pdfInclude
for Progress OpenEdge

creation of pdf documents and reports
from within Progress OpenEdge ABL
free from external modules

developed by Snow Wolf Software
snowwolf-software.com

Version 5.1

This document was generated by pdfInclude 5.2.dev in 48.15 seconds.
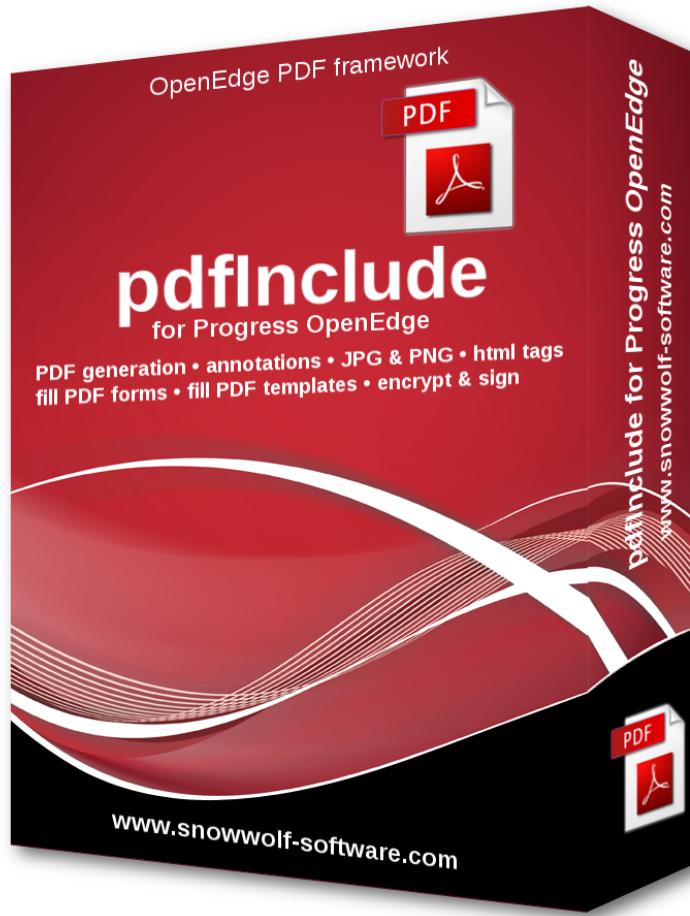
Introduction

pdfInclude

pdfInclude is a Progress library which allows Progress developers to output documents in Adobe PDF file format without having to use third-party tools or utilities.

pdfInclude is a stand-alone component that defines a tool set of Progress functions and procedures that aid in the output of a PDF file directly from within the Progress 4GL/ABL. pdfInclude is developed itself 100% in Progress 4GL/ABL*.

pdfInclude makes use of Progress code that is compliant with versions 9, 10 and 11 of the Progress 4GL/ABL on UNIX and/or Windows platforms.

* Except for the compression routines (PDF's Deflate), which use the excellent zlib library.

Benefits

Some of the benefits of using pdfInclude 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 is written in Progress 4GL/ABL 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).
- **Secure Report Output** : PDF 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, or Foxit Reader).

  This reduces your development effort by not having to create a separate Report Viewer. Furthermore you can embed and drive the viewer from within your application by using its OCX (with Acrobat Reader).

Note: this document was created using readme.p which came with this package, and can serve as a good introduction about how to use pdfInclude.

Changes

This document

As documents are revised, important information is added or removed.

Refer to this table to make sure you have not missed the latest information.

- 4-SEP-2012 JC Cardot:
  - document initial creation
- 1-APR-2015 JC Cardot:
  - update to match pdfInclude 4.2 & 5.0 functionalities
- 4-APR-2015 JC Cardot:
  - update for pdfInclude 5.1

pdfInclude evolutions

See the details in the annex.

Main changes to the framework:

v5.1

Pdf_inc.p

- external pdf templates: huge performance enhancements:
  - implemented a cache (new parameter "usePdfCache"): whenever a given pdf has been opened once, it will subsequently be loaded from the cache.
  - plus if the same pdf has already been opened for another stream, then no need to parse it again, we just copy it to the other stream (still faster than using the cache!)
- pdf_PageFooter: if called with "" as the procedure, then consider that the page has a footer not to add one later in pdf_close: it is the dev intention not to have a footer
- Merged documents made smaller by deduplicating the common pdf objects (including fonts, pictures, s.o.).

v5.0

pdf_inc.p

- **UTF-8 support**
- Fonts
  - support for Unicode fonts
  - **font sub-setting** (embed only the used characters in order to keep the result small)
- Pictures
  - Support for more image types (all done in ABL): BMP (1, 4, 8 & 24 bits tested), GIF (only 8 bits, non-interlaced)
- Other enhancements
  - New tag when using html like tags: . Let you specify different fonts. Usage: <font=myFont>some text, where "myFont" has been loaded previously.

- Filling forms: the caller can now specify which font and size to use, thus not using the font defined in the template (option font=myFont,fontSize=nn of pdf_fill_text()). This also works with Unicode fonts (which of course must have been loaded previously) and font subsets.
  - default header
    - new parameter headerNotOn1stPage (TRUE/FALSE)
    - new parameter headerLogo "picture_file_path|picture_width|picture_height|alignment" for the default header
  - new parameter insertPageMode: append,insert,next. By default append. When we go past the page footer, instead of always creating a new page (append), we can now insert a page, or just go to the next page.
  - new APIs:
    - pdf_close_path2 (close a path without filling it)
    - pdf_set_dash_pattern (allows to set the pattern of dashed lines)
    - pdf_place_pdf_page: like pdf_use_pdf_page but with an extra options parameter with values for Scale,X,Y,Rotate,Background,Border,BorderWidth; allows to embed an external pdf file as if it were a picture
    - pdf[_set]_RightMargin: added the right margin concept (used e.g. by default header and footer)
    - pdf_wrap_text_x (same as pdf_wrap_text but with pdf points instead of chars)

Tools

- Matrix: 2 new parameters:
  - PageBreak: Flag enabling a matrix to span on more than one page.
  - Repeat1stRow: Flag, only usable is PageBreak = "YES", used to repeat the first line of the matrix in case of a page break.

Other

- ttf2ufm.p: TTF font parser to generate .ufm files from a TTF file; entirely done in ABL, does not need an external .exe file
- PDFInclude toolkit: graphical interface to extract pages from source pdf files, rotate them, concatenate them, add a watermark, an overlay, header and footer, pdf properties (Author, title…) – à la pdftk.
- new readme file generated with pdfInclude! (this document)

---

v4.2

pdf_inc.p

- Patterns (pdf xobjects)

  You can define blocks of text, graphic, images, etc. and reuse them many times. The new API is: pdf_pattern_begin, pdf_pattern_end and pdf_pattern_use. Example implementation in pdf_default_header.
- default page header and/or footer

  (new boolean parameters "defaultHeader" and "defaultFooter") with some optional customization for the header - see the annex for more detail
- Dynamic justification for special "@@" tags (@@PageNo and @@TotalPages) is back.
- new parameter "reuseExternal" to enable many uses of an external pdf file opened once.

---

v4.1

pdf_inc.p

new functionalities

- Pictures:
  - added support for CMYK & Gray scale JPEG pictures (before, only RGB was supported)
- Encryption:
  - rewrote and optimized encryption code. RC4-128 and AES-128 is now available! Encryption works with UTF-8 sessions.
- Getting info from an existing pdf file:
  - pdf_new() can now be called with ? as a filename, thus allowing to write code to get information about an external pdf without the need of producing any pdf file (see pdfinfo.p).
  - new API pdf_ext_get_path allows to get any information from an external pdf, using "pdf paths" like "/Root/Pages/Kids[1]/Cropbox" or - more useful - /Root/Pages/Count.

v4.0

pdf_inc.p

new functionalities

- Pictures:
  - PNG support, including transparency, either one transparent color or a full alpha channel
  - pdf_place_image now uses the image dimensions if they have not been specified as parameters
- Added support for 1254 code page (Turkish) - new file 1254.diff
- New transaction mechanism, using pdf_transaction_begin/rollback/commit.
- pdf tools: implement wrap in cells for the TABLE & the MATRIX tools.
- the tag functionality now allows for underline <u>, strike <s> and links <url=> (internal to the document or http).   This also works for rotated text.
- add pdf_get_parameter2, same as pdf_get_parameter with a default value (also added pdf_get_tool_parameter2)
- pdf_wrap_text_xy[_dec] now returns (as return-value not to break the API) the Y coordinate of the last line
- Fill pdf forms has been greatly enhanced
  - combo and list boxes, check-boxes and radio buttons are now supported
  - the field design (font, color, alignment, multiline, s.o.) is now used.
  - it is now possible to retain the original form in the created document! This allows e.g. the creation of pre-filed forms.
  - It is now possible to retain the annotations from the original pdf document (e.g. links, sticky notes...).
- integration with PLOP (PDF Linearization, Optimization, Protection), a commercial product from PDFlib GmbH (PSP has been made obsolete), using pdf_Encrypt. This allows to use AES-256 encryption.

optimizations

- big rewrite to optimize the output of the text & graphic pdf operators, up to 40% of reduction in the size of the generated pdf

Pdfextract.p

- **Almost totally rewritten!** Now, should supports every pdf even with version > 1.4.

v3.3

This is the old version, available on OEHive.org/pdfInclude as free software (Eclipse licence).

Table of contents

See also the API Index for an alphabetically sorted list of all pdfInclude APIs.

pdfInclude Overview

Usage

To use pdfInclude, you have to add one include at the top of your program:

{pdf_inc.i "THIS-PROCEDURE"}

or
{pdf_inc.i}
or
{pdf_inc.i "NOT SUPER"}

This include may take one parameter which can take the following values:

THIS-PROCEDURE	The pdfInclude library is loaded as a super procedure of the THIS-PROCEDURE handle.
nothing	The pdfInclude library is loaded as a super procedure of the SESSION system handle.
other	The pdfInclude library is run persistent and accessible through the h_PDFInc handle. In this case, all RUN statements must include "IN h_PDFInc", e.g. RUN pdf_new_page **IN h_PDFInc** (pdfStream).

Coordinate system & units

Default user space units

According to the PDF specification, the coordinates on a page, be it for text or graphic elements, are working bottom-up from lower left hand corner. In other words, when looking at a page, the coordinates of the lower left hand corner are (0, 0), with X extending positively to the right and Y extending positively up.

PDF does not have a defined resolution, this is to say you can zoom a pdf page still retaining fidelity. The measurements in the pages are done in so called **default user space units**, which are 1/72 inch.

Default user space units
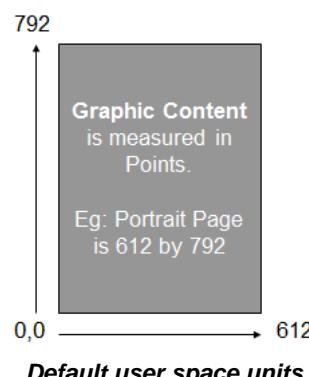
This example of a Letter page is 612 units wide and 792 units high. An A4 sized page is approximately 595.276 units wide and 841.89 units high.

pdfInclude Column/Row units

pdfInclude also implements, for some of the text API procedures, a Column/Row coordinate systems which mimics the Progress Row and Column coordinate system. And these are really only useful if you use a Fixed width font (such as Courier).

In this system, the Text starts at the top/left corner and moves across and downward. The graphical space retains the standard placement of the PDF specification.

Column/Row pdfInclude units
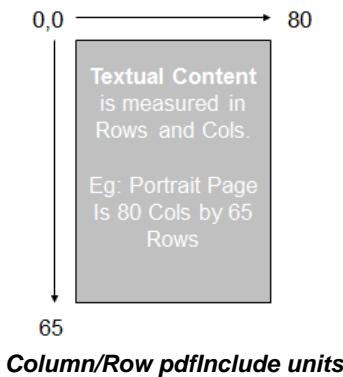
Fonts - predefined and TTF - Unicode support

While pdfInclude allows you to load any font you would like to see in your pdf document, this often results in a bigger document (the whole font file being embedded within the pdf file, or a subset of it, starting with pdfInclude 5.0).

That's why you can use the 14 pdf predefined fonts, which will not add a byte to the document:

- Courier (Courier, **Courier-Bold**, *Courier-Oblique*, ***Courier-BoldOblique***)
- Helvetica (Helvetica, **Helvetica-Bold**, *Helvetica-Oblique*, ***Helvetica-BoldOblique***)
- Times-Roman (Times-Roman, **Times-Bold**, *Times-Italic*, ***Times-BoldItalic***)
- Symbol (Σψμβολ φοντ σαμπλε ∃∀)
- ZapfDingbats (❋❀☐❀❀❀❋■❋☀❀▼▲ ❀☐■▼ ▲❀○☐●❋ ✐✑✒✓✔✗✘✗✘✛)

You can still use and embed any True Type Font within your pdf documents.

Starting with pdfInclude 5.0, there are two ways of loading a font for using within a pdf document:

- the legacy way: for this you will have to create the corresponding .afm file, using the provided ttfpt1.exe program (or using any utility allowing you to do that). Starting with pdfInclude 5.0, it is also possible to load the TTF file along with an ufm file, for Unicode support.
- the new way, where all font (i.e. TTF file) parsing is done in Progress. The font can be loaded in order to be used for UTF-8 text or any single byte code page.

Both ways allow, also starting with pdfInclude 5.0, not to load the full font file within the pdf file, but only the subset of characters which have been used, resulting in much smaller files, for the very same visual result.

See also the Fonts chapter below.

Starting with 5.0, full UTF-8 support has been implemented. This require the use of Unicode fonts. For more information, see the Text encoding chapter.

PDF viewers

Any pdf viewer is suitable, as pdfInclude generates documents conforming to the standard pdf format.

Almost any machine should have Adobe Reader installed.

However a lot of pdf viewers are available. For example Foxit Reader allows you to save pdf forms along with the contents of the widgets (like fill-ins, checkboxes...), whereas Sumatra PDF allows you to keep a PDF open while generating it with pdfInclude (very practical for debugging).

pdfInclude Introduction

Hello World

First, let's see a small example to demonstrate the basics of pdfInclude:

```
{pdf_inc.i}
/* create a new "hello.pdf" file, name the stream "Spdf" */
RUN pdf_new("Spdf", "hello.pdf").
/* create a new page in our pdf */
RUN pdf_new_page("Spdf").
/* output some text */
RUN pdf_text("Spdf", "Hello World!").
/* close the file (this will create "hello.pdf") */
RUN pdf_close("Spdf").
```

As you would expect, these few lines of code will create a new pdf file "hello.pdf", containing the very original sentence "Hello World!".

This example contains the minimum statements required for generating a pdf document.

You will find more examples at the end of this document, and more in the **samples** and **utilities** directories of the distribution.

The "PDF Stream"

As you can see in the hello world example, almost all procedures or functions take as a first argument a string, which use is to identify the pdf stream.

This is something similar to Progress streams, and allows you to generate more than one pdf simultaneously.

The "Spdf" name above is arbitrary, you can choose any string, keeping in mind that at generation time, one string identifies one pdf file.

The stream name is defined through the first call to pdf_new.

Compression

Compression of the PDF stream can be used in order to minimize the size of the resulting PDF file. To enable compression, you need to :

- install zlib (free software available for Windows and Unix)
- eventually tweak the location of the zlib library (through a modification of pdf_pre.i). For MS Windows, the library is named zlib1.dll (32 bits version is provided) whereas for Unix, the name is libz.so.1.
- in the code, just set the "Compress" parameter to TRUE:
  ```
  RUN pdf_set_parameter ("Spdf", "Compress", "TRUE").
  ```

Encryption

Encryption of the PDF stream can be used to:
- ensure that it is impossible to modify the pdf file
- and/or it is impossible to open the pdf file without a password.

pdfInclude, since version 4.1, supports the following encryption modes:
- native RC4 (40 and 128 bits) using a provided dll (only 32 bits dll is provided),
- native AES (128 bits) (using pure 4GL/ABL),
- and encryption through PLOP from PDFlib GmbH.

Native encryption

Native encryption is achieved through the use of some parameters:

Parameter	Possible values	Description
Encrypt	TRUE/**FALSE**	This parameter allows you to activate the encryption for your data stream. RC-4 encryption does not change the size of the data stream, while AES-128 encryption can increase it by a few bytes.
UserPassword	Any alpha-numeric character combination. Max 32 characters.	This parameter allows you to add a User password to the document. This password must be entered whenever the document is opened.
MasterPassword	Any alpha-numeric character combination. Max 32 characters.	This parameter allows you to add a Master password to the document. This password must be entered when trying to modify the security settings.
EncryptAlgorithm	**RC4** (default), AES.	This parameter allows you to choose between RC4 and AES encryption.
EncryptKey	**40** (default), 128	This parameter allows you to specify the level of encryption.
AllowPrint	TRUE/**FALSE**	This parameter allows you to specify whether the PDF document is printable or not.
AllowCopy	TRUE/**FALSE**	This parameter allows you to specify whether elements (such as text or Graphics) can be copied or not.
AllowModify	TRUE/**FALSE**	This parameter allows you to specify whether elements (such as Form objects) can be modified or not.
AllowAnnots	TRUE/**FALSE**	This parameter allows you to specify whether Annotations are allowed or not.
AllowForms	TRUE/**FALSE**	This parameter allows you to specify whether Forms processing can be performed or not.

Parameter	Possible values	Description
AllowExtract	TRUE/**FALSE**	This parameter allows you to specify whether Extraction can be performed or not.
AllowAssembly	TRUE/**FALSE**	This parameter allows you to specify whether Assembly can be performed or not.

PLOP encryption

PLOP encryption needs an external library to be installed, but supports stronger - although not PDF 1.4 compatible - encryption methods, such AES-256. PLOP encryption can also be interactive through the use of a dialog box allowing the user to define the password, the encryption and the permissions.

PLOP encryption uses a commercial product called "PDF Linearization, Optimization, Protection" (PLOP) from PDFlib GmbH. The PLOP product can be downloaded from www.pdflib.com.

To use this method, just call pdf_Encrypt this way:

```
RUN pdf_Encrypt ("Spdf", "masterPassword", "userPassword",
    "noprint,nohiresprint,nomodify,noforms",
    "1.6", "SHARED", FALSE).
```

See the documentation for the pdf_Encrypt procedure.

Note: the PLOP encryption does not make use of any of the native encryption parameters described above. The "Encrypt" parameter must be left to its default value (FALSE) when using PLOP encryption.

Security recommendations

In order to make your encrypted pdf files difficult/impossible to crack, the following should be respected:

- passwords of less or equal to 6 characters are more susceptible to be brute-forced (using crack software which will try all possible passwords), thus must be avoided.
- passwords should contain special characters and not ressemble a dictionnary word.
- RC4-40 bits encryption is too weak: with today's computer power it is easy to try all the possible keys.
- AES encryption is recommended over the older RC4 (and does not depend on any external dll).

In summary, AES-128 should be used (or AES-256 using PLOP), with strong passwords (longer than 6 characters, not part of the dictionnary, containing special characters).

Text encoding

Code page support

Up to version 3.3, only Latin 1 is supported.

Starting with 4.0, single byte code pages are supported, only for the 14 base fonts, through the use of pdf_set_base14_codepage.

Starting with 5.0, full UTF-8 support has been implemented. This require the use of fonts where the glyphs you plan to use are present (e.g. to create a document with Chinese characters, one has to load a font containing the Chinese characters, which is not the case of all TTF fonts you can find).

UTF-8 text and Unicode fonts can be used both in single bytes and UTF-8 OpenEdge sessions. For UTF-8 sessions, the parameter *CodePage* can be set, in order to tell pdfInclude to which codepage to convert the strings to when using a single byte encoding. If not set, its default value is "iso8859-1".

See also How to create multi-lingual documents.

Displaying text

Before displaying text, one has to choose a font, using pdf_set_font.

If the font is not part of the 14 base fonts, the font has to be previously loaded using pdf_load_font or pdf_load_font2. These procedures allow you to specify whether the text to display will be a single byte code page, or UTF-8. In order to tell to pdfInclude you will be using UTF-8 strings:

- pdf_load_font: use a .ufm font metrics (instead of .afm),
- pdf_load_font2: set the call parameter *Unicode flag* to TRUE.

To display text in a pdf file, there is a bunch of procedures which take the string to display as an argument. For example pdf_text.

When using a single byte font, the text to be passed to these procedures must be single byte, else you will have strange results. For example, passing the UTF-8 string "Jicé" to pdf_text would result in "JicÃ©" to be displayed.

When using a Unicode font, you have to pass UTF-8 strings to pdfInclude APIs.

pdfInclude API

For a full list of all pdfInclude APIs, ordered alphabetically, see the API Index.

Document properties

pdf_set_info (PROCEDURE)

This procedure allows you to set values that help you define attributes about the PDF document itself.

Sample call

```
RUN pdf_set_info("Spdf", "Author", "Jean-Christophe Cardot").
```

Parameters

- stream name (CHARACTER)
- attribute name (CHARACTER)
- attribute value (CHARACTER)

Valid attributes

- **Author**: Who created the document
- **Producer**: What is the primary producer of the document (e.g.: pdfInclude)
- **Creator**: What created the document (e.g.: 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
  e.g.: Accounts Payable
- **Title**: Any phrase that would describe the contents of the document
  e.g.: Vendor Listing

pdf_get_info (FUNCTION)

This function allows you to get back the value previously defined using pdf_set_info.

Return type

CHARACTER

Sample call

```
cAuthor = pdf_get_info ("Spdf").
```

Parameters

- stream name (CHARACTER)
- attribute name (CHARACTER), one among Author,Creator,Producer,Keywords, Subject,Title,OutputTo, the latter one being the pdf file currently being created.

pdf_set_MinPdfVersion (PROCEDURE)

This procedure allows you to specify the minimum pdf version to be used by the document. By default, pdfInclude generates pdf 1.4. Some functionalities, like 16 bits PNG pictures require higher versions, in this case 1.5. In this case, this call allows to force a minimum pdf version for the document.

Sample call

```
RUN pdf_set_MinPdfVersion ("Spdf", "1.6").
```

Parameters

- stream name (CHARACTER)
- version (CHARACTER) - the minimum pdf version to be used for the document

Setting & getting parameters

pdf_set_parameter (PROCEDURE)

This procedure allows you to set parameters associated with the specified PDF stream.

Sample call

```
RUN pdf_set_parameter ("Spdf", "Compress", "TRUE").
```

Parameters

- stream name (CHARACTER)
- parameter name (CHARACTER)
- parameter value (CHARACTER)

Valid parameter names

The following table outlines the valid parameter names and potential values. The default value is printed in bold.

- *Compress*: TRUE/**FALSE**

  The parameter allows you to specify that you want to compress the data streams. Compression occurs before encryption.

  See also How to Implement Compression

- *CodePage*

  for UTF-8 sessions, tells pdfInclude to which codepage to convert the strings to when using a single byte encoding.

  Default value: **iso8859-1**

- *LineSpacer*

  Tells pdf_skip how much extra space it must add between two lines (in pdf points).

- *ScaleX* and
- *ScaleY*

  These allow you to specify the scaling of the text. The default for each of the parameters is 1 (scale same as point size). To double the vertical size (or height) call:

```
RUN pdf_set_parameter("Spdf","ScaleY","2").
```

- *insertPageMode*: **append**, insert or next

  Starting with version 5.0, this parameter allows to change the behaviour of pdfInclude when it goes past the bottom margin when displaying text:

  ○ append: a new page is created (this is the legacy behaviour)

  ○ insert: a page is inserted after the current one

  ○ next: no page is added nor inserted, the current page is set to the next (already existing) one

    Each value sets the X and Y position at the left top of the new page (top and left margins taken into account).

Read-only parameters:

- *Version*

  this parameter contains pdfInclude version.

- *pdfVersion*

Version of the pdf file pdfInclude is generating. By default, pdfInclude generates version 1.4 of PDF. Nevertheless some functionalities imply a higher version.

Parameters related to the way the document is displayed by the viewer:

- *HideToolbar*: TRUE/**FALSE** - flag specifying whether to hide the conforming reader's menu bar when the document is active.
- *HideMenubar*: TRUE/**FALSE** - flag specifying whether to hide the conforming reader's tool bars when the document is active.
- *HideWindowUI*: TRUE/**FALSE** - flag specifying whether to hide user interface elements in the document's window (such as scroll bars and navigation controls), leaving only the document's contents displayed
- *FitWindow*: TRUE/**FALSE** - flag specifying whether to resize the document's window to fit the size of the first displayed page.
- *CenterWindow*: TRUE/**FALSE** - flag specifying whether to position the document's window in the center of the screen.
- *DisplayDocTitle*: TRUE/**FALSE** - flag specifying whether the window's title bar should display the document title taken from the Title, as defined using pdf_set_info.
- *PageMode*
  Valid values:
  - UseNone - Neither document outline nor thumbnail images visible (DEFAULT)
  - UseOutlines - Document outline (Bookmarks) visible
  - UseThumbs - Thumbnail images visible
  - FullScreen - Full-screen mode, with no menu bar, window controls, or any other window visible
- *PageLayout*
  Valid values:
  - SinglePage - Display one page at a time (DEFAULT)
  - OneColumn - Display the pages in one column
  - TwoColumnLeft - Display the pages in two columns, with odd numbered pages on the left
  - TwoColumnRight - Display the pages in two columns, with odd numbered pages on the right.

See the Default headers & footers paragraph for a description of parameters related to default header and footer:

- *defaultHeader*
- *headerLogo*
- *headerLine1*
- *headerLine2*
- *headerSeparator*
- *headerNotOn1stPage*
- *footerLine1*
- *defaultFooter*
- *footerSeparator*

See the Use HTML-like tags paragraph for a description of parameters related to writing rich text:

- *UseTags*
- *DefaultColor*
- *LinkColor*

- *BoldFont*
- *ItalicFont*
- *BoldItalicFont*
- *DefaultFont*

See the [Using an external pdf file](#) paragraph for a description of parameters related to external pdf files:

- *UseExternalPageSize*
- *formFlatten*
- *formFlattenWithDefaultValues*
- *retainAnnots*
- *drawFormFieldRect*
- *reuseExternal*
- *usePdfCache*

See the [encryption paragraph](#) above for a detailed explanation of each parameter related to encryption:

- *Encrypt*
- *UserPassword*
- *MasterPassword*
- *EncryptAlgorithm*
- *EncryptKey*
- *AllowPrint*
- *AllowCopy*
- *AllowModify*
- *AllowAnnots*
- *AllowForms*
- *AllowExtract*
- *AllowAssembly*

pdf_get_parameter (FUNCTION)

Get a parameter's value.
See parameter list in [pdf_set_parameter](#).

Return type

Character

Sample call

```
cValue = pdf_get_parameter ("Spdf", "Compress").
```

Parameters

- stream name (CHARACTER)
- parameter name (CHARACTER)

pdf_get_parameter2 (FUNCTION)

Get a parameter's value. If the parameter have not been set previously, return the provided default value.

Return type

Character

Sample call

```
cValue = pdf_get_parameter ("Spdf", "Compress", "FALSE").
```

Parameters

- stream name (CHARACTER)
- parameter name (CHARACTER)
- default value (CHARACTER)

---

pdf_incr_parameter (PROCEDURE)

This procedure allows to increment the value of an integer parameter. This is a shortcut for the use of pdf_get_parameter() and pdf_set_parameter.

Sample call

```
RUN pdf_incr_parameter(pdfStream, "BoldCount").
```

Parameters

- stream name (CHARACTER)
- parameter name (CHARACTER)

---

pdf_decr_parameter (PROCEDURE)

This procedure allows to decrement the value of an integer parameter. This is a shortcut for the use of pdf_get_parameter() and pdf_set_parameter.

Sample call

```
RUN pdf_decr_parameter(pdfStream, "BoldCount").
```

Parameters

- stream name (CHARACTER)
- parameter name (CHARACTER)

Document structure

pdf_new (PROCEDURE)

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:

- 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
- RightMargin: 10 - starting with version 5.0
- TopMargin: 50

Sample call

```
RUN pdf_new("Spdf", "/path/to/file/myfile.pdf").
```

Parameters

- stream name (CHARACTER) - this is where the stream name is defined and created.
- file name (CHARACTER) - the name of the PDF file to generate (as it will be stored on the Operating System). The name might be '?' (the unknown value), in which case no file will be generated. This can be useful for example to query an existing pdf file in order to fetch some information from it (like title, author, number of pages), without generating any extra pdf file.

pdf_new_page (PROCEDURE)

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.

Sample call

```
RUN pdf_new_page("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_new_page2 (PROCEDURE)

Same as pdf_new_page 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.

Sample call

```
RUN pdf_new_page2("Spdf", "Landscape").
```

Parameters

- stream name (CHARACTER)
- page orientation (CHARACTER) : "Portrait" or "Landscape"

pdf_insert_page (PROCEDURE)

This procedure inserts a page between two already generated pages.

Sample call

```
RUN pdf_insert_page("Spdf", 5, "Before").
```

Parameters

- stream name (CHARACTER)
- page number (INTEGER)
- position ("Before" or "After") (CHARACTER)

pdf_close (PROCEDURE)

This procedure closes the identified PDF stream and generates the PDF document that was identified in the pdf_new procedure.
The RETURN-VALUE of this procedure might be set to the list of encountered problems. If RETURN-VALUE > "", the generated pdf should be considered as invalid.

Sample call

```
RUN pdf_close("Spdf").
IF RETURN-VALUE > "" THEN
    MESSAGE RETURN-VALUE
        VIEW-AS ALERT-BOX INFO BUTTONS OK.
```

Parameters

- stream name (CHARACTER)

pdf_set_PaperType (PROCEDURE)

This procedure allows you to select Paper Types that have predefined height and widths.

Sample call

```
RUN pdf_set_PaperType("Spdf", "A4").
```

Parameters

- stream name (CHARACTER)
- paper type (CHARACTER)

Valid paper types

The paper sizes are given in PDF units.
- A0: 2380x3368
- A1: 1684x2380
- A2: 1190x1684
- A3: 842x1190
- A4: 595x842
- A5: 421x595
- A6: 297x421
- B5: 501x709
- LETTER: 612x792
- LEGAL: 612x1008
- LEDGER: 1224x792

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 values would be reversed.

Note: when working with an external PDF file, you can use the external file paper type, setting the parameter "UseExternalPageSize" to "TRUE".

pdf_set_PageRotate (PROCEDURE)

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.

Sample call

```
RUN pdf_set_PageRotate("Spdf", 90).
```

Parameters

- stream name (CHARACTER)
- angle (INTEGER)

Valid angles

0,90,180,270

pdf_set_Orientation (PROCEDURE)

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. You can use pdf_new_page2 to directly create a new page, specifying its orientation.

Sample call

```
RUN pdf_set_Orientation("Spdf", "Landscape").
```

Parameters

- stream name (CHARACTER)
- orientation (CHARACTER)

Valid orientations

Portrait, Landscape (Portrait is the default).

pdf_set_PageWidth (PROCEDURE)

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 procedure.

Sample call

```
RUN pdf_set_PageWidth("Spdf", 144).
```

Parameters

- stream name (CHARACTER)
- page width (INTEGER)

pdf_set_PageHeight (PROCEDURE)

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.

Sample call

```
RUN pdf_set_PageHeight("Spdf", 1728).
```

Parameters

- stream name (CHARACTER)
- page height (INTEGER)

pdf_set_LeftMargin (PROCEDURE)

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).

Sample call

```
RUN pdf_set_LeftMargin("Spdf", 12).
```

Parameters

- stream name (CHARACTER)
- left margin (INTEGER)

pdf_set_RightMargin (PROCEDURE)

This procedure allows you to set the right margin of the document. Appeared in pdfInclude 5.0.
Used for example in the default header and footer.

Sample call

```
RUN pdf_set_RightMargin("Spdf", 12).
```

Parameters

- stream name (CHARACTER)
- right margin (INTEGER)

pdf_set_TopMargin (PROCEDURE)

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

Sample call

```
RUN pdf_set_TopMargin("Spdf", 50).
```

Parameters

- stream name (CHARACTER)
- top margin (INTEGER)

pdf_set_BottomMargin (PROCEDURE)

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. 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.

Sample call

```
RUN pdf_set_BottomMargin("Spdf", 50).
```

Parameters

- stream name (CHARACTER)
- bottom margin (INTEGER)

pdf_set_VerticalSpace (PROCEDURE)

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.

Sample call

```
RUN pdf_set_VerticalSpace("Spdf", 10).
```

Parameters

- stream name (CHARACTER)
- vertical space (DECIMAL)

pdf_VerticalSpace (FUNCTION)

This function returns the current VerticalSpace.

Return type

DECIMAL

Sample call

```
dVertSpc = pdf_VerticalSpace ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_set_Page (PROCEDURE)

This procedure allows to change the page number. Using pdf_set_Page the developer can add information to a previously generated page, thus it is not compulsory to generate the pdf file in the order of pages.

Sample call

```
RUN pdf_set_Page("Spdf", 2).
```

Parameters

- stream name (CHARACTER)
- page number (INTEGER)

pdf_TotalPages (FUNCTION)

This function allows you to print the total number of pages in your document. This is particularly useful for headers and footers.

Return type

CHARACTER

Sample call

```
RUN pdf_text("Spdf", pdf_TotalPages ("Spdf")).
```

Parameters

- stream name (CHARACTER)

pdf_PageNo (FUNCTION)

This function allows you to print the current page number in your document. This is useful when you don't know in advance the number of the page you are generating (especially when using pdf_insert_page), else you can use pdf_Page which returns the current page number. This call is usually used in headers and/or footers.

Return type

CHARACTER

Sample call

```
RUN pdf_text("Spdf", pdf_PageNo ("Spdf")).
```

Parameters

- stream name (CHARACTER)

Headers and footers

Apart from default header and footer, pdfInclude allows you to define your custom ones. For more detail, see How to Implement Page Headers and Footers.

pdf_PageHeader (FUNCTION)

This function allows to specify a custom header procedure.

Return type
LOGICAL

Sample call
```
pdf_PageHeader("Spdf", THIS-PROCEDURE, "myHeader").
```

Parameters
- stream name (CHARACTER)
- procedure handle (HANDLE) - handle of the procedure where the header internal procedure is located

- procedure name (CHARACTER) - name of the internal procedure which will output the header

pdf_PageFooter (FUNCTION)

This function allows to specify a custom footer procedure. It can be called many times during the generation of the documents. The "procedure name" argument will be run for all the pages starting at the current one, until another call to this API.

Return type
LOGICAL

Sample call

```
pdf_PageFooter("Spdf", THIS-PROCEDURE, "myFooter").
```

Parameters
- stream name (CHARACTER)
- procedure handle (HANDLE) - handle of the procedure where the footer internal procedure is located
- procedure name (CHARACTER) - name of the internal procedure which will output the footer. Starting with version 5.1, calling this API with procedure name = "" will disable the footer for the current page and subsequent ones, until another call is issued.

pdf_exec_footer (PROCEDURE)

This procedure allows you to force the display of the footer in the current page, in case it got "forgotten".

Starting with pdfInclude 5.0, this should not be necessary anymore.

Sample call

```
RUN pdf_exec_footer ("Spdf").
```

Parameters
- stream name (CHARACTER)

Stream manipulation

Streams can be merged using the pdf_merge_stream procedure.

This allows to simultaneously generate various pdf files (e.g. invoices) and at the end of the process concatenate all (or some) of them into another pdf file (like a summary document), provided the stream name is different for each generated file.

pdf_merge_stream (PROCEDURE)

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).

Sample call

```
RUN pdf_merge_stream ("Spdf2", "Spdf", 1).
```

Parameters

- stream from (CHARACTER) - stream name we are merging
- stream to (CHARACTER) - stream name into which we are merging
- number of copies (INTEGER) - number of times <stream from> will be copied into <stream to>

---

pdf_ReplaceText (PROCEDURE)

This procedure allows you to replace some text on the copies of the stream realized via pdf_merge_stream.

Sample call

```
RUN pdf_ReplaceText ("Spdf",1,"Title","Title - COPY").
```

Parameters

- stream name (CHARACTER)
- pdfNbrCopies (INTEGER) - Number of the stream copy where to replace text
- text from (CHARACTER) - Text to be searched. Should be inferior or equal to the number of copies (see pdf_merge_stream arguments)

---

pdf_reset_stream (PROCEDURE)

This procedure clears all pdf content for a given pdf stream. This call cancels all the calls done for one stream from the first pdf_new, included.

Sample call

```
RUN pdf_reset_stream ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_reset_all (PROCEDURE)

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

Sample call

```
RUN pdf_reset_all ("Spdf").
```

Parameters

Getting values

pdf_Page (FUNCTION)

This function returns the value of the current Page parameter. The Page parameter is incremented after every call to a pdf_new_page. It is also updated after each call to pdf_set_page or pdf_insert_page.

Return type
INTEGER

Sample call
```
iPage = pdf_Page("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_PaperType (FUNCTION)

This function returns the current paper type, defined using pdf_set_PaperType (see above), or the default value set by pdf_new_page.

Return type
CHARACTER

Sample call
```
cSheet = pdf_PaperType ("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_LeftMargin (FUNCTION)

This function returns the value of the Left Margin.

Return type
INTEGER

Sample call
```
iMargin = pdf_LeftMargin("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_RightMargin (FUNCTION)

This function returns the value of the Right Margin. Appeared in pdfInclude 5.0.

Return type
INTEGER

Sample call
```
iMargin = pdf_RightMargin("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_TopMargin (FUNCTION)

This function returns the value of the Top Margin.

Return type
INTEGER

Sample call
```
iMargin = pdf_TopMargin("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_BottomMargin (FUNCTION)

This function returns the value of the Bottom Margin.

Return type
INTEGER

Sample call
```
iMargin = pdf_BottomMargin("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_PageRotate (FUNCTION)

This function returns the value of the current PageRotate parameter.

Return type
INTEGER - Can only be 0, 90, 180 or 270.

Sample call
```
iRotate = pdf_PageRotate("Spdf").
```

Parameters
- stream name (CHARACTER)

---

pdf_PageWidth (FUNCTION)

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_set_PaperType.

Return type

INTEGER

Sample call

```
iWidth = pdf_PageWidth("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_PageHeight (FUNCTION)

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_set_PaperType.

Return type

INTEGER

Sample call

```
iHeight = pdf_PageHeight("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_Orientation (FUNCTION)

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

Return type

CHARACTER
Possible Values: "Portrait" or "Landscape"

Sample call

```
IF pdf_Orientation("Spdf") = "Landscape" THEN...
```

Parameters

- stream name (CHARACTER)

Document encryption

Document encryption preferred method is achieved through the use of the encryption parameters described <u>above</u>. Native encryption allows for RC4-40, RC4-128 and AES-128 algorithms.

Nevertheless, it is also possible to use an external library, PLOP from <u>PDFLib GmbH</u>, using pdf_Encrypt. This method allows to use more encryption algorithms, still not available through native encryption, like AES-256.

pdf_Encrypt (PROCEDURE)

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

This procedure call integrates with PDFplop.w (another external procedure). PDFplop.w uses a commercial product called "PDF Linearization, Optimization, Protection" (PLOP) from PDFlib GmbH. The PLOP product can be downloaded from <u>www.pdflib.com</u>.

Note : previous versions of pdfInclude used Pretty Safe PDF (PSP) from the same editor. PSP has been superseded - and thus replaced in pdfInclude - by PLOP.

Sample call

```
RUN pdf_Encrypt ("Spdf", "masterPassword", "userPassword",
    "noprint,nohiresprint,nomodify,noforms",
    "1.6", "SHARED", FALSE).
```

Parameters

- stream name (CHARACTER)
- Master Password (CHARACTER) - The PDF document's Master Password
- User Password (CHARACTER) - The PDF document's User Password
- Access List (CHARACTER) - List identifying Access Rights
- PDF version (CHARACTER) - 1.4, 1.5, 1.6, 1.7 or 2.0. PLOP will select the best encryption algorithm depending on the PDF version.
- Encryption Mode (CHARACTER) - COM, SHARED or OS
- Silent Encryption (LOGICAL)

Parameters' details

- 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 list outlines the valid entries for the comma-delimited list:
  - 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
- • PDF version: 1.4, 1.5, 1.6, 1.7 or 2.0. PLOP will select the best encryption algorithm depending on the PDF version.
- • Encryption Mode OS Description
  - ○ COM Windows Uses the COM version of PLOP. Accessed via a COM-Handle.
  - ○ SHARED Windows, *NIX Uses the DLL (for Windows) or Shared Library (for *NIX). Accessed via Functions calls defined in PLOP.i and PLOPbind.p.
  - ○ OS Windows, *NIX Uses the command line PLOP tool. Accessed via an OS-COMMAND call.
- • Silent Encryption: Either a TRUE or FALSE value.

If TRUE, then the Encryption routine (PDFplop.w) is called without any interaction. The Encryption is performed silently based on the passed parameters.

If FALSE then the Encryption routine (PDFplop.w) is called expecting the user to enter some values.   The Access Rights are defaulted to the supplied list but the Master and User passwords are ignored. This user must enter these values. The following screen print illustrates the interactive encryption screen:
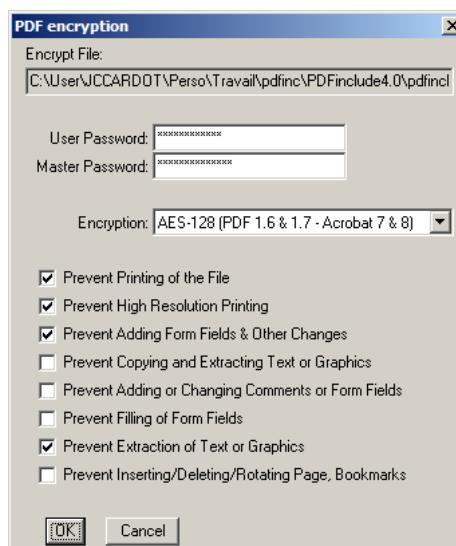
pdfInclude PLOP graphical interface

Text properties

Text orientation

pdf_text_rotate (PROCEDURE)

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

The sample call produces this kind of results.

Sample call

```
RUN pdf_text_rotate ("Spdf", 5).
```

Parameters

- stream name (CHARACTER)
- angle (INTEGER) - Angle to rotate and display text in. Starting with version 4, all values between 0 and 360 are possible. Older versions only support 0, 90, 180 and 270 degrees.

pdf_Angle (FUNCTION)

This function returns the current text rotation angle, set by pdf_text_rotate.

Return type
INTEGER

Sample call

```
iAngle = pdf_Angle ("Spdf").
```

Parameters

- stream name (CHARACTER)

Text shape

pdf_text_scale (PROCEDURE)

This procedure sets the scale to be used when displaying text. The scale can be expressed for both horizontal and vertical directions, allowing to stretch the text in one direction.

The sample call produces this kind of results.

Sample call

```
RUN pdf_text_scale ("Spdf", 0.5, 1.5).
```

Parameters

- stream name (CHARACTER)
- horizontal scale (DECIMAL)
- vertical scale (DECIMAL)

---

pdf_ScaleX (FUNCTION)

This function returns the current scale on the X axis.

Return type

DECIMAL

Sample call

```
iScaleX = pdf_ScaleX ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_ScaleY (FUNCTION)

This function returns the current scale on the Y axis.

Return type

DECIMAL

Sample call

```
iScaleY = pdf_ScaleY ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_text_skew (PROCEDURE)

This procedure allows to skew the text that will be displayed later. The skew amount is expressed using two angles a and b, which skew the x axis by an angle a and the y axis by an angle b.

The sample call produces this kind of results.

Sample call

```
RUN pdf_text_skew ("Spdf", 5, 10).
```

Parameters

- stream name (CHARACTER)
- angle a (DECIMAL)
- angle b (DECIMAL)

---

pdf_text_render (PROCEDURE)

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

Sample call

```
RUN pdf_text_render ("Spdf", 1).
```

Parameters

- stream name (CHARACTER)
- Render Type as Integer - An integer value representing how to render the text. The result is demonstrated below - except for invisible text of course:

  ○ 0 - Fill Text
  ○ 1 - Stroke Text
  ○ **2 - Fill, then Stroke Text**
  ○ 3 - Neither Fill nor Stroke Text (invisible)

pdf_Render (FUNCTION)

This function returns the last value used when calling pdf_text_render. See above.

Return type
INTEGER

Sample call

```
pdf_Render ("Spdf").
```

Parameters

- stream name (CHARACTER)

Text Color

pdf_set_TextRed (PROCEDURE)

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 (e.g.: 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)'.

Sample call

```
RUN pdf_set_TextRed("Spdf", 1.0).
```

Parameters

- stream name (CHARACTER)
- red value (DECIMAL): any decimal value between 0 and 1.

pdf_set_TextGreen (PROCEDURE)

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 (e.g.: 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)'.

Sample call

```
RUN pdf_set_TextGreen("Spdf", 1.0).
```

Parameters

- stream name (CHARACTER)
- green value (DECIMAL): any decimal value between 0 and 1.

pdf_set_TextBlue (PROCEDURE)

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 (e.g.: 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)'.

Sample call

```
RUN pdf_set_TextBlue("Spdf", 1.0).
```

Parameters

- stream name (CHARACTER)
- blue value (DECIMAL): any decimal value between 0 and 1.

pdf_text_color (PROCEDURE)

This procedure allows you to set the color that the following text displays should appear in. Thus you can change the 3 color components at once, instead of calling the 3 procedures documented above.

The sample call produces this kind of results.

Sample call

```
RUN pdf_text_color("Spdf", 0.0, 0.4, 0.4).
```

Parameters

- stream name (CHARACTER)
- red value (DECIMAL): any decimal value between 0 and 1.
- green value (DECIMAL): any decimal value between 0 and 1.
- blue value (DECIMAL): any decimal value between 0 and 1.

pdf_rgb (PROCEDURE)

The color procedures, both for text and graphic elements, both accept decimal numbers between 0 and 1. Sometimes the color is available as a triplet of integer values between 0 and 255, or as an hexadecimal notation, like in HTML.

pdf_rgb transforms these values in order to call the specified procedure with the expected range of values.

Sample call

```
RUN pdf_rgb ("Spdf", "pdf_text_color", "127,12,144"). /* triplet    */
RUN pdf_rgb ("Spdf", "pdf_text_color", "127012144").  /* same       */
RUN pdf_rgb ("Spdf", "pdf_stroke_color", "#12BABE").  /* HTML colour */
RUN pdf_rgb ("Spdf", "pdf_stroke_fill", "0xFACE12").  /* Hexadecimal */
```

Parameters

- stream name (CHARACTER)
- pdfInclude API (CHARACTER) - one of pdf_text_color, pdf_stroke_color, pdf_stroke_fill
- color (CHARACTER) - see the sample calls above for the 4 possible color formats.

pdf_TextRed (FUNCTION)

This function allows you to get the red component for the current text color.

Return type

DECIMAL between 0 and 1

Sample call

```
dRed = pdf_TextRed ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_TextGreen (FUNCTION)

This function allows you to get the green component for the current text color.

<u>Return type</u>

DECIMAL between 0 and 1

<u>Sample call</u>

```
dGreen = pdf_TextGreen ("Spdf").
```

<u>Parameters</u>

* stream name (CHARACTER)

pdf_TextBlue (FUNCTION)

This function allows you to get the blue component for the current text color.

<u>Return type</u>

DECIMAL between 0 and 1

<u>Sample call</u>

```
dBlue = pdf_TextBlue ("Spdf").
```

<u>Parameters</u>

* stream name (CHARACTER)

Fonts

Before using a font different from the base 14 fonts, its TTF file has to be loaded by pdfInclude, with pdf_load_font or pdf_load_font2. Then the code will switch to the loaded font using pdf_set_font, before calling any procedure which display text.

Starting with version 5.0, both pdf_load_font and pdf_load_font2:
- can load single byte or Unicode fonts,
- allow you to specify an "subset" option, which will embed only the subset of characters used within your pdf document, resulting in a much smaller pdf file. The list of used characters can be extended using pdf_subset_add_string and/or pdf_subset_add_range, which allows you to "force include" some characters in the embedded font subset.

---

pdf_load_font (PROCEDURE)

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

You will need an AFM file, generated from the TTF file, using for example the provided ttf2pt1.exe. Starting with version 5.0, you can also use an UFM file (generated using ttf2ufm.exe), in order to take advantage of the new UTF-8 support.

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/public/developer/en/opentype/glyphlist.txt

Sample call

```
RUN pdf_load_font ("Spdf","Code39","fonts\Code39.ttf",
                   "fonts\code39.afm","").
```

Parameters
- stream name (CHARACTER)
- Font Name (CHARACTER) - Unique Font Name (no spaces). To the font name can be added 2 extra parameters, separated by pipes "|":

- NOEMBED: do not embed the font file within the PDF file. In this case, the PDF file can only be opened on a computer with the font installed. This is not recommended.
- SUBSET: embed only definitions for the characters which have been used on the document. This can reduce dramatically the size of the generated PDF file, specially when using Unicode fonts which can be huge (more than 20 Mb for Arial Unicode). This functionality appeared in version 5.0.
- Font File (CHARACTER) - Location of TTF Font File (uses PROPATH)
- Font AFM File (CHARACTER) - Location of AFM file (uses PROPATH)
- Font DIF File (CHARACTER) - Location of DIF file (uses PROPATH)

---

pdf_load_font2 (PROCEDURE)

Starting with version 5.0, pdfInclude can parse the TTF files by itself, without the need of the extra AFM or UFM file (thus without the need of an external executable).

This parsing, being more integrated with pdfInclude, leads to better results. Notably, for some fonts, some characters cannot be shown if using the old pdf_load_font.

Sample call

```
RUN pdf_load_font2 ("Spdf","ArialUnicode","fonts\arialuni.ttf",YES,"","SUBSET").
```

Parameters

- stream name (CHARACTER)
- Font Name (CHARACTER) - Unique Font Name (no spaces).
- Font File (CHARACTER) - Location of TTF Font File (uses PROPATH)
- Unicode flag (LOGICAL) - Flag to indicate if the font is loaded to display UTF-8 text or not.
- Font DIF File (CHARACTER) - Location of DIF file (uses PROPATH)
- Font loading options (CHARACTER) - an option list, i.e. a key=value coma separated list. The recognised keys are:
  - NOEMBED: do not embed the font file within the PDF file. In this case, the PDF file can only be opened on a computer with the font installed. This is not recommended.
  - SUBSET: embed only character definitions fore those which have been used on the document. This can reduce dramatically the size of the generated PDF file, specially when using Unicode fonts which can be huge (more than 20 Mb for Arial Unicode). This functionality appeared in version 5.0.

    Note: for LOGICAL values, the value is useless. This means that "SUBSET" is equivalent to "SUBSET=YES". To not create the font subset, do not specify anything, or "SUBSET=NO"; the two being equivalent.
  - Font encoding options: you might need these two options if you get the following error message: "parseFont-Font "myFont" has no encoding table for PlatformID/EncodingID=3/1. Possible values are: 1/0 3/0"
    - PlatformID: Windows: 3, Macintosh: 1
    - EncodingID: for platform 3: 0: Symbol, 1: Unicode, 2: ShiftJIS, 3: PRC, 4: Big5, 5: Wansung, 6: Johab, 7: Reserved, 8: Reserved, 9: Reserved, 10: UCS-4

---

pdf_set_font (PROCEDURE)

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 to another font.

As mentioned before, there is no need in loading the font if it is in the list below. Else, the

---

font have to be loaded using pdf_load_font or pdf_load_font2.

The following list defines the names of the PDF Base14 Fonts:

- `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 Courier 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-Roman: 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
- Symbol: Σψμβολ φοντ σαμπλε ∃∀
- ZapfDingbats: ✳❀❑✸✤✲■❅✳✣❀▼▲ ❀❑■▼ ▲❀○❑●❀ ✐✑✒✓✔✕✖✗✘✛

pdf_PointSize (FUNCTION)

This function returns the current font size, as set by pdf_set_font.

Return type
DECIMAL

Sample call
```
pdf_PointSize ("Spdf").
```

Parameters
- stream name (CHARACTER)

pdf_font_diff (PROCEDURE)

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).

Sample call
```
RUN pdf_font_diff ("Spdf", 190,"/nsuperior")
```

Parameters
- stream name (CHARACTER)
- 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

pdf_set_base14_codepage (PROCEDURE)

This procedure sets the code page used when displaying text using one of the 14 base fonts.

The default PDF code page is Latin 1 (ISO8859-1 or MS 1252).

This procedures allows to change it to another single byte code page. Currently supported by pdfInclude:

- ISO859-2 (Latin 2)
- 1250 (Latin 2)
- 1251 (Cyrillic)
- 1254 (Turkish)

Supporting a new code page is only a matter of creating a new .diff file, based on the model of 1251.diff or 1254.diff (which use 2 different definition styles).

<u>Sample call</u>

```
RUN pdf_set_base14_codepage ("Spdf", "iso8859-2").
```

<u>Parameters</u>

- stream name (CHARACTER)
- code page (CHARACTER): one among 1250, 1251, 1254 or iso8859-2

---

pdf_subset_add_string (PROCEDURE)

This procedure, used in conjunction with font sub-setting (starting with version 5.0), allows to add more glyphs (or characters) to the subset of used glyphs within the document. It can be useful if you need to ensure that certain glyphs are present in the font subset embedded in the pdf file.

<u>Sample call</u>

```
RUN pdf_subset_add_string ("Spdf", "ArialUni", "ABC...XYZ", YES).
```

<u>Parameters</u>

- stream name (CHARACTER)
- font name (CHARACTER): name of the font
- string (CHARACTER): string from which to take the characters to add to the subset
- Unicode flag (CHARACTER): YES/NO - consider the input string as UTF-8 (YES) or single byte (NO).

---

pdf_subset_add_range (PROCEDURE)

This procedure performs the same as does pdf_subset_add_string, for a range of glyphs (or characters).

<u>Sample call</u>

```
RUN pdf_subset_add_range ("Spdf", "ArialUni", 65, 90).
```

<u>Parameters</u>

- stream name (CHARACTER)
- font name (CHARACTER): name of the font
- character code from (INTEGER): first character of the range

* character code to (INTEGER): last character of the range

---

pdf_Font (FUNCTION)

This function returns the name for the current font.

Return type

CHARACTER

Sample call

```
cFont = pdf_Font ("Spdf").
```

Parameters

* stream name (CHARACTER)

---

pdf_Font_Loaded (FUNCTION)

This function tells if a given font has been loaded or not.

Return type

LOGICAL

Sample call

```
IF NOT pdf_Font_Loaded ("Spdf", "ArialUni") THEN...
```

Parameters

* stream name (CHARACTER)
* font name (CHARACTER)

---

pdf_FontType (FUNCTION)

This function returns the pitch of the current font: VARIABLE or FIXED.

Return type

CHARACTER

Sample call

```
cFontType = pdf_FontType ("Spdf").
```

Parameters

* stream name (CHARACTER)

---

pdf_GetBestFont (PROCEDURE)

This procedure calculate the best font size to use to insert text into a given range along the X axis; tests are done in 0.5 point size increments.

Sample call

```
RUN pdf_GetBestFont ("Spdf", "ArialUni",
                     INPUT-OUTPUT cText, INPUT-OUTPUT dFontSize,
```

```
6, NO, 0, 48).
```

Parameters

- stream name (CHARACTER)
- font name (CHARACTER) - name of the font to use
- text (CHARACTER), INPUT-OUTPUT - Text to measure; when the text is chopped, the value is modified.
- font size (DECIMAL), INPUT-OUTPUT - Start font size; the calculated value is returned through this parameter.
- smallest size (DECIMAL) - Smallest font size to use
- chop text (LOGICAL) - If the smallest font is too small for the text to fit, the input text can be chopped to the biggest fitting text
- X from (INTEGER) - start X position
- X to (INTEGER) - end X position

Width of a character string

The following functions all return the width of a character string. The width of a string is the number of points of a given text string using a specific font and font size. The width is calculated as follows:

width = sum of each character widths / 1000 * font size

For FIXED fonts, this is equivalent to:

width = LENGTH(text) * character width / 1000 * font size

For VARIABLE fonts, the individual character widths may differ for each character within a given font, whereas for FIXED fonts, every characters have the same width. These functions are useful in a lot of situations when you need to know the size of the text to - for example - surround it by a rectangle, or position it precisely, relatively to other elements.

The following functions differ only in their parameters and the way they get the font and font size.

pdf_text_width (FUNCTION)

This function uses the current Font and PointSize.

Return type
INTEGER

Sample call

```
pdf_text_width ("Spdf", "pdfInclude").
```

Parameters
- stream name (CHARACTER)
- text (CHARACTER) - the text for which we want to compute the width

pdf_text_widthdec (FUNCTION)

Same as the previous one, returns DECIMAL instead of INTEGER.

Return type
DECIMAL

Sample call

```
pdf_text_widthdec ("Spdf", "pdfInclude").
```

Parameters
- stream name (CHARACTER)
- text (CHARACTER) - the text for which we want to compute the width

pdf_text_widthdec2 (FUNCTION)

Uses the font tag and a font size passed as parameters.

Return type
DECIMAL

Sample call

```
pdf_text_widthdec2 ("Spdf", "F1", 10, "pdfInclude").
```

Parameters

- stream name (CHARACTER)
- font tag (CHARACTER)
- font size (DECIMAL)
- text (CHARACTER) - the text for which we want to compute the width

---

pdf_text_fontwidth (FUNCTION)

Uses the font name passed as parameter, and the current PointSize as font size.

Return type
DECIMAL

Sample call

```
pdf_text_fontwidth ("Spdf", "ArialUni", "pdfInclude").
```

Parameters

- stream name (CHARACTER)
- font name (CHARACTER)
- text (CHARACTER) - the text for which we want to compute the width

---

pdf_text_fontwidth2 (FUNCTION)

Uses the font name and size passed as parameters.

Return type
DECIMAL

Sample call

```
pdf_text_fontwidth2 ("Spdf", "ArialUni", 12, "pdfInclude").
```

Parameters

- stream name (CHARACTER)
- font name (CHARACTER)
- font size (DECIMAL)
- text (CHARACTER) - the text for which we want to compute the width

---

pdf_GetNumFittingChars (FUNCTION)

This function allows you to know how many chars from the input text will fit into the width specified using a starting and an ending X coordinate.

Return type

INTEGER

Sample call

```
iNumChars = pdf_GetNumFittingChars ("Spdf", "pdfInclude rulez", 12, 144).
```

Parameters

- stream name (CHARACTER)
- text (CHARACTER) - the text to be tested
- from X (INTEGER) - starting X coordinate
- to X (INTEGER) - ending Y coordinate

Return type

INTEGER

Sample call

Writing text

pdfInclude supports two different concepts for writing text:

- text flow
- text absolute positioning

The two ways of writing text can be mixed into the same document.

The first one allows to write a document, it is more like a word processing software. You output text, skip lines, pages are created automatically, s.o.

The second one allows to position text at the exact position in the page you want to, using absolute coordinates, expressed in points (in fact in user space units).

For the position arguments, we use 2 different terms below: Column/Row or X/Y, according to the type of coordinates the procedure is using. See Coordinate system & units.

Starting with version 5.0, the text writing procedures all support UTF-8 strings, provided a Unicode font has been loaded and selected before.

Write text as a flow

The following procedures write the text in the document as a continuous flow.

Most of the procedures below accept their position argument (if any) as columns and rows.

pdf_text (PROCEDURE)

This procedure allows you to place the passed text string at the current position (i.e. current TextX and TextY Text Space points).

Sample call

```
RUN pdf_text ("Spdf","Hello World").
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value

pdf_text_char (PROCEDURE)

This procedure outputs the character using the specified character code. This might be useful if for example you want to change your font to ZapfDingbats then output a check mark into your text stream.

Sample call

```
RUN pdf_text_char ("Spdf",63).
```

Parameters

- stream name (CHARACTER)
- Value (INTEGER) - Any octal value between 0 and 377

pdf_skip (PROCEDURE)

This procedure will skip to the next line in the Text Space (the Text space Y position is decremented of the Point size of the font, defined in pdf_set_font). This also updates the current TextX and TextY attributes for the stream.

When the parameter *LineSpacer* is defined, pdf_skip uses its value to skip more space (the value is defined in pdf points).

When the text goes past the bottom margin, a new page is automatically appended to the document. Starting with pdfInclude 5.0, the parameter *insertPageMode* can be set to append, insert or simply go to the next (existing) page.

Sample call

```
RUN pdf_skip ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_skipn (PROCEDURE)

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.

Sample call

```
RUN pdf_skipn ("Spdf",2).
```

Parameters

- stream name (CHARACTER)
- Number of Lines (INTEGER) - Number of lines to skip

pdf_text_at (PROCEDURE)

This procedure allows you to place the passed text string at the specified column on the current row (TextX). The column parameter corresponds to a number of characters (if using a mono-space font) starting from the left margin. For proportional fonts, the column parameter corresponds to a number of spaces (" " character).

Sample call

```
RUN pdf_text_at ("Spdf","Hello World",12).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value
- Column (INTEGER) - Any non-zero integer value

pdf_text_to (PROCEDURE)

This procedure allows you to place the passed text string right aligned to the specified column, on the current row (TextX). The column parameter corresponds to a number of characters (if using a mono-space font) starting from the left margin. For proportional fonts, the column parameter corresponds to a number of the "E" character.

Sample call

```
RUN pdf_text_to ("Spdf","Hello World",30).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value
- Column (INTEGER) - Any non-zero integer value

pdf_wrap_text (PROCEDURE)

This procedure allows you to place text into the PDF document within a from/to column location. The text starts on the current row (TextX).

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 large character fields or strings you do not know the size at development time.

This is particularly useful for mono-space fonts; for proportional fonts, the from and to columns consider the "W" character.

Sample call

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

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value. Might contain carriage returns.
- From Column (INTEGER) - Column at which to start Text output
- To Column (INTEGER) - Column to stop Text Output
- Alignment (CHARACTER) - how to align the text between the two columns: "left" or "right"
- Max Y (INTEGER) - OUTPUT variable: last Y position (i.e. where we ended in the page)

pdf_get_wrap_length (FUNCTION)

This function returns how high (in pdf points) a text would be if you were to display it using pdf_wrap_text.

The result is a multiple of the VerticalSpace or, if it is not defined, the PointSize.

Return type

INTEGER

Sample call

```
 pdf_get_wrap_length ("Spdf").
```

Parameters

- stream name (CHARACTER)
- text (CHARACTER) - the text to be tested
- width (INTEGER) - maximum width, expressed in number of characters.

---

pdf_wrap_text_x (PROCEDURE)

This procedure is equivalent a pdf_wrap_text, with the columns expressed in space units. This makes it more suitable for proportional fonts. Appeared in pdfInclude 5.0.

Sample call

```
RUN pdf_wrap_text_x
    ("Spdf",
     "This is a lot of text that could include line breaks etc.",
     50, 570,
     "left", OUTPUT deY).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value. Might contain carriage returns.
- From X (DECIMAL) - X position at which to start Text output
- To X (DECIMAL) - X position to stop Text Output
- Alignment (CHARACTER) - how to align the text between the two columns: "left", "right" or "center"
- Y (DECIMAL) - OUTPUT variable: last Y position - allows to know where in the page is the last line of text.

Write text using absolute type placement

It is possible to position the text very precisely within the page, keeping full control on the result, using the following procedures.

pdf_set_TextX (PROCEDURE)

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.

Sample call

```
RUN pdf_set_TextX("Spdf",200).
```

Parameters

* stream name (CHARACTER)
* X value (DECIMAL) - Value to set the Text X coordinate to

pdf_set_TextY (PROCEDURE)

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.

Sample call

```
RUN pdf_set_TextY("Spdf",300).
```

Parameters

* stream name (CHARACTER)
* Y value (DECIMAL) - Value to set the Text Y coordinate to

pdf_set_TextXY (PROCEDURE)

Combine the two previous procedures into one unique call.

Sample call

```
RUN pdf_set_TextXY("Spdf",200,300,YES).
```

Parameters

* stream name (CHARACTER)
* X value (DECIMAL) - Value to set the Text X coordinate to
* Y value (DECIMAL) - Value to set the Text Y coordinate to
* update text matrix (LOGICAL) - should always be YES

pdf_TextX (FUNCTION)

This function returns the current TextX position in the text space.

<u>Return type</u>
DECIMAL

<u>Sample call</u>
```
dX = pdf_TextX ("Spdf").
```

<u>Parameters</u>
- stream name (CHARACTER)

pdf_TextY (FUNCTION)

This function returns the current TextY position in the text space.

<u>Return type</u>
DECIMAL

<u>Sample call</u>
```
dY = pdf_TextY ("Spdf").
```

<u>Parameters</u>
- stream name (CHARACTER)

pdf_text_xy (PROCEDURE)

This procedure allows you to specifically place the passed text string at the passed X and Y position.

<u>Sample call</u>
```
RUN pdf_text_xy ("Spdf","Hello World",10,100).
```

<u>Parameters</u>
- stream name (CHARACTER)
- Text (CHARACTER) - Any string value.
- X value (INTEGER) - Any non-zero value
- Y value (INTEGER) - Any non-zero value

pdf_text_xy_dec (PROCEDURE)

This procedure allows you to specifically place the passed text string at the passed X and Y positions.

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

Sample call

```
RUN pdf_text_xy_dec ("Spdf","Hello World",10.0,100.0).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value.
- X value (DECIMAL) - Any non-zero value
- Y value (DECIMAL) - Any non-zero value

pdf_text_boxed_xy (PROCEDURE)

This procedure allows you to specifically place the passed text string at the passed X/Y position. A box will be placed around the text.

Sample call

```
RUN pdf_text_boxed_xy ("Spdf","Hello World",10,100,15,105,"LEFT",1).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value.
- X value (INTEGER) - Any non-zero value
- Y value (INTEGER) - Any non-zero value
- Box width (INTEGER) - Any non-zero value
- Box height (INTEGER) - Any non-zero value
- Justify (CHARACTER) - Alignment of the text within the box; one of "left", "right" or "center" - Implemented since version 4.2
- Weight (DECIMAL) - Weight of the box line

pdf_text_align (PROCEDURE)

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

- **LEFT** – starts placing the text at the passed X/Y coordinate
- **CENTER** – centers the text string over the X/Y coordinate
- **RIGHT** – places the end of the text string at the X/Y coordinate

Sample call

```
RUN pdf_text_align ("Spdf","Hello World","CENTER", 60, 100).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value
- Align (CHARACTER) - LEFT,CENTER,RIGHT
- X (INTEGER) - Any non-zero integer value
- Y (INTEGER) - Any non-zero integer value

pdf_text_center (PROCEDURE)

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

Sample call

```
RUN pdf_text ("Spdf","This is Centered Text",50,100).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value
- X (INTEGER) - Any non-zero integer value
- Y (INTEGER) - Any non-zero integer value

pdf_text_charxy (PROCEDURE)

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.

Sample call

```
RUN pdf_text_charxy ("Spdf","063",100,100).
```

Parameters

- stream name (CHARACTER)
- Value (CHARACTER) - Any character code (as an octal value between 0 and 377)
- X (INTEGER) - Any non-zero integer value
- Y (INTEGER) - Any non-zero integer value

pdf_wrap_text_xy (PROCEDURE)

This procedure allows you to place text into the PDF document within a box, defined by its X/Y location, width and height.
If the text is longer than the width then it will automatically wrap the text to the next line.
If the text doest not fit in the height, it will be truncated.

Sample call

```
RUN pdf_wrap_text_xy
    ("Spdf",
     "This is a lot of text that could include line breaks etc.",
     10, 20, /* X and Y */
     200, 100, /* Width and Height */
     pdf_PointSize("Spdf"),
     "left").
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value. Might contain carriage returns.
- X (INTEGER) - Any non-zero integer value
- Y (INTEGER) - Any non-zero integer value
- Width (INTEGER) - Any non-zero integer value
- Height (INTEGER) - Any non-zero integer value

- Skip (INTEGER) - Number of points to skip between lines. Usually pdf_PointSize().
- Alignment (CHARACTER) - how to align the text between the two columns: "left" or "right"

pdf_wrap_text_xy_dec (PROCEDURE)

Same as pdf_wrap_text_xy, with DECIMAL parameters.

Sample call

```
RUN pdf_wrap_text_xy_dec
    ("Spdf",
     "This is a lot of text that could include line breaks etc.",
     10, 20, /* X and Y */
     200, 100, /* Width and Height */
     pdf_PointSize("Spdf"),
     "left").
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value. Might contain carriage returns.
- X (DECIMAL) - Any non-zero decimal value
- Y (DECIMAL) - Any non-zero decimal value
- Width (DECIMAL) - Any non-zero decimal value
- Height (DECIMAL) - Any non-zero decimal value
- Skip (DECIMAL) - Number of points to skip between lines. Usually pdf_PointSize().
- Alignment (CHARACTER) - how to align the text between the two columns: "left" or "right"

Use HTML-like tags

All the procedures used to write text can use html-like tags for:
- <i>italics</i>
- bold
- <color=blue>color</color>
- <u>underline</u>
- <s>strike</s>
- <url=http://example.com>internet hyperlink</url>
- <url=#myBookmark>internal hyperlink to a bookmark</url>
- <font=myFont>some text using font myFont.

When using the *font* tag, the font - if not within the 14 base fonts - has to be loaded before using it, using pdf_load_font or pdf_load_font2.

It is possible to create hyperlinks internal to the current document, making them point to a bookmark, created with pdf_bookmark. The bookmark name must be prefixed by a "#" character.

In links (internet links or internal to the document), using a space can lead to unpredictable results. You should replace the spaces by "%20", e.g. <url=#my%20chapter>my chapter</url>.

This functionality is activated through the UseTags parameter:

```
RUN pdf_set_parameter(pdfStream,"UseTags","TRUE").
```

Some parameters have to be set to define the usable colors, for example:
```
RUN pdf_set_parameter(pdfStream, "TagColor:Black", "0,0,0").
RUN pdf_set_parameter(pdfStream, "TagColor:Red", "255,0,0").
RUN pdf_set_parameter(pdfStream, "TagColor:Green", "0,200,0").
RUN pdf_set_parameter(pdfStream, "TagColor:Blue", "0,0,255").
```
Any color name can be defined using this construct.

the default color:
```
RUN pdf_set_parameter(pdfStream, "DefaultColor", "Black").
```


the hyperlink color:
```
RUN pdf_set_parameter(pdfStream, "LinkColor", "144,12,144"). /* "0,0,255" (blue) by
```


and the different fonts:
```
RUN pdf_set_parameter(pdfStream, "BoldFont", "Helvetica-Bold").
RUN pdf_set_parameter(pdfStream, "ItalicFont", "Helvetica-Oblique").
RUN pdf_set_parameter(pdfStream, "BoldItalicFont", "Helvetica-BoldOblique").
RUN pdf_set_parameter(pdfStream, "DefaultFont", "Helvetica").
```


Then, every procedure which takes a string to be displayed as an argument will honor the tags to display the formatted text within the pdf file. For example:

```
RUN pdf_text("Spdf", "<b><color=green>This</color></b> is an "
            + "<url=http://example.com>example</url> "
            + "of <color=red>tags usage</color>!").
```


will give the following result:

This is an example of tags usage!

pdfInclude - README.pdf

Jicé

Miscelaneous

pdf_watermark (PROCEDURE)

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.

Sample call

```
RUN pdf_set_WaterMark
    ("Spdf",
     "Sample Report",
     "Courier", 16, /* Font, Font size */
     1.0, 0.0, 0,0, /* R,G,B */
     100, 500       /* X,Y */).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Any string value. Might contain carriage returns.
- Font (CHARACTER) - Name of Font to display Text in
- Font size (INTEGER) - How large the text should appear
- Red (DECIMAL) - Value to set Red option for text Colour
- Green (DECIMAL) - Value to set Green option for text Colour
- Blue (DECIMAL) - Value to set Blue option for text Colour
- X (DECIMAL) - X location to display text at
- Y (DECIMAL) - Y location to display text at

(c) 2015 snowwolf-software.com   Page 64 of 141

Templates

Templates are a way of defining easily simple pdf documents, or templates for documents (like an invoice template) to be filled later with some more calls to pdfInclude APIs.

See template.p in the samples directory.

Template syntax

Four elements are supported in templates:
- Text
- Images
- Rectangles
- Lines

Each element support properties, which are demonstrated below in the example.

<u>Example of a template file</u>:

```
# Text: Text String,Font Tag,X,Y,Point Size,Stroke RGB
# Image: Image Name,Image File,X,Y,Width,Height
# Rectangle: FromX,FromY,Width,Height,Stroke RGB,Fill RGB,Weight
# Line: FromX, FromY, ToX, ToY, Stroke RGB, Weight

# This displays the image at the top of the page
Image:Logo|samples/support/prosyslogo.jpg|100|720|175|40

# This section displays the invoice detail box (with darkened header)
Rectangle:100|200|400|400|0|0|0|1|1|1|0.5
Rectangle:100|585|400|15|0|0|0|0|0|0|0.5

# This section adds the separation lines to the invoice detail box
Line:300|200|300|585|0|0|0|0.5
Line:300|585|300|600|1|1|1|2
Line:400|200|400|585|0|0|0|0.5
Line:400|585|400|600|1|1|1|2

# This section displays the invoice detail text header
Text:DESCRIPTION|BF5|105|590|8|1|1|1
Text:QTY|BF5|305|590|8|1|1|1
Text:COST|BF5|405|590|8|1|1|1
```

pdf_load_template (PROCEDURE)

This procedure loads a template file within the current stream.
A template must be loaded only once.

Sample call

```
RUN pdf_load_template ("Spdf", "invTmpl", "templates/invoice.tpl").
```

Parameters

- stream name (CHARACTER)
- Template identifier (CHARACTER) - identifier used later to show the template
- Template file (CHARACTER) - file describing the template (see example above). Uses PROPATH since version 5.0.

---

pdf_use_template (PROCEDURE)

This procedure writes the previously loaded template within the pdf stream.
A template can be used many times within the same pdf document (on various pages).

Sample call

```
RUN pdf_use_template("Spdf", "invTmpl").
```

Parameters

- stream name (CHARACTER)
- Template identifier (CHARACTER) - identifier used to load the template

Graphic

Graphic state

See also pdf_rgb.

pdf_move_to (PROCEDURE)

This procedure allows you to dynamically change Graphic State locations within the PDF stream. It is also used to start a new sub-path.

Sample call

```
RUN pdf_move_to ("Spdf", 12, 12).
```

Parameters

- stream name (CHARACTER)
- X value (DECIMAL) - A non-zero decimal value representing a columnar location that you want to move the Graphic State cursor to
- Y value (DECIMAL) - A non-zero decimal value representing a row location that you want to move the Graphic State cursor to

  Note: starting with pdfInclude 5.0.8, the X and Y parameters are now DECIMAL instead of INTEGER.

pdf_set_GraphicX (PROCEDURE)

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.

Sample call

```
RUN pdf_set_GraphicX("Spdf",300).
```

Parameters

- stream name (CHARACTER)
- X value (INTEGER) - Value to set the Graphic X coordinate to

pdf_set_GraphicY (PROCEDURE)

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.

Sample call

```
RUN pdf_set_GraphicX("Spdf",300).
```

Parameters
- stream name (CHARACTER)
- Y value (INTEGER) - Value to set the Graphic Y coordinate to

pdf_GraphicX (FUNCTION)

This function returns the current column position (GraphicX) in the graphic space.

Return type
DECIMAL

Sample call

```
dX = pdf_GraphicX ("Spdf").
```

Parameters
- stream name (CHARACTER)

pdf_GraphicY (FUNCTION)

This function returns the current row position (GraphicY) in the graphic space.

Return type
DECIMAL

Sample call

```
dY = pdf_GraphicY ("Spdf").
```

Parameters
- stream name (CHARACTER)

pdf_stroke_fill (PROCEDURE)

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

Sample call

```
RUN pdf_stroke_fill ("Spdf",1.0,0.0,0.0).
```

Parameters
- stream name (CHARACTER)
- Red value (DECIMAL) - any value between 0 and 1
- Green value (DECIMAL) - any value between 0 and 1
- Blue value (DECIMAL) - any value between 0 and 1

pdf_set_FillRed (PROCEDURE)

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 (e.g.: 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)'.

<u>Sample call</u>

```
RUN pdf_set_FillRed ("Spdf",1.0).
```

<u>Parameters</u>

- stream name (CHARACTER)
- Red value (DECIMAL) - any value between 0 and 1

pdf_set_FillGreen (PROCEDURE)

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 (e.g.: 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)'.

<u>Sample call</u>

```
RUN pdf_set_FillGreen ("Spdf",1.0).
```

<u>Parameters</u>

- stream name (CHARACTER)
- Green value (DECIMAL) - any value between 0 and 1

pdf_set_FillBlue (PROCEDURE)

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 (e.g.: 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)'.

<u>Sample call</u>

```
RUN pdf_set_FillBlue ("Spdf",1.0).
```

<u>Parameters</u>

- stream name (CHARACTER)
- Blue value (DECIMAL) - any value between 0 and 1

pdf_FillRed (FUNCTION)

This function allows you to get the red component for the current fill graphic color.

Return type

DECIMAL between 0 and 1

Sample call

```
dRed = pdf_FillRed ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_FillGreen (FUNCTION)

This function allows you to get the green component for the current fill graphic color.

Return type

DECIMAL between 0 and 1

Sample call

```
dGreen = pdf_FillGreen ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_FillBlue (FUNCTION)

This function allows you to get the blue component for the current fill graphic color.

Return type

DECIMAL between 0 and 1

Sample call

```
dBlue = pdf_FillBlue ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_stroke_color (PROCEDURE)

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.

Sample call

```
RUN pdf_stroke_color ("Spdf",1.0,0.0,0.0).
```

Parameters

- stream name (CHARACTER)
- Red value (DECIMAL) - any value between 0 and 1
- Green value (DECIMAL) - any value between 0 and 1
- Blue value (DECIMAL) - any value between 0 and 1

---

pdf_set_dash (PROCEDURE)

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 (e.g.: On=1) then have a fairly large space where the line doesn't appear at all (e.g.: Off=10).

Sample call

```
RUN pdf_set_dash ("Spdf", 2, 2).
```

Parameters

- stream name (CHARACTER)
- Points On (INTEGER) - An integer value representing how many points to display for the dash
- Points Off (INTEGER) - An integer value representing how many points not to display for the gap

Examples

```
RUN pdf_set_dash ("Spdf", 5, 5).
```
- -

```
RUN pdf_set_dash ("Spdf", 10, 4).
```
— — — — — — — — — — — — — — — — —

```
RUN pdf_set_dash ("Spdf", 1, 1).
```
....

```
RUN pdf_set_dash ("Spdf", 0, 2). /* with line cap = 1 (rounded) */
```
....

pdf_set_dash_pattern (PROCEDURE)

Starting with pdfInclude 5.0, this procedure allows you to define exactly the dash pattern you want to use. It is a space separated list of integer pairs, representing the number of points "On" (dash) and the number of points "Off" (gap).

Sample call

```
RUN pdf_set_dash_pattern ("Spdf", "1 5 10 5", 0).
```

Parameters

- stream name (CHARACTER)
- dash pattern (CHARACTER) - a list-item pairs separated by space; each pair is composed of two integers representing the number of points "On" (dash) and the number of points "Off" (gap).
- dash phase (INTEGER) - number of points to skip from the pattern before beginning to draw the line

Examples

```
RUN pdf_set_dash_pattern ("Spdf", "1 5 10 5", 0).
```
.—.—.—.—.—.—.—.—.—.—.—.—.—.

```
RUN pdf_set_dash_pattern ("Spdf", "20 5 10 5", 0).
```
— — — — — — — — — — — — — —

pdf_set_linejoin (PROCEDURE)

This procedure allows you to define the Line Join Styles. This will typically be used when drawing a Rectangle or a Path (using pdf_move_to, pdf_path_add_segment and pdf_close_path).

Possible values are:

- 0 - Miter Join
- 1 - Round Join
- 2 - Bevel Join

<u>Sample call</u>

```
RUN pdf_set_linejoin ("Spdf", 1).
```

<u>Parameters</u>

- stream name (CHARACTER)
- Join style (INTEGER) - 0, 1 or 2

<u>Valid Join style values</u>

- 0 - Miter Join
- 1 - Round Join
- 2 - Bevel Join

<u>Examples</u>

```
RUN pdf_set_linejoin ("Spdf", 0).
```
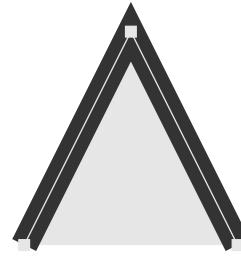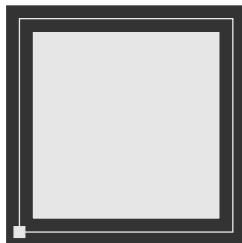
```
RUN pdf_set_linejoin ("Spdf", 1).
```
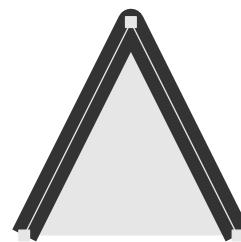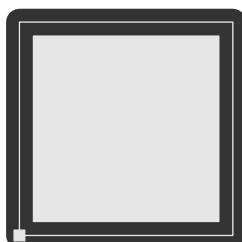
```
RUN pdf_set_linejoin ("Spdf", 2).
```
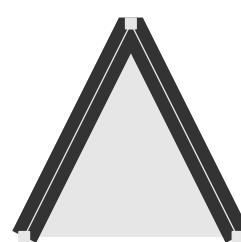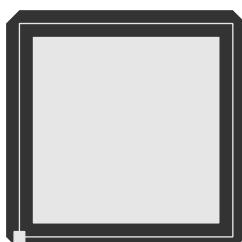
pdf_set_linecap (PROCEDURE)

This procedure allows you to define the Line Cap Styles. This will typically be used when drawing an open path.

Sample call

```
RUN pdf_set_linecap ("Spdf", 1).
```

Parameters

- stream name (CHARACTER)
- Join style (INTEGER) - 0, 1 or 2

Valid Line Cap style values

- 0 - Butt cap
- 1 - Round cap
- 2 - Projective square cap

Examples

```
RUN pdf_set_linecap ("Spdf", 0).
```
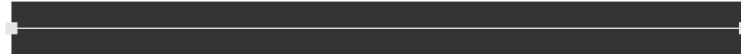


```
RUN pdf_set_linecap ("Spdf", 1).
```



```
RUN pdf_set_linecap ("Spdf", 2).
```


The start and end point of the line have been marked with small squares, so that the line cap is highlighted.

Drawing

You start a path by calling pdf_move_to. Then you add segments by successive calls to pdf_path_add_segment or curves, using pdf_curve. Finally, you close the path, using pdf_close_path.

---

pdf_path_add_segment (PROCEDURE)

This procedure adds a segment to a path, from the current graphic X/Y position, to a new X/Y position.

Sample call

```
RUN pdf_path_add_segment ("Spdf", 100, 100).
```

Parameters

- stream name (CHARACTER)
- X value (INTEGER) - X coordinate of the end of the segment
- Y value (INTEGER) - Y coordinate of the end of the segment

---

pdf_curve (PROCEDURE)

This procedure adds 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 pdf_stroke_color and pdf_stroke_fill.

Sample call

```
RUN pdf_curve ("Spdf",10,10,100,200,200,100,0.5).
```

Parameters

- stream name (CHARACTER)
- X1 (DECIMAL) - X1 coordinate - first control point
- Y1 (DECIMAL) - Y1 coordinate - first control point
- X2 (DECIMAL) - X2 coordinate - second control point
- Y2 (DECIMAL) - Y2 coordinate - second control point
- X3 (DECIMAL) - X3 coordinate - end of the curve
- Y3 (DECIMAL) - Y3 coordinate - end of the curve

Example

```
RUN pdf_move_to ("Spdf", 200, pdf_TextY("Spdf") - 40) /* start of the curve   */
RUN pdf_curve ("Spdf", 250, pdf_TextY("Spdf") - 5,    /* first control point   */
                       350, pdf_TextY("Spdf"),        /* second control point */
                       400, pdf_TextY("Spdf") - 45,1) /* end of the curve      */
RUN pdf_close_path ("Spdf").
```
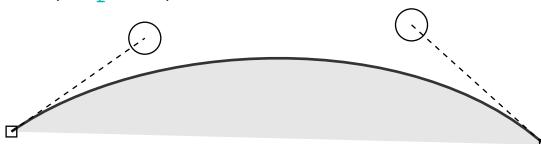

Note: The small circles above are the control points; the squares are the start and the end points.

---

pdf_close_path (PROCEDURE)

This procedure closes the previously drawn path. This will basically stroke and fill anything that has been drawn on the graphic plane.

Sample call

```
RUN pdf_close_path ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_close_path2 (PROCEDURE)

This procedure closes the previously drawn path. This will stroke anything that has been drawn on the graphic plane. It does not fill the path. Appeared in pdfInclude 5.0.

Sample call

```
RUN pdf_close_path2 ("Spdf").
```

Parameters

- stream name (CHARACTER)

---

pdf_rect (PROCEDURE)

This procedure draws a rectangle onto the current PDF page at the specified location using the specified width and height.
This procedure uses the colours set by pdf_stroke_color and pdf_stroke_fill.
The new Graphic X/Y location is set to the upper right corner of the rectangle.

Sample call

```
RUN pdf_rect ("Spdf", 10, 10, 200, 16, 0.5).
```

Parameters

- stream name (CHARACTER)
- X value (INTEGER) - X coordinate of the lower left point of the rectangle
- Y value (INTEGER) - Y coordinate of the lower left point of the rectangle
- Width (INTEGER) - Width of the rectangle
- Height (INTEGER) - Height of the rectangle
- Weight (DECIMAL) - Value representing how thick a line to draw

Example
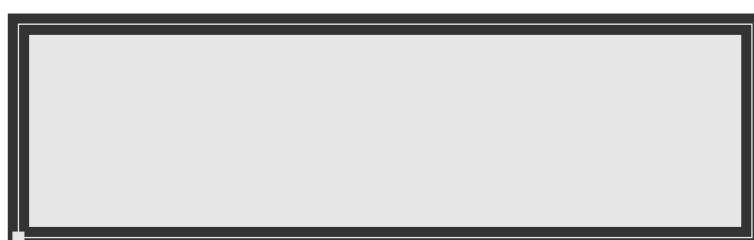
Note: the rectangle X/Y point has been marked using a small square.

---

pdf_rectdec (PROCEDURE)

Same call, with DECIMAL coordinates.

Sample call

```
RUN pdf_rect_dec ("Spdf", 10, 10, 200, 16, 0.5).
```

Parameters

- stream name (CHARACTER)
- X value (DECIMAL) - X coordinate of the lower left point of the rectangle
- Y value (DECIMAL) - Y coordinate of the lower left point of the rectangle
- Width (DECIMAL) - Width of the rectangle
- Height (DECIMAL) - Height of the rectangle
- Weight (DECIMAL) - Value representing how thick a line to draw

---

pdf_rect2 (PROCEDURE)

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.

The new Graphic X/Y location is set to the upper right corner of the rectangle.

Sample call

```
RUN pdf_rect2 ("Spdf", 10,10,200,16,0.5).
```

Parameters

- stream name (CHARACTER)
- stream name (CHARACTER)
- X value (INTEGER) - X coordinate of the lower left point of the rectangle
- Y value (INTEGER) - Y coordinate of the lower left point of the rectangle
- Width (INTEGER) - Width of the rectangle
- Height (INTEGER) - Height of the rectangle
- Weight (DECIMAL) - Value representing how thick a line to draw

Example

---

pdf_circle (PROCEDURE)

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 location after drawing the circle.

This procedure uses the colours set by pdf_stroke_color and pdf_stroke_fill.

Sample call

```
RUN pdf_circle ("Spdf",300,100,40,2.5).
```

Parameters

- stream name (CHARACTER)
- X value (DECIMAL) - X coordinate of the circle center
- Y value (DECIMAL) - Y coordinate of the circle center
- Radius (DECIMAL) - Radius of circle
- Weight (DECIMAL) - Value representing how thick to draw the circle border (stroke)

Example
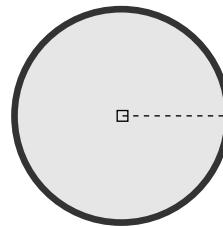
pdf_ellipse (PROCEDURE)

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

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

Sample call

```
RUN pdf_ellipse("Spdf", 250, 100, 60, 30, 0.35, 0.5).
```

Parameters

- stream name (CHARACTER)
- X value (DECIMAL) - X coordinate of the ellipse center
- Y value (DECIMAL) - Y coordinate of the ellipse center
- Major axis (DECIMAL) - value controlling the major axis
- Minor axis (DECIMAL) - value controlling the minor axis
- Eccentricity (DECIMAL) - Ellipse eccentricity - controls the oval shape of the ellipse
- Weight (DECIMAL) - Value representing how thick to draw the ellipse border (stroke)

Example
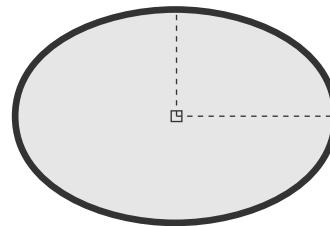
pdf_line (PROCEDURE)

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.

The new Graphic X/Y location is set to the end point.

Sample call

```
RUN pdf_line ("Spdf",10,10,10,600,2,0.5).
```

Parameters

- stream name (CHARACTER)
- From X (INTEGER) - X coordinate for the start of the line
- From Y (INTEGER) - Y coordinate for the start of the line
- To X (INTEGER) - X coordinate for the end of the line
- To Y (INTEGER) - Y coordinate for the end of the line
- Weight (DECIMAL) - Value representing how thick a line to draw

pdf_line_dec (PROCEDURE)

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.

The new Graphic X/Y location is set to the end point.

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.

Sample call

```
RUN pdf_line_dec ("Spdf",10.0,10.0,10.0,600.0,2.0,0.5).
```

Parameters

- stream name (CHARACTER)
- From X (DECIMAL) - X coordinate for the start of the line
- From Y (DECIMAL) - Y coordinate for the start of the line
- From X (DECIMAL) - X coordinate for the end of the line
- From Y (DECIMAL) - Y coordinate for the end of the line
- Weight (DECIMAL) - Value representing how thick a line to draw

<u>Examples</u>

Combinations with different line weights, caps and dashes. In the examples below,
dLeftMargin, dTextY, dPageWidth, dRightMargin are DECIMAL values, representing
respectively the value of pdf_LeftMargin, pdf_TextY, pdf_PageWidth, pdf_RightMargin.

```
RUN pdf_line ("Spdf",dLeftMargin + 100,              dTextY + 1,
              dPageWidth - dRightMargin - 100, dTextY + 1, 0.5).
```

The same with a dash defined to "4 4":
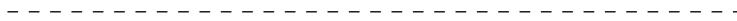
The same with a dash defined to "1 1":

```
RUN pdf_line ("Spdf",dLeftMargin + 100,              dTextY + 1,
              dPageWidth - dRightMargin - 100, dTextY + 1, 1).
```

```
RUN pdf_line ("Spdf",dLeftMargin + 100,              dTextY + 1,
              dPageWidth - dRightMargin - 100, dTextY + 1, 5).
```

For all the examples below, the instruction to draw the horizontal line is the same:

```
RUN pdf_line ("Spdf",dLeftMargin + 100,              dTextY + 1,
              dPageWidth - dRightMargin - 100, dTextY + 1, 6).
```

Before this call, the line cap and dash are set like written below.

```
RUN pdf_set_dash ("Spdf", 6, 12).
RUN pdf_set_linecap ("Spdf", 1).
```

```
RUN pdf_set_dash ("Spdf", 0, 20).
RUN pdf_set_linecap ("Spdf", 1).
```

```
RUN pdf_set_dash_pattern ("Spdf", "0 20", 17).
RUN pdf_set_linecap ("Spdf", 1).
```

This example and the previous one illustrate the "dash phase": the previous line started
20 points from its start point because of the dash set at zero. The "phase" in the second
one allows to skip 17 points, thus making the line start 3 points from its start point (the
gap of 3 points - half of the line weight - being filled by the rounded cap).

```
RUN pdf_set_dash_pattern ("Spdf", "10 20 0 20", 47).
RUN pdf_set_linecap ("Spdf", 1).
```

```
RUN pdf_set_dash ("Spdf", 1, 1).
RUN pdf_set_linecap ("Spdf", 0).
```

```
RUN pdf_set_dash ("Spdf", 6, 6).
RUN pdf_set_linecap ("Spdf", 0).
```

```
RUN pdf_set_dash_pattern ("Spdf", "6 12 18 12", 0).
RUN pdf_set_linecap ("Spdf", 0).
```

Images

pdfInclude supports various image formats:

- **JPEG**: preferred format for photographic work. Starting with version 4.1, CMYK & Gray scale JPEG pictures are supported.
- **PNG**: (starting with version 4.0) lossless, so preferred for graphics or line art. All bit depths supported; indexed and RGB pictures, full transparency and alpha layer included.
- **GIF**: (starting with version 5.0) only 8 bits, non interlaced GIF pictures are supported.
- **BMP**: (starting with version 5.0) all bit depths are supported; the alpha layer in 32 bits BMP is currently ignored; 16 bits BMP are supported (X-B-R-G = 1-5-5-5) but a bug in Acrobat X makes it to display strangely.

No external tool is used to extract information from the images and load them within the pdf file.

Example of a JPEG picture

pdf_load_image (PROCEDURE)

This procedure allows you to load and embed and external image. You can 'use' the image by using the pdf_place_image procedure.

Sample call

```
RUN pdf_load_image ("Spdf","PdfIncLogo","logo.jpg").
```

Parameters

- stream name (CHARACTER)
- Image Name (CHARACTER) - Unique Image Name (no spaces)
- File name (CHARACTER) - File name for the image. Uses PROPATH.

pdf_place_image (PROCEDURE)

This procedure places a previously loaded image onto the current PDF page at the specified location, with a specified size. If the size arguments are not specified (using the unknown value '?'), then the image size is used instead.

Note: there is also an (obsolete) API pdf_place_image2, maintained for backward compatibility only.

Sample call

```
RUN pdf_place_image ("Spdf","PdfIncLogo",200,16).
```

Parameters

- stream name (CHARACTER)
- Image Name (CHARACTER) - Unique Image Name created in pdf_load_image
- X value (DECIMAL) - X coordinate of the lower left point of the image
- Y value (DECIMAL) - Y coordinate of the lower left point of the image
- Width (INTEGER) - Width to use for the image. Starting with version 4.0, this can be the unknown value '?' in which case the image pixel width will be used.
- Height (INTEGER) - Height to use for the image. Starting with version 4.0, this can be the unknown value '?' in which case the image pixel height will be used.

pdf_ImageDim (FUNCTION)

This function returns the height or the width for a previously loaded image. If the image has not been loaded, it returns 0.

Return type

INTEGER

Sample call

```
iWidth = pdf_ImageDim ("Spdf", "MyPic", "WIDTH").
```

Parameters

- stream name (CHARACTER)
- image name (CHARACTER)
- dimension (CHARACTER) - can be "HEIGHT" or "WIDTH" according to which value you want to retrieve.

pdf_get_image_info (PROCEDURE)

This procedure gives the type, height and width of a previously loaded image. Appeared in version 4.0.

<u>Sample call</u>

```
RUN pdf_get_image_info ("Spdf", "PdfIncLogo",
                        OUTPUT cImgType,
                        OUTPUT iHeight, OUTPUT iWidth).
```

<u>Parameters</u>
- stream name (CHARACTER)
- Image Name (CHARACTER) - Unique Image Name created in pdf_load_image
- Image Type (CHARACTER) - Type of the image (one value among JPG,PNG,GIF, BMP)
- Image Height (INTEGER) - Height of the image in pixels
- Image Width (INTEGER) - Width of the image in pixels

Annotations (bookmarks, links, notes...)

pdf_link (PROCEDURE)

This procedure allows you to create a clickable rectangular zone, which is a hyperlink to an internet address, or an internal link to a bookmark.

This can be useful when used with linking a Customer ID to another document that lists all Customer Invoices. The clickable zone could also be placed around an image (such as a logo image) that once clicked, could redirect someone to your corporate web page.

Sample call

```
RUN pdf_link ("Spdf", 100, 100, 300, 300,
              "http://example.com",
              0.0, 0.0, 1.0, 1, "I").
```

Parameters

- stream name (CHARACTER)
- From X (INTEGER) - A non-zero integer value representing the X axis starting point
- From Y (INTEGER) - A non-zero integer value representing the Y axis starting point
- Link Width (INTEGER) - How wide should the link boundary be?
- Link Height (INTEGER) - How high should the link boundary be?
- Link Text (CHARACTER) - Text to appear when link is hovered over
- Text Red (DECIMAL) - Red value of RGB Colour
- Text Green (DECIMAL) - Green value of RGB Colour
- Text Blue (DECIMAL) - Blue value of RGB Colour
- Link Border (INTEGER) - Non-negative integer for the border width
- Link Style (CHARACTER) - Display style of link

Valid Style values

- **N** – No highlighting (blank)
- **I** – Invert the contents of the link
- **O** – Invert the links border
- **P** – Display a 'down' (or Pushed) appearance

pdf_Bookmark (PROCEDURE)

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.

Sample call

```
RUN pdf_bookmark ("Spdf","Customer 0001",0,NO,OUPUT iBookmark).
```

Parameters

- stream name (CHARACTER)
- Title (CHARACTER) - Bookmark Description

- Parent (INTEGER) - 0 or valid Bookmark number. This a bookmark number from a previously created bookmark.
- Expand (LOGICAL) - Expand this bookmark (if parent) – Yes/No
- Bookmark number (INTEGER) - OUTPUT parameter to give back the number of the created bookmark. Useful to parent bookmarks.

pdf_note (PROCEDURE)

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

The note usually appears as an icon, with the text in a tool-tip which appears on mouse hover.

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 sample call creates the note you can see on the left of the sample call.

Sample call

```
RUN pdf_note("Spdf","My Note Text","My Note Title",
            "Note",              /* Icