Since this is a table specific parameter the Indicator must be zero (0).
HeaderTextColor	This represents the colour used to display the text in, for each column header. Since this is a table specific parameter the Indicator must be zero (0).
MaxX	This value sets the maximum value allowed for the X coordinate of a column. Usually, this is set by pdftool.p and is primarily used to draw the rectangle (outline) around the column. Since this is a column specific parameter, the indicator must represent the column you wish to adjust.
MaxY	This value sets the maximum value allowed for the Y coordinate of the table. Usually, this is set by pdftool.p and is primarily used to determine the height of the outline box. Since this is a table specific parameter the Indicator must be zero (0).
Outline	This should be a decimal value representing the width of the outline box. If zero, then no box will appear around the table elements. If non-zero, then a box will be drawn with the specified width. Since this is a table specific parameter the Indicator must be zero (0).

For an example of how to implement the TABLE tool, check out the *samples/super/table.p* procedure.

10.2CALENDAR Tool

This Calendar tool allows you to generate a calendar for a given Year/Month combination. The Calendar ID is associated with a specific page but you can create a calendar on any page and you can create as many calendars as you like on a specific page.

The following illustrates multiple calendars on a single page.


Acrobat Reader - [calendartool2.pdf]
File Edit Document Tools View Window Help
158%
Bookmarks
Thumbnails
Signatures
1 of 1
612 x 792 pt

May 2004

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23 30	24 31	25	26	27	28	29

February 2004

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29						

March 2004

Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Mars 2004

Dimanche	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi
	1	2	3	4	5	6

The Calendar tool has a number of parameters that can be set. Those parameters are outlined below.

Parameter Name	Description
DayBGColor	This parameter allows you to specify what the background colour will be for all the days within the calendar. Since this is a calendar specific parameter the Indicator must be zero (0).
DayFont	This parameter allows you to control the font to be used to display all days in, or you can specify the font for a particular day. If the Indicator is zero, then the font specified becomes the default for all days. If the Indicator is non-zero, and it is the same value as a day in the calendar, then the default font is overridden and the specified font is used instead.
DayFontColor	This parameter allows you to control the font colour to be used to display all days in, or you can specify the font colour for a particular day. If the Indicator is zero, then the font colour specified becomes the default for all days.

	If the Indicator is non-zero, and it is the same value as a day in the calendar, then the default font colour is overridden and the specified font colour is used instead.
DayFontSize	This parameter allows you to control the font size to be used to display all days in, or you can specify the font size for a particular day. If the Indicator is zero, then the font size specified becomes the default for all days. If the Indicator is non-zero, and it is the same value as a day in the calendar, then the default font size is overridden and the specified font size is used instead.
DayHighlight	This parameter allows you to highlight a given day within the calendar. The Indicator must represent a day within the calendar's month. The parameter value is a textual value representing 'why' you want the highlight.
DayLabelBGColor	This parameter allows you to specify the background colour where the Day Labels (eg: Mon. Tues. Wed etc) are displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
DayLabelFont	This parameter allows you to specify the Font where the Day Labels (eg: Mon. Tues. Wed etc) are displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
DayLabelFontColor	This parameter allows you to specify the Font colour where the Day Labels (eg: Mon. Tues. Wed etc) are displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
DayLabelFontSize	This parameter allows you to specify the Font size where the Day Labels (eg: Mon. Tues. Wed etc) are displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
DayLabelHeight	This parameter allows you to specify the height of the box surrounding where the Day Labels (eg: Mon. Tues. Wed etc) are displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
DayLabelY	Usually not specified. But if required, allows you to specifically set the Y coordinate where each day label will appear.

	If used, then the Instance should be 1 through 7.
HeaderBGColor	This parameter allows you to specify the background colour where the Header Label (usually month name) is displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
HeaderHeight	This parameter allows you to specify the height of the box where the Header Label (usually month name) is displayed. Since this is a calendar specific parameter the Indicator must be zero (0).
HeaderFont	This parameter allows you to specify the Font of the Header Label (usually month name). Since this is a calendar specific parameter the Indicator must be zero (0).
HeaderFontColor	This parameter allows you to specify the Font colour of the Header Label (usually month name). Since this is a calendar specific parameter the Indicator must be zero (0).
HeaderFontSize	This parameter allows you to specify the Font Size of the Header Label (usually month name). Since this is a calendar specific parameter the Indicator must be zero (0).
Height	This parameter allows you to specify the total height of the Calendar tool. Since this is a calendar specific parameter the Indicator must be zero (0).
HighlightColor	If a day is to be highlighted (see DayHighlight), then this lets you determine what colour it should be highlighted in. Since this is a calendar specific parameter the Indicator must be zero (0).
Month	This parameter lets you set which month of a year (see Year) you want to produce the calendar for. Since this is a calendar specific parameter the Indicator must be zero (0).
Title	This parameter allows you to set the text that will appear in the Header section of the calendar (typically the month name and year). Since this is a calendar specific parameter the Indicator must be zero

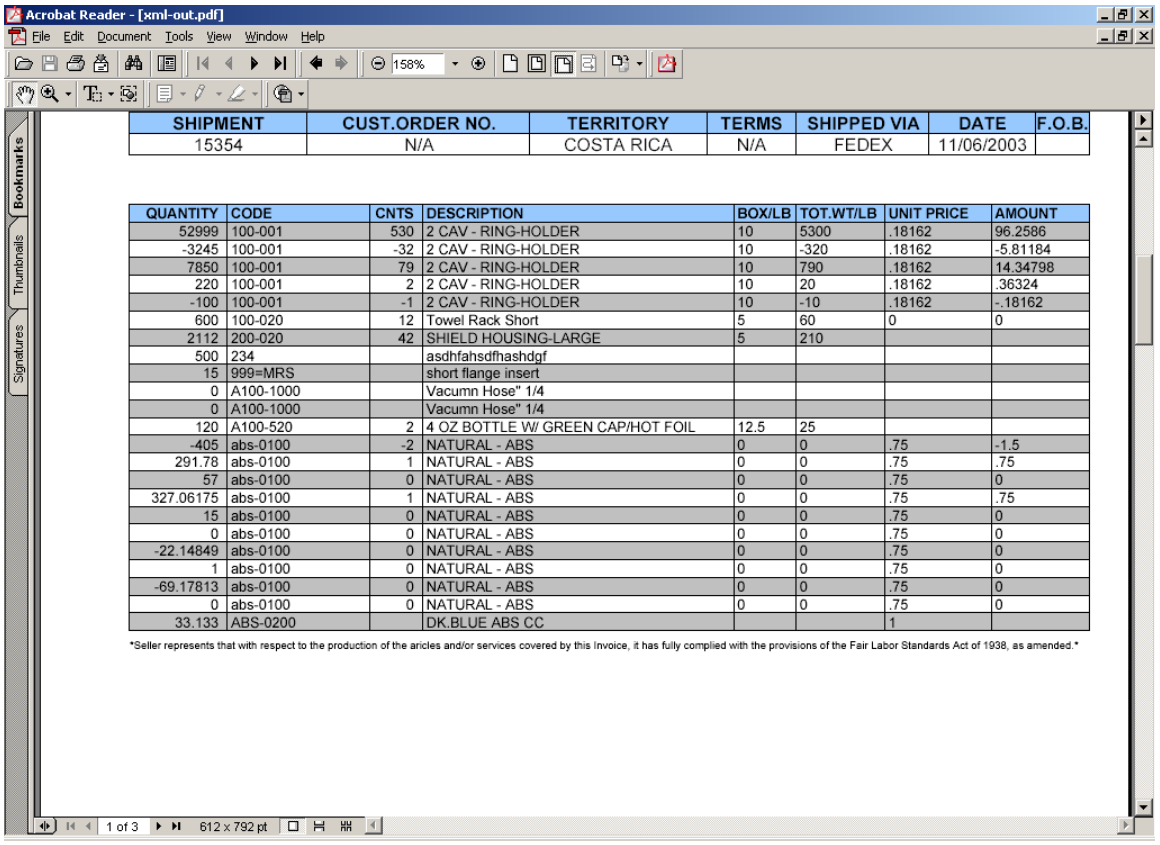
	(0).
Weekdays	This parameter allows you to set the Day Labels (eg: Sun. Mon. Tue etc) of the calendar tool. If not specified then it is defaulted to “Sun,Mon,Tue,Wed”. This must be a comma-delimited list. Since this is a calendar specific parameter the Indicator must be zero (0).
WeekdayStart	This parameter allows you to specify which day to start the week on. Valid values are 1 (Sunday) through 7 (Saturday). If this value isn’t set then the WeekDayStart default to 1 (Sunday). Since this is a calendar specific parameter the Indicator must be zero (0). Use the Parameter value to set which day to start on. **Note**: If you start your day on a different day that Sunday (day you want it started on Monday) then you must also change the ‘WeekDays’ parameter to correctly reflect the day labels.
Width	This parameter allows you to specify the total Width of the Calendar tool. Since this is a calendar specific parameter the Indicator must be zero (0).
X	This parameter allows you to specify the X Coordinate of the Calendar tool. Since this is a calendar specific parameter the Indicator must be zero (0).
Y	This parameter allows you to specify the Y Coordinate of the Calendar tool. Since this is a calendar specific parameter the Indicator must be zero (0).
Year	This parameter lets you set which year you want to produce the calendar for (used in conjunction with the ‘Month’ parameter). Since this is a calendar specific parameter the Indicator must be zero (0).

For an example of how to implement the CALENDAR tool, check out the *samples/super/calendar-euro.p* procedure.

10.3MATRIX Tool

The Matrix tool allows you to generate a matrix (or table) on a specific page. You can have as many matrices on one page as you like but the matrix definition/usage cannot span a page.

The following illustrates multiple matrices on a single page.


Acrobat Reader - [xml-out.pdf]
File Edit Document Tools View Window Help
158%
Bookmarks
Thumbnails
Signatures
1 of 3
612 x 792 pt

SHIPMENT	CUST.ORDER NO.	TERRITORY	TERMS	SHIPPED VIA	DATE	F.O.B.
15354	N/A	COSTA RICA	N/A	FEDEX	11/06/2003	

QUANTITY	CODE	CNTS	DESCRIPTION	BOX/LB	TOT.WT/LB	UNIT PRICE	AMOUNT
52999	100-001	530	2 CAV - RING-HOLDER	10	5300	.18162	96.2586
-3245	100-001	-32	2 CAV - RING-HOLDER	10	-320	.18162	-5.81184
7850	100-001	79	2 CAV - RING-HOLDER	10	790	.18162	14.34798
220	100-001	2	2 CAV - RING-HOLDER	10	20	.18162	.36324
-100	100-001	-1	2 CAV - RING-HOLDER	10	-10	.18162	-.18162
600	100-020	12	Towel Rack Short	5	60	0	0
2112	200-020	42	SHIELD HOUSING-LARGE	5	210		
500	234		asdhfahsdfhashdgf				
15	999=MRS		short flange insert				
0	A100-1000		Vacumn Hose" 1/4				
0	A100-1000		Vacumn Hose" 1/4				
120	A100-520	2	4 OZ BOTTLE W/ GREEN CAP/HOT FOIL	12.5	25		
-405	abs-0100	-2	NATURAL - ABS	0	0	.75	-1.5
291.78	abs-0100	1	NATURAL - ABS	0	0	.75	.75
57	abs-0100	0	NATURAL - ABS	0	0	.75	0
327.06175	abs-0100	1	NATURAL - ABS	0	0	.75	.75
15	abs-0100	0	NATURAL - ABS	0	0	.75	0
0	abs-0100	0	NATURAL - ABS	0	0	.75	0
-22.14849	abs-0100	0	NATURAL - ABS	0	0	.75	0
1	abs-0100	0	NATURAL - ABS	0	0	.75	0
-69.17813	abs-0100	0	NATURAL - ABS	0	0	.75	0
0	abs-0100	0	NATURAL - ABS	0	0	.75	0
33.133	ABS-0200		DK.BLUE ABS CC			1	

Seller represents that with respect to the production of the aricles and/or services covered by this Invoice, it has fully complied with the provisions of the Fair Labor Standards Act of 1938, as amended.

The Matrix tool has a number of parameters that can be set. Those parameters are outlined below.

Parameter Name	Description
BGColor	This parameter allows you to define the background colour for each row of the matrix. The Indicator is used to specify which row you want to apply the background colour to. If the Indicator is zero (0) then the colour is applied as a default to each row of the matrix.
CellValue	This parameter lets you specify the value that will appear in each cell of the matrix. The Indicator represents the cell number where the Cell Value will appear. The cell number is determined by starting at the first possible cell of the matrix (cell number = one) and incrementing by one for each column and row. eg: The following matrix has 3 rows and 5 columns. The cell number is shown in the cell.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

ColumnAlign	This parameter lets you specify the cell value alignment for a column that will appear in the Matrix. The Indicator must represent the column to which you are referring to. If this value isn't set for a particular column the 'LEFT" alignment is assumed. Possible values are: LEFT, RIGHT, CENTER
Columns	This parameter lets you specify the maximum number of columns that should appear in the Matrix. Since this is a matrix specific parameter the Indicator must be zero (0).
ColumnWidth	This parameter lets you specify the width of each column that will appear in the Matrix. The Indicator must represent the column to which you are referring to.
FGColor	The Indicator is used to specify which row you want to apply the text colour to. If the Indicator is zero (0) then the colour is applied as a default to each row of the matrix.
Font	This parameter allows you to define the font for each row of the matrix. If not set for a row, then the current Font (see pdf_Font) for the PDF stream is used. The Indicator is used to specify which row you want to apply the font to.
FontSize	This parameter allows you to define the size of the font for each row of the matrix. If not set for a row, then the current PointSize (see pdf_PointSize) for the PDF stream is used. The Indicator is used to specify which row you want to apply the font sizing to.
GridColor	The parameter allows you to specify the colour of the grid line. Since this is a matrix specific parameter the Indicator must be zero (0).
GridWeight	This parameter allows you to specify the weight of the line to be drawn around each matrix cell. Since this is a matrix specific parameter the Indicator must be zero (0).
Rows	This parameter lets you specify the maximum number of rows that should appear in the Matrix. Since this is a matrix specific parameter the Indicator must be zero (0).
X	This parameter lets you set the starting X coordinate of the Matrix.

	Since this is a matrix specific parameter the Indicator must be zero (0).
Y	This parameter lets you set the starting Y coordinate of the Matrix. Since this is a matrix specific parameter the Indicator must be zero (0).

For an example of how to implement the MATRIX tool, check out the *samples/super/xml.p* procedure.

11.Introducing Complex Forms

Typically the phrases 'Progress' and 'Complex Forms' are not used within the same sentence but with the advent of PDFinclude PRO the terms can now be used without laughter (or, at least, without emptying your pocket book).

So, all kidding aside, how does PDFinclude PRO allow me to create 'Complex Forms' from my Progress application?

First off, you need to get the content that will appear on the form and there are three different options that are available for obtaining form content. These are:

- Use existing text files as content
- Create content using PDFinclude PRO
    - Either by static placement of data, Or
    - By 'publishing' data
- Read an XML document

Secondly, you need the ability to define the graphical template that the content will be placed onto. There are two options available for doing this and those are:

- Programmatically (or statically) place your graphical elements
- Build your template using MS Word and then print it through Adobe Distiller to generate a blank PDF template. Then you can use that template from within PDFinclude PRO.

Each combination of content/template is useful depending on your circumstances. The following table outlines the combinations and defines when they would be useful.

Content Type	Graphic Form Type	Practical Use
Existing Text File	Programmatically Generated	For either combination, this would typically be used when you don't have the ability (or need) to modify the source output but you do want to add graphical elements to your output. This would be a post-generation process.
	PDF Template	

PDFinclude PRO Generated Content – Static Placement **Note:** Both combinations produce a document during the actual process. That is, no post-processing is required on the document. These combinations also require program changes.	Programmatically Generated	This combination would be used if you want to create a static document that you don't want the user to modify. It also disallows them from modifying the placement of the content ... although you could allow for dynamic placement/usage of elements if you accommodated this in your application (eg: you let them define what their logo will be).
	PDF Template	This combination is useful if you want the end user to control the look-and-feel of the document … but they can't control the placement of the data. This could be used to allow the end user to change the look-and-feel of the graphical elements (like changing the logo) without changing the actual data content placement.
PDFinclude PRO Generated Content – Published Data **Note:** Both combinations produce a document during the actual process. That is, no post-processing is required on the document. These combinations also require program changes.	Programmatically Generated	This combination actually won't produce any data content. It will use whatever form you have generated but no data will appear since the published data method uses the pdf_fill_text procedure which performs no data placement at all. Consider this a useless combination.
	PDF Template	This combination offers the most flexibility when producing complex forms. It gives the end user the ability to create their own forms, define where data will be placed and how it will appear. From the application perspective, you don't have to place data, instead you publish an element name (eg: InvoiceNumber) which the end user can then use in their form definition.
Read XML Document **Note:** When reading from an XML Document you can either perform static placement or 'publish' data elements.	Programmatically Generated PDF Template	These combination are similar to the 'PDFinclude PRO Generated Content' methods, it's just that the data elements are coming from an XML file as opposed to the database. The data elements can either be statically placed or published.

Each method has its pluses and minuses and each method allows for differing control and definition of the form's complexity.

In this section we are going to discuss the “PDFinclude PRO Generated Content – Published Data to a PDF Template”. This option offers the most functionality when creating dynamic complex forms.

11.1Publishing Data to a PDF Template

Typically, the people who design forms aren’t necessarily the people who fill the forms with data. Or they don’t associate the data with the form since the data comes from an application they use.

As application developers (or users) it would be nice to give the business user the ability to define their own form, use the data from our DB, and not have to bother us to do so.

Sounds nice. But how would you implement this into your application?

There are many tools available for doing this.

- Some costs lots of money. Some don’t.
- Some offer lots of functionality (PDFinclude PRO has lots of functionality – including scripting, templating, compression and encryption). Others don’t.
- Some are easily integrated (PDFinclude PRO is written in the Progress 4GL. How easy is that?). Others are a pain.

Anyway, were not here to discuss them. We’re here to discuss PDFinclude PRO and how it can work for you. The following subsections will hopefully illustrate the publishing/template functionality enough that you will run to your boss and say ‘we gotta do this … it’s easy for us, it will get the end users off our backs, and it will create awesome forms’.

11.1.1Overview

In your typical Progress program you will not only design the form (using FORM statement, Control Characters etc) but you will also statically place the data elements to specific coordinates (using DISPLAY, PUT etc). So not only have you pigeon-holed yourself into static data placement but you need to use pre-printed forms to get a visually appealing form (with your logo, shading, different fonts, colours etc). And, of course, the placement of information may change with each end user and/or form redesign.

To overcome these limitations (static placement and pre-printed forms) you can use PDFinclude PRO to ‘publish’ your data elements then you can also ‘use’ a pre-designed form to place those data elements onto.

‘Publishing’ data elements allows you to separate the data from the form and from the placement. And by being able to ‘use’ a pre-designed form (with colouring, images etc) you remove the need for pre-printed forms (say ‘cost savings’).

So how do we tie the published data to the pre-defined form? Basically, the pre-defined form will have placeholders (Adobe Form Fields) that the data will be plugged into. With these placeholders you have the ability to dynamically position them plus you can modify the font it uses (so you could display the Invoice Number as a BarCode), the Font Size and the Font Colour. With this you can give the entire form design and data placement process back to the end user. And you don’t have to change any of your code … even if you have multiple clients with multiple languages (of course, you need to be able to link the process to the document it is to produce. This will be application specific and you may need to add some additional DB fields to handle this portion).

Now, that I’ve confused you let me try and make it a little clearer. The following subsections outline what has to get done to accommodate this functionality.

11.1.2Modify Your Process(es)

Since your current process is probably generating the form, you need to modify your process not to perform any output. Instead you need to:

- Tell the process which form to use
- ‘Publish’ the data
- Produce the merged document (merging the template and the data).

A simple example of an Invoice generation routine is show below (assume that all the detail lines will fit on one page):

```
{ pdf_inc.i}

DEFINE VARIABLE i_LineCounter AS INTEGER NO-UNDO.

DEFINE VARIABLE dec_Shipping  AS DECIMAL NO-UNDO.
DEFINE VARIABLE dec_SubTotal  AS DECIMAL NO-UNDO.

RUN pdf_new ("Spdf","\InvoiceDoc-Fill.pdf").

/* Set the PageFooter Routine - this is used to load subtotal/total
figures */
pdf_PageFooter ("Spdf",
                THIS-PROCEDURE:HANDLE,
                "PageFooter").

RUN pdf_open_PDF("Spdf","InvoiceDoc.pdf","Inv"). /* Open Template PDF
*/

/* Here is my Invoice logic */
FOR EACH Invoice WHERE Invoice.CustNum = 81 NO-LOCK,
    FIRST Order WHERE Order.OrderNum = Invoice.OrderNum
    NO-LOCK:

    RUN DoNewPage.

    ASSIGN i_LineCounter = 0
           dec_SubTotal  = 0
           dec_Shipping  = Invoice.ShipCharge.

    FOR EACH OrderLine OF Order.
      I_LineCounter = i_LineCounter + 1.
      RUN DoOrderLine.
    END.
  END.

RUN pdf_close("Spdf").

/* ------------------ INTERNAL PROCEDURES ------------------ */
```

```
PROCEDURE DoNewPage:
  RUN pdf_new_page("Spdf").
  RUN pdf_use_PDF_page("Spdf","Inv",1).  /* Use Page 1 from Template */

  RUN pdf_fill_text("Spdf",
                    "InvoiceNumber",
                    STRING(Invoice.InvoiceNum,"99999"),
                    "").

  /* Set TODAYS date */
  RUN pdf_fill_text("Spdf", "Today", STRING(TODAY,"99/99/99"), "").

  /* Set Company Info */
  RUN pdf_fill_text("Spdf", "CompanyName","Acme", "").
  RUN pdf_fill_text("Spdf", "CompanyPhone","(123) 555-1234", "").
  RUN pdf_fill_text("Spdf", "CompanyFax","(123) 555-4321", "").
  RUN pdf_fill_text("Spdf", "CompanyEmail","test@acme.com", "").
  RUN pdf_fill_text("Spdf", "CompanyWeb","www.acme.com", "").

  /* Get Customer Info */
  FIND Customer WHERE Customer.CustNum = Invoice.CustNum
       NO-LOCK NO-ERROR.
  RUN pdf_fill_text("Spdf",
                    "CustomerID",
                     STRING(Customer.CustNum,"99999"),
                     "").

  /* Display 'To' information */
  RUN pdf_fill_text("Spdf","BillTo_Name",Customer.Name,"").
  RUN pdf_fill_text("Spdf","BillTo_Address1",Customer.Address,"").
  RUN pdf_fill_text("Spdf","BillTo_Address2",Customer.Address2,"").

  /* Display 'Ship To' Information - same as Customer for now */
  RUN pdf_fill_text("Spdf","ShipTo_Name",Customer.Name,"").
  RUN pdf_fill_text("Spdf","ShipTo_Address1",Customer.Address,"").
  RUN pdf_fill_text("Spdf","ShipTo_Address2",Customer.Address2,"").

  RUN pdf_fill_text("Spdf",
                    "InvoiceDate",
                    STRING(Invoice.InvoiceDate,"99/99/99"),
                    "").
  RUN pdf_fill_text("Spdf",
                    "Shipping",
                    STRING(Invoice.ShipCharge,">,>>>,>>9.99"),
                    "").

  RUN pdf_fill_text("Spdf","Terms",Order.Terms,"").
  RUN pdf_fill_text("Spdf",
                    "OrderNumber",
                    STRING(Order.OrderNum,"999999"),
                    "").
  RUN pdf_fill_text("Spdf","SalesRep",Order.SalesRep,"").
  RUN pdf_fill_text("Spdf","FOB",Order.WarehouseNum,"").

END.

PROCEDURE PageFooter:
  RUN pdf_fill_text("Spdf",
                    "SubTotal",
                    STRING(dec_SubTotal,">,>>>,>>9.99"),
                    "align=right").
  RUN pdf_fill_text("Spdf",
                    "Total",
                    STRING(dec_SubTotal,">,>>>,>>9.99"),
                    "align=right").

  RUN pdf_fill_text("Spdf",
                    "Balance",
                    STRING(dec_SubTotal + dec_Shipping,">,>>>,>>9.99"),
                    "align=center").

END.

PROCEDURE DoOrderLine:

  RUN pdf_fill_text("Spdf",
                    "Quantity_" + STRING(i_LineCounter),
                    STRING(OrderLine.Qty,">,>>>,>>9.99"),
                    "align=right").
  RUN pdf_fill_text("Spdf",
                    "Item_" + STRING(i_LineCounter),
                    OrderLine.ItemNum,
                    "").

  RUN pdf_fill_text("Spdf",
                    "Units_" + STRING(i_LineCounter),
                    "EACH",
                    "").

  FIND Item WHERE Item.ItemNum = OrderLine.ItemNum NO-LOCK NO-ERROR.
  RUN pdf_fill_text("Spdf",
                    "Description_" + STRING(i_LineCounter),
                    Item.ItemName,
                    "").

  RUN pdf_fill_text("Spdf",
                    "Discount_" + STRING(i_LineCounter),
                    STRING(OrderLine.Discount,">,>>>,>>9.99"),
                    "align=right ").

  RUN pdf_fill_text("Spdf",
                    "UnitPrice_" + STRING(i_LineCounter),
                    STRING(Item.Price,">,>>>,>>9.99"),
                    "align=right ").

  RUN pdf_fill_text("Spdf",
                    "Total_" + STRING(i_LineCounter),
                    STRING((Item.Price * OrderLine.Qty)
                           - OrderLine.Discount,">,>>>,>>9.99"),
                    "align=right").

  ASSIGN dec_SubTotal = dec_SubTotal
                      + (Item.Price * OrderLine.Qty)
                      - OrderLine.Discount.

END.
```

In the preceding example, you will see that nowhere does it say where to place data. Using the *pdf_fill_text* procedure we are telling the procedure to publish the data to PDFinclude PRO which will merge the data with the template (InvoiceDoc.pdf) to produce InvoiceDoc-Fill.pdf.

Now how do you create a template? The next section outlines this process.

11.1.3Create A Template

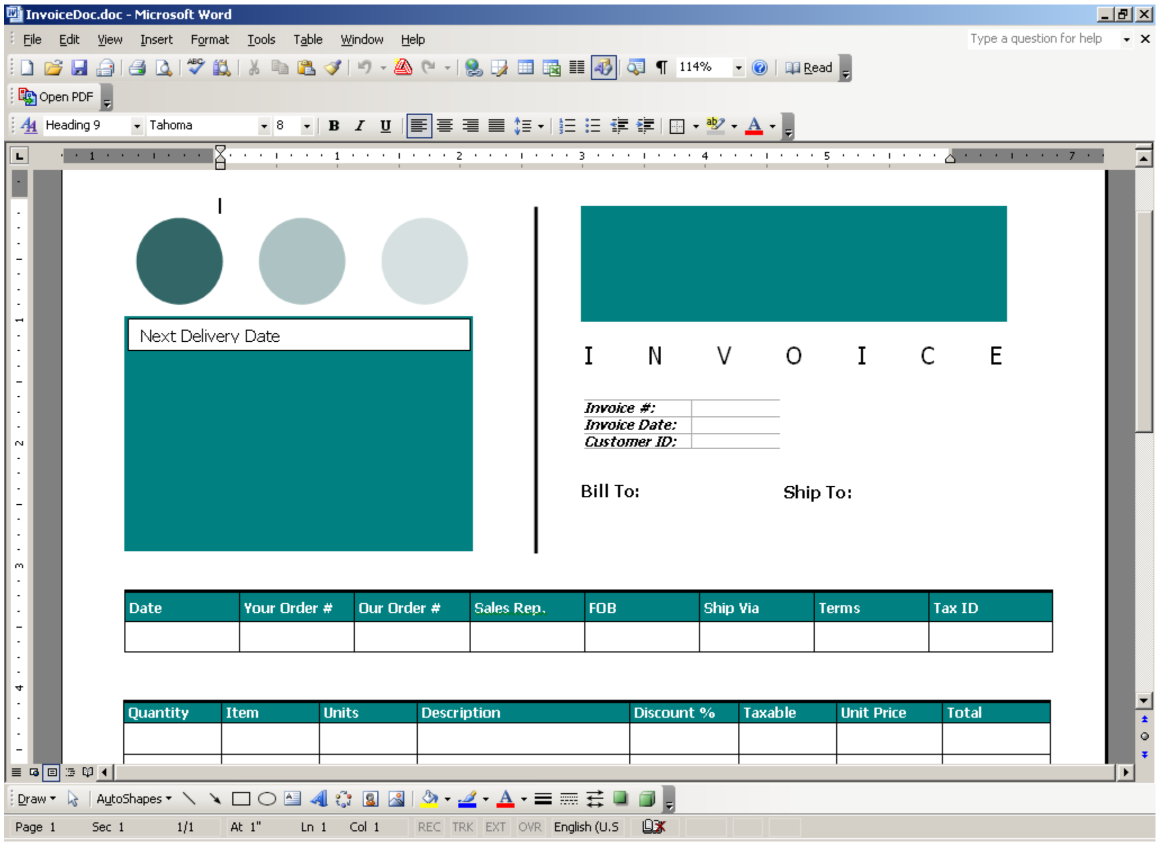
You, or your end user, can create the template in whichever language(s) they require. The process is fairly simple and is outlined below.

Step 0:

Purchase Adobe Professional (www.Adobe.com) or utilize OpenOffice (www.OpenOffice.org). The following steps outline the process when utilizing Adobe Professional.

Step 1:

Using your favourite productivity tool (eg: MS Word) create your blank template. See example below:


InvoiceDoc.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Type a question for help
114%
Read
Open PDF
Heading 9
Tahoma
8
Draw
AutoShapes
Page 1 Sec 1 1/1 At 1" Ln 1 Col 1 REC TRK EXT OVR English (U.S

Next Delivery Date

I N V O I C E

Invoice #:	
Invoice Date:	
Customer ID:	

Bill To:

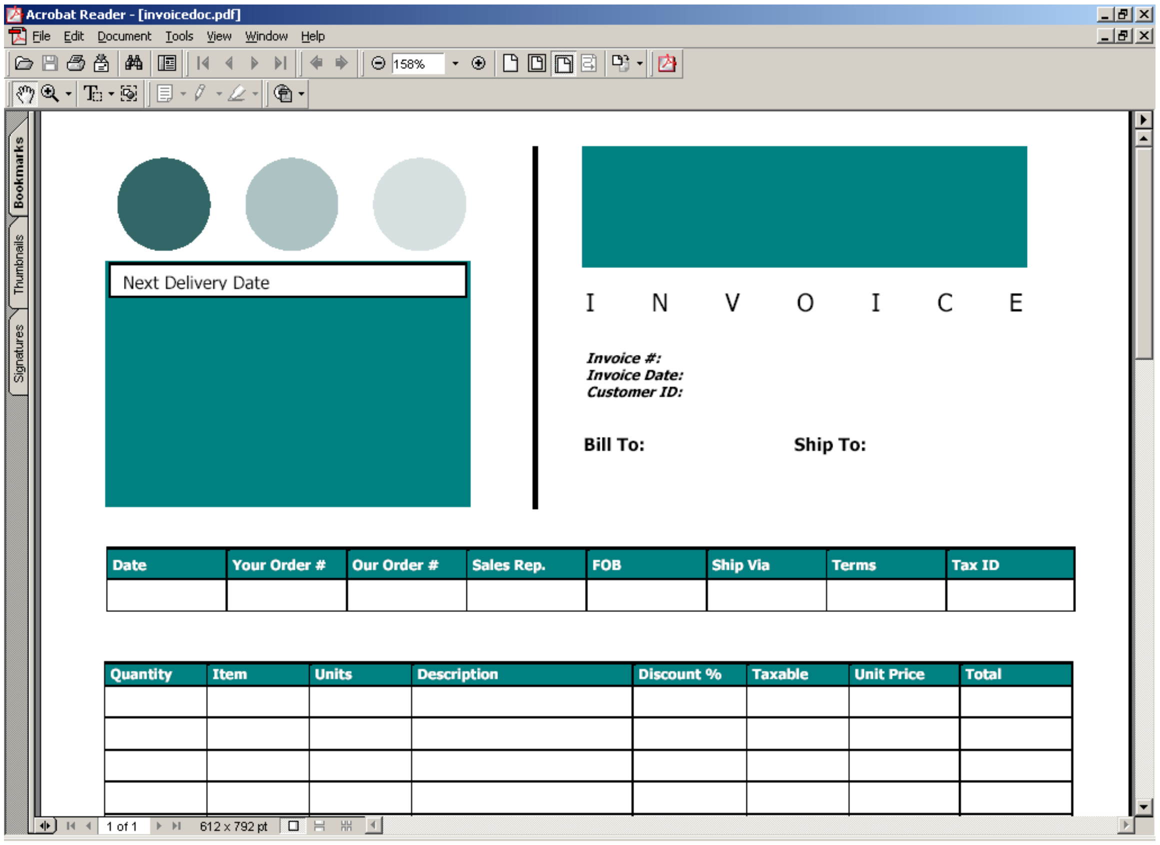
Ship To:

Date	Your Order #	Our Order #	Sales Rep.	FOB	Ship Via	Terms	Tax ID

Quantity	Item	Units	Description	Discount %	Taxable	Unit Price	Total

Step 2:

Once your document is complete, print it through Adobe Distiller to create a blank PDF document. See example below.


Acrobat Reader - [invoicedoc.pdf]
File Edit Document Tools View Window Help
158%
Bookmarks
Thumbnails
Signatures
1 of 1
612 x 792 pt

Next Delivery Date

I N V O I C E

Invoice #:
Invoice Date:
Customer ID:

Bill To: **Ship To:**

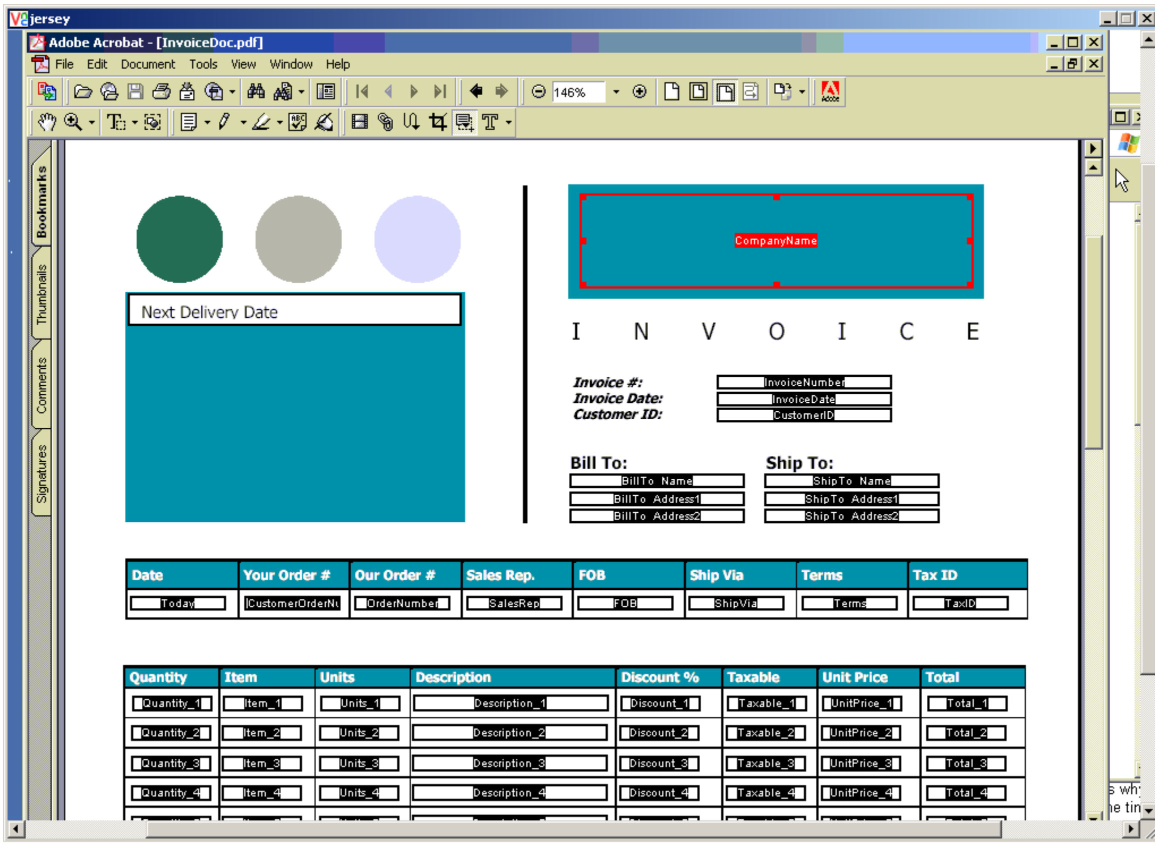
Date	Your Order #	Our Order #	Sales Rep.	FOB	Ship Via	Terms	Tax ID

Quantity	Item	Units	Description	Discount %	Taxable	Unit Price	Total

This has created a blank PDF document with no Form Fields.

Step 3:

Using Adobe Acrobat Professional you need to add your Form Fields.


jersey
Adobe Acrobat - [InvoiceDoc.pdf]
File Edit Document Tools View Window Help
146%
Bookmarks
Thumbnails
Comments
Signatures

Next Delivery Date

CompanyName

I N V O I C E

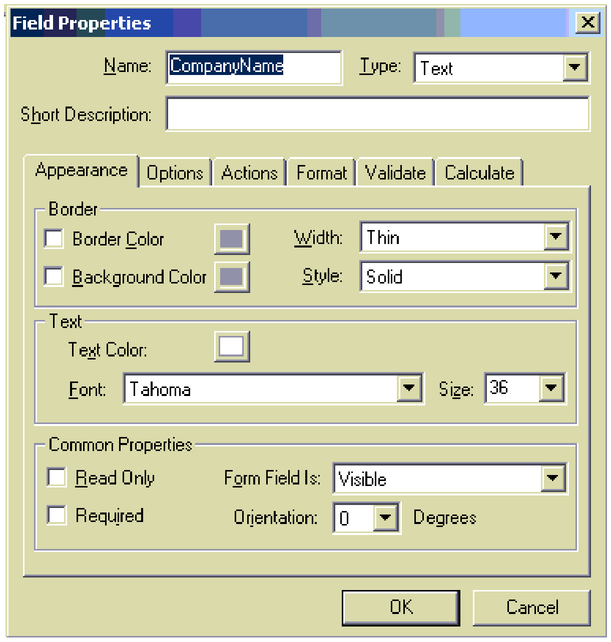
Invoice #:	InvoiceNumber
Invoice Date:	InvoiceDate
Customer ID:	CustomerID

Bill To:	Ship To:
BillTo Name	ShipTo Name
BillTo Address1	ShipTo Address1
BillTo Address2	ShipTo Address2

Date	Your Order #	Our Order #	Sales Rep.	FOB	Ship Via	Terms	Tax ID
Today	CustomerOrderNu	OrderNumber	SalesRep	FOB	ShipVia	Terms	TaxID

Quantity	Item	Units	Description	Discount %	Taxable	Unit Price	Total
Quantity_1	Item_1	Units_1	Description_1	Discount_1	Taxable_1	UnitPrice_1	Total_1
Quantity_2	Item_2	Units_2	Description_2	Discount_2	Taxable_2	UnitPrice_2	Total_2
Quantity_3	Item_3	Units_3	Description_3	Discount_3	Taxable_3	UnitPrice_3	Total_3
Quantity_4	Item_4	Units_4	Description_4	Discount_4	Taxable_4	UnitPrice_4	Total_4

The above example illustrates field placement using Adobe Form Fields. You can have as many fields as you like and you can control the placement of the fields. For each Form Field defined you can specify some properties.



Name: CompanyName
Type: Text
Short Description:

Border
- [ ] Border Color
- [ ] Background Color
- Width: Thin
- Style: Solid

Text
- Text Color:
- Font: Tahoma
- Size: 36

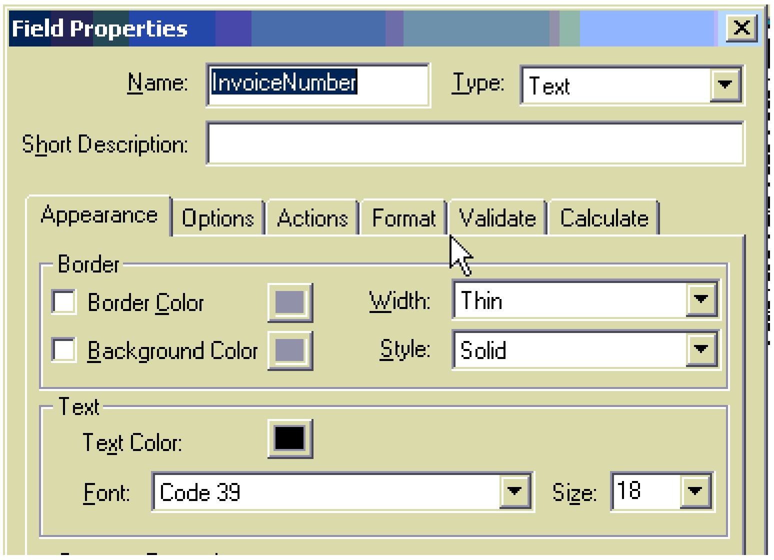
Common Properties
- [ ] Read Only
- [ ] Required
- Form Field Is: Visible
- Orientation: 0 Degrees

The only properties that are utilized by PDFinclude PRO are:

- Name
- Type (must be 'Text')
- Text Color
- Font
- Size

All other properties are ignored. **NOTE**: **Do not use periods in the FieldName.**

If you wanted a field printed with a BarCode font, then this is where you'd set it up to do so. For example:



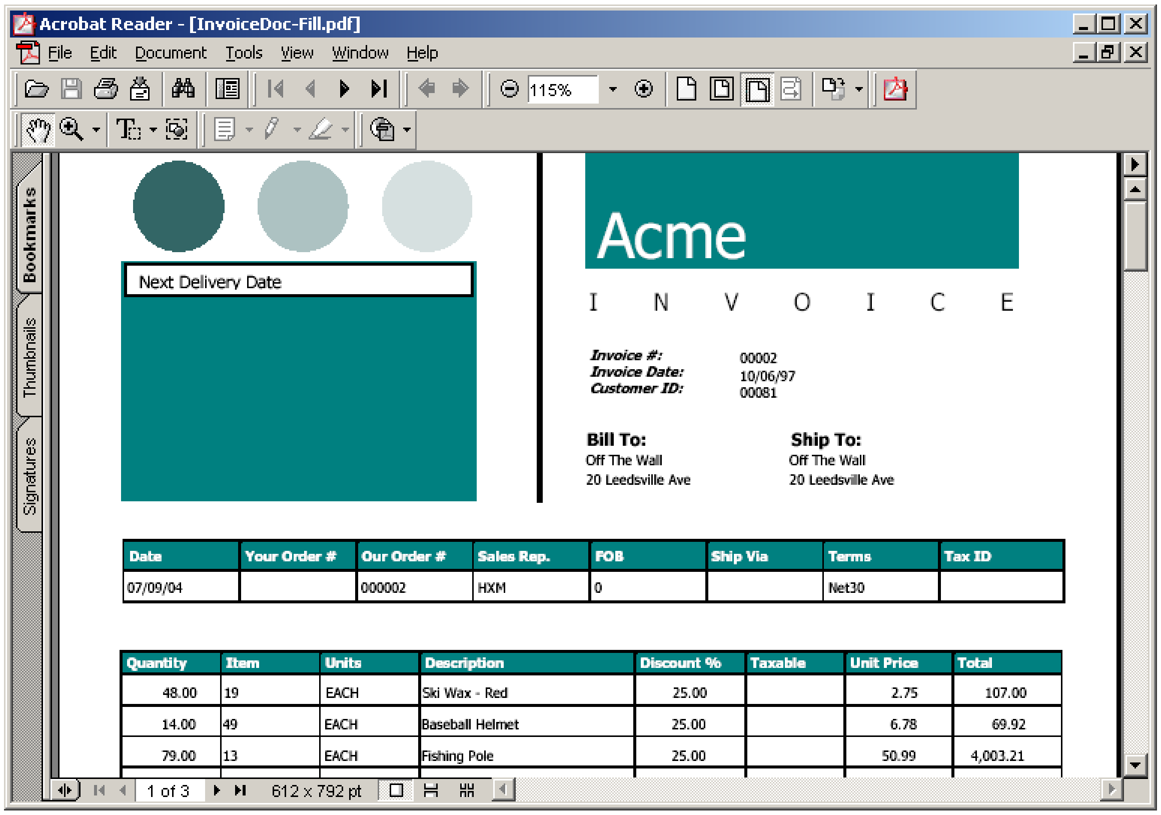
Name: InvoiceNumber
Type: Text
Short Description:
Border Color
Background Color
Width: Thin
Style: Solid
Text Color:
Font: Code 39
Size: 18

The Form Field called InvoiceNumber is to be printed as Size 18 with the Code 39 font that I have installed on my local machine (but it can be used anywhere as the fonts are embedded into the PDF document). You can use the same Form Field name as many times as you want on the same page, controlling how the appearance/placement will be for each of them (so have one instance of InvoiceNumber as a BarCode while another instance would just show the numeric value).

Once you've completed your form and field placement (which may take multiple interations), it's best to use the 'Save As' option when creating the working template. This is due to the fact that for each time you use the 'Save' option Adobe saves the previous object version within itself. This greatly increases the number of objects that PDFinclude PRO has to parse plus it also increases the template file size. Using 'Save As' creates a fresh copy of the file.

That's it. With the combination of the programming required (as illustrated in the previous section) and the template generated you can produce complex, and visually appealing forms, from within your Progress application.

The following image illustrates the resulting merge document.



Next Delivery Date

Acme

I N V O I C E

Invoice #:	00002
Invoice Date:	10/06/97
Customer ID:	00081

Bill To:
Off The Wall
20 Leedsville Ave

Ship To:
Off The Wall
20 Leedsville Ave

Date	Your Order #	Our Order #	Sales Rep.	FOB	Ship Via	Terms	Tax ID
07/09/04		000002	HXM	0		Net30	

Quantity	Item	Units	Description	Discount %	Taxable	Unit Price	Total
48.00	19	EACH	Ski Wax - Red	25.00		2.75	107.00
14.00	49	EACH	Baseball Helmet	25.00		6.78	69.92
79.00	13	EACH	Fishing Pole	25.00		50.99	4,003.21

12.Sports2000 Samples

This sections gives a couple of sample procedures to run against the Sports2000 database that is typically supplied with the Progress Installation.

The Samples assume that:

1. You are connected to the Sports2000 database
2. You have *pdf_inc.i* in your PROPATH
3. You have Acrobat Reader (or some other PDF viewer) installed on you machine
4. You have the free Code39 BarCode downloaded from www.barcodetrader.com.
5. You have created the code39.afm file using *ttf2pt1* (See Appendix B)
6. You will need to change the directories used for file storage and retrieval (eg: the samples use c:\sports2000 you may use another directory for testing).

12.1Hello World

The following sample produces a minimal PDF document. The output file is included in the Zipped download and is called hello.pdf.

```
{ ./pdf_inc.i }
RUN pdf_new ("Spdf","c:\sports2000\PDFinc\hello.pdf").
RUN pdf_new_page("Spdf").
RUN pdf_text    ("Spdf", "Hello World").
RUN pdf_close("Spdf").
```

12.2Hello World – Compressed and Encrypted

The following sample produces a minimal PDF document that is both compressed and encrypted.

```
{ ./pdf_inc.i }
RUN pdf_new ("Spdf","c:\sports2000\PDFinc\hello-encrypt.pdf").
RUN pdf_new_page("Spdf").
RUN pdf_set_parameter("Spdf",”Encrypt”,”TRUE”).
RUN pdf_set_parameter("Spdf",”Compress”,”TRUE”).
RUN pdf_text    ("Spdf", "Hello World").
RUN pdf_close("Spdf").
```

12.3Text Positioning

The following sample produces a PDF document displaying the different type of Text options available. The output file is included in the Zipped download and is called text.pdf.

```
{ ./pdf_inc.i }


RUN pdf_new ("Spdf","c:\sports2000\PDFinc\text.pdf").
RUN pdf_new_page("Spdf").

/* Place some text */
RUN pdf_text    ("Spdf", FILL("123456789 ",8)).
RUN pdf_skip    ("Spdf").
RUN pdf_text_at ("Spdf", "Position AT Column 10",10).
RUN pdf_text_to ("Spdf", "Position TO Column 70",70).
RUN pdf_text    ("Spdf", "Text placed at last X/Y Coordinate").
RUN pdf_skip    ("Spdf").

/* Change the text color to red */
RUN pdf_text_color("Spdf",1.0,.0,.0).

RUN pdf_set_font("Spdf","Courier",14.0).
RUN pdf_text_xy ("Spdf","This is text placed using XY coordinates",100,500).
RUN pdf_set_font("Spdf","Courier",10.0).

/* Change the text color back to black */
RUN pdf_text_color("Spdf",.0,.0,.0).

/* Change the Rectangle border to red and the fill to white */
RUN pdf_stroke_color("Spdf",1.0,.0,.0).
RUN pdf_stroke_fill("Spdf",1.0,1.0,1.0).

/* Display a boxed text string */
RUN pdf_text_boxed_xy ("Spdf",
            "This is BOXED text placed using XY coordinates",
            100,
            450,
            pdf_text_width("Spdf", "This is BOXED text placed using XY coordinates"),
            10,"Left",1).

RUN pdf_close("Spdf").
```

12.4Customer Listing

This sample produces a Customer Listing that illustrates how to create a basic PDF report. The output file is included in the Zipped download and is called custlist.pdf.

Copy the following code into your Editor and run.

```
/* custlist.p - sample PDF generation */

{ ./pdf_inc.i }

DEFINE VARIABLE Vlines      AS INTEGER NO-UNDO.
DEFINE VARIABLE Vpage       AS INTEGER NO-UNDO.
DEFINE VARIABLE Vold_Y      AS INTEGER NO-UNDO.

/* Create stream for new PDF file */
RUN pdf_new        ("Spdf","c:\sports2000\PDFinc\custlist.pdf").

/* Load Bar Code Font */
RUN pdf_load_font  ("Spdf","Code39","c:\windows\fonts\code39.ttf","c:\sports2000\code39.afm").

/* Load PRO-SYS Logo File */
RUN pdf_load_image ("Spdf","ProSysLogo","c:\sports2000\pdfinc\prosyslogo.jpg",1354,166).

/* Set Document Information */
RUN pdf_set_info("Spdf","Author","Gordon Campbell").
RUN pdf_set_info("Spdf","Subject","Accounts Receivable").
RUN pdf_set_info("Spdf","Title","Customer List").
RUN pdf_set_info("Spdf","Keywords","Customer List Accounts Receivable").
RUN pdf_set_info("Spdf","Creator","PDFinclude").
RUN pdf_set_info("Spdf","Producer","custlist.p").

/* Instantiate a New Page */
RUN new_page.

/* Loop through appropriate record set */
FOR EACH customer WHERE NAME BEGINS "AB" NO-LOCK
    BREAK BY State
          BY CustNum:

  /* Output the appropriate Record information */
RUN  pdf_set_font ("Spdf","Courier-Oblique",10.0).
RUN  pdf_text_at  ("Spdf", customer.state,1).
RUN  pdf_set_font ("Spdf","Courier",10.0).
RUN  pdf_text_at  ("Spdf", STRING(customer.CustNum,">>>9"),6).
RUN  pdf_text_at  ("Spdf", customer.NAME,12).
RUN  pdf_text_at  ("Spdf", customer.phone,44).
RUN  pdf_text_to  ("Spdf", STRING(customer.balance),80).

  /* Display a BarCode for each Customer Number */
RUN  pdf_set_font ("Spdf","Code39",10.0).

  Vold_Y = pdf_TextY("Spdf").  /* Store Original Text Path */
RUN  pdf_text_xy ("Spdf", STRING(customer.CustNum,"999999"),500, pdf_textY("Spdf")).
RUN  pdf_set_TextY("Spdf",Vold_y). /* Reset Original Text Path */
RUN  pdf_set_font ("Spdf","Courier",10.0).

  /* Skip to Next Line */
RUN  pdf_skip    ("Spdf").

  /* Process Line Counter */
  Vlines = Vlines + 1.
  IF Vlines = 60 THEN
    RUN new_page.

  IF LAST-OF(State) THEN DO:
    IF Vlines + 2 > 60 THEN
      RUN new_page.
    /* Put a red line between each of the states */
   RUN  pdf_skip ("Spdf").
   RUN  pdf_stroke_color ("Spdf",1.0,.0,.0).
   RUN pdf_set_dash ("Spdf",2,2).
   RUN pdf_line  ("Spdf", pdf_LeftMargin("Spdf") , pdf_TextY("Spdf") + 5, pdf_PageWidth("Spdf") - 20 ,
pdf_TextY("Spdf") + 5, 1).
    pdf_stroke_color ("Spdf",.0,.0,.0).
    pdf_skip    ("Spdf").

    Vlines = Vlines + 2.
  END. /* Last-of State */
END.

RUN end_of_report.

RUN pdf_close("Spdf").

/* -------------------- INTERNAL PROCEDURES -------------------------- */

PROCEDURE new_page:

  /* Display Footer UnderLine */
  IF Vpage > 0 THEN
    RUN page_footer.

  RUN  pdf_new_page("Spdf").

  Vpage = Vpage + 1.

  /* Place Logo but only on first page of Report */
  IF Vpage = 1 THEN DO:
   RUN  pdf_place_image ("Spdf","ProSysLogo",pdf_LeftMargin("Spdf"), pdf_TopMargin("Spdf") - 20
,179,20).
  END.

  /* Set Header Font Size and Colour */
  RUN pdf_set_font ("Spdf","Courier-Bold",10.0).
  RUN pdf_text_color ("Spdf",1.0,.0,.0).

  /* Put a Rectangle around the Header */
  RUN pdf_stroke_color ("Spdf", .0,.0,.0).
  RUN pdf_stroke_fill ("Spdf", .9,.9,.9).
  RUN pdf_rect ("Spdf", pdf_LeftMargin("Spdf"), pdf_TextY("Spdf") - 3, pdf_PageWidth("Spdf") - 30  ,
pdf_TextX("Spdf") + 12).

  /* Output Report Header */
  RUN pdf_text_at  ("Spdf","St",1).
  RUN pdf_text_at  ("Spdf","Nbr",6).
  RUN pdf_text_at  ("Spdf","Customer Name",12).
  RUN pdf_text_at  ("Spdf","Phone Number",44).
  RUN pdf_text_to  ("Spdf","Balance",80).

  /* Display Header UnderLine */
  RUN pdf_skip ("Spdf").
  RUN pdf_set_dash ("Spdf",1,0).
  RUN pdf_line  ("Spdf", pdf_LeftMargin("Spdf"), pdf_TextY("Spdf") + 5, pdf_PageWidth("Spdf") - 20 ,
pdf_TextY("Spdf") + 5, 2).
  RUN pdf_skip ("Spdf").

  /* Set Detail Font Colour */
  RUN pdf_text_color ("Spdf",.0,.0,.0).
  Vlines = 3.
END. /* new_page */

PROCEDURE page_footer:
  RUN pdf_skip ("Spdf").
  RUN pdf_set_dash ("Spdf",1,0).
  RUN pdf_line  ("Spdf", 0, pdf_TextY("Spdf") - 5, pdf_PageWidth("Spdf") - 20 , pdf_TextY("Spdf") - 5, 2).
  RUN pdf_skip ("Spdf").
  RUN pdf_skip ("Spdf").
  RUN pdf_text_to  ("Spdf","Page: " + STRING(Vpage), 97).
END. /* page_footer */

PROCEDURE end_of_report:
  RUN page_footer.

  /* Display Footer UnderLine and End of Report Tag (Centered) */
  RUN pdf_skip ("Spdf").
  RUN pdf_stroke_fill ("Spdf",1.0,1.0,1.0).
  RUN pdf_text_boxed_xy ("Spdf", "End of Report", 250, pdf_TextY("Spdf") - 20,
pdf_Text_Width("Spdf","End Of Report"), 12, "Left",1).
END. /* end_of_report */

/* end of custlist.p */
```

12.5Formatted Item Listing

This sample produces a Item Listing that illustrates how you can control the output of a fairly custom PDF report. The output file is included in the Zipped download and is called itemlist.pdf.

Copy the following code into your Editor and run.

```
/* itemlist.p - sample PDF generation */

{ ./pdf_inc.i }

DEFINE VARIABLE Vitems      AS INTEGER NO-UNDO.
DEFINE VARIABLE Vrow        AS INTEGER NO-UNDO.
DEFINE VARIABLE Vcat-desc   AS CHARACTER EXTENT 4 FORMAT "X(40)" NO-UNDO.

/* Create stream for new PDF file */
RUN pdf_new        ("Spdf","c:\sports2000\PDFinc\itemlist.pdf").

/* Load Bar Code Font */
RUN pdf_load_font  ("Spdf","Code39","c:\windows\fonts\code39.ttf","c:\sports2000\code39.afm").

/* Set Document Information */
RUN pdf_set_info("Spdf","Author","Gordon Campbell").
RUN pdf_set_info("Spdf","Subject","Inventory").
RUN pdf_set_info("Spdf","Title","Item Catalog").
RUN pdf_set_info("Spdf","Keywords","Item Catalog Inventory").
RUN pdf_set_info("Spdf","Creator","PDFinclude").
RUN pdf_set_info("Spdf","Producer","itemlist.p").

/* Instantiate a New Page */
RUN new_page.

/* Loop through appropriate record set */
FOR EACH ITEM NO-LOCK
    BREAK BY ItemNum:
```

```
  RUN display_item_info.

  /* Process Record Counter */
  Vitems = Vitems + 1.
  IF Vitems = 6 THEN
    RUN new_page.

END.

RUN pdf_close("Spdf").

/* -------------------- INTERNAL PROCEDURES -------------------------- */

PROCEDURE display_item_info:

  /* Draw main item Box */
  RUN pdf_stroke_fill("Spdf",.8784,.8588,.7098).
  RUN pdf_rect ("Spdf", pdf_LeftMargin("Spdf"), Vrow, pdf_PageWidth("Spdf") - 20  , 110).

  /* Draw Smaller box (beige filled) to contain Category Description */
  RUN pdf_rect ("Spdf", 350, Vrow + 10, 240, 45).

  /* Draw Smaller box (white filled) to contain Item Picture (when avail) */
  RUN pdf_stroke_fill("Spdf",1.0,1.0,1.0).
  RUN pdf_rect ("Spdf", pdf_LeftMargin("Spdf") + 10, Vrow + 10, pdf_LeftMargin("Spdf") + 100  , 90).


  /* Display Labels with Bolded Font */
  RUN pdf_set_font("Spdf","Courier-Bold",10.0).
  RUN pdf_text_xy ("Spdf","Part Number:", 140, Vrow + 90).
  RUN pdf_text_xy ("Spdf","Part Name:", 140, Vrow + 80).
  RUN pdf_text_xy ("Spdf","Category 1:", 140, Vrow + 40).
  RUN pdf_text_xy ("Spdf","Category 2:", 140, Vrow + 30).
  RUN pdf_text_xy ("Spdf","Qty On-Hand:", 350, Vrow + 90).
  RUN pdf_text_xy ("Spdf","Price:", 350, Vrow + 80).
  RUN pdf_text_xy ("Spdf","Category Description:", 350, Vrow + 60).

  /* Display Fields with regular Font */
  RUN pdf_set_font("Spdf","Courier",10.0).
  RUN pdf_text_xy ("Spdf",STRING(item.ItemNuM), 230, Vrow + 90).
  RUN pdf_text_xy ("Spdf",item.ItemName, 230, Vrow + 80).
  RUN pdf_text_xy ("Spdf",item.Category1, 230, Vrow + 40).
  RUN pdf_text_xy ("Spdf",item.Category2, 230, Vrow + 30).
  RUN pdf_text_xy ("Spdf",STRING(item.OnHand), 440, Vrow + 90).
  RUN pdf_text_xy ("Spdf",TRIM(STRING(item.Price,"$>>,>>9.99-")), 440, Vrow + 80).

  /* Now Load and Display the Category Description */
  RUN load_cat_desc.
  RUN pdf_text_xy ("Spdf",Vcat-desc[1], 352, Vrow + 46).
  RUN pdf_text_xy ("Spdf",Vcat-desc[2], 352, Vrow + 36).
  RUN pdf_text_xy ("Spdf",Vcat-desc[3], 352, Vrow + 26).
  RUN pdf_text_xy ("Spdf",Vcat-desc[4], 352, Vrow + 16).

  /* Display text in Image Box */
 RUN  pdf_text_color("Spdf",1.0,.0,.0).
 RUN pdf_text_xy ("Spdf","NO", 40, Vrow + 66).
 RUN  pdf_text_xy ("Spdf","IMAGE", 40, Vrow + 56).
 RUN  pdf_text_xy ("Spdf","AVAILABLE", 40, Vrow + 46).

  RUN pdf_text_color("Spdf",.0,.0,.0).
  /* Display the Product Number as a Bar Code */
  RUN pdf_set_font("Spdf","Code39",14.0).
  RUN pdf_text_xy ("Spdf",STRING(item.ItemNuM,"999999999"), 140, Vrow + 60).

  Vrow = Vrow - 120.
END. /* display_item_info */

PROCEDURE new_page:
  RUN pdf_new_page("Spdf").

  /* Reset Page Positioning etc */
  ASSIGN Vrow   = pdf_PageHeight("Spdf") - pdf_TopMargin("Spdf") - 110
         Vitems = 0.
END. /* new_page */

PROCEDURE load_cat_desc:
  DEFINE VARIABLE L_cat     AS CHARACTER NO-UNDO.

  DEFINE VARIABLE L_loop    AS INTEGER NO-UNDO.
  DEFINE VARIABLE L_extent  AS INTEGER NO-UNDO.

  ASSIGN Vcat-desc = ""
         L_cat      = item.catdescr.
  REPLACE(L_cat,CHR(13),"").
  REPLACE(L_cat,CHR(10),"").

  L_extent = 1.
  DO L_Loop = 1 TO NUM-ENTRIES(L_cat," "):
    IF (LENGTH(Vcat-desc[L_extent]) + LENGTH(ENTRY(L_loop,L_cat," ")) + 1) > 40
    THEN DO:
      IF L_extent = 4 THEN LEAVE.

      L_extent = L_extent + 1.
    END.

    Vcat-desc[L_extent] = Vcat-desc[L_extent] + ENTRY(L_loop,L_cat," ") + " ".
  END.
END. /* load_cat_desc */

/* end of itemlist.p */
```

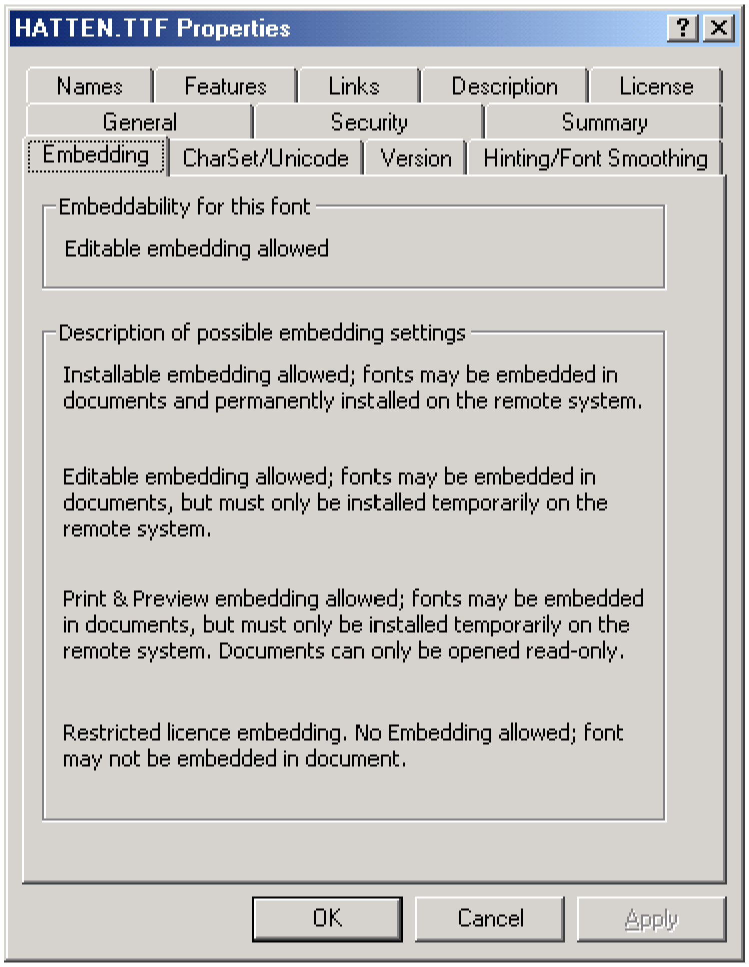
Appendix A – Embedding Fonts

As previously mentioned, you need to be legally responsible when embedding True Type Font files into any documents. Most fonts are embeddable but not distributable. In order to ascertain whether you can legally embed the TTF file you should download the Extended Font Viewer from Microsoft.

The Extended Font Viewer can be obtained via the following link:

http://www.microsoft.com/typography/property/property.htm?fname=%20&fsize=

Once installed you can right-click on a TTF file, select the 'Properties' option and the following screen (or very similar – dependant on Windows version) will appear.



Embeddability for this font

Editable embedding allowed

Description of possible embedding settings

Installable embedding allowed; fonts may be embedded in documents and permanently installed on the remote system.

Editable embedding allowed; fonts may be embedded in documents, but must only be installed temporarily on the remote system.

Print & Preview embedding allowed; fonts may be embedded in documents, but must only be installed temporarily on the remote system. Documents can only be opened read-only.

Restricted licence embedding. No Embedding allowed; font may not be embedded in document.

Click on the "Embedding" tab. Here you will see the license options available for the Font file. Basically, if it says that you have 'Restricted License Embedding" then you cannot legally embed the font file.

PRO-SYS is not legally responsible for any actions you take with embedding font files into your PDF file.

Appendix B – Creating AFM File

The Adobe Font Metrics (AFM) file is used by PDFinclude PRO to determine how the glyphs (or characters) should appear when displayed by a PDF viewer (such as Adobe Acrobat).

The AFM file stores information such as glyph widths, is the font italicized, glyphs defined etc.

PDFinclude PRO does not generate the AFM file for you. Instead you must resort to using a free utility called 'ttf2pt1'. This utility will read a TTF file and generate the AFM file.

To download 'ttf2pt1' go to the following link:

http://ttf2pt1.sourceforge.net/

Or you can download the binary executable for Windows from:

http://www.epro-sys.com/downloads/ttf2pt1.zip

The utility is written in C and must be compiled into binary executable format on whichever platform you are running.

Note: PRO-SYS was unable to compile the 3.4.0 version but we were able to compile the 3.3.5 version.

Once installed and compiled, the process to create the AFM file is very simple. The following command can be used as an example.

ttf2pt1 –A \windows\fonts\arial.ttf arial

The '–A' is important. This option actually indicates to ttfpt1 that it must output an AFM file.

You may wish to read the documentation that comes with the download for a further understanding of what ttf2pt1 is capable of.

PRO-SYS does not resell ttf2pt1 as a component of the PDFinclude PRO package. If you have another tool that generates AFM files then by all means use it.

PRO-SYS does not support ttf2pt1 and is not legally responsible for anything related to the utility.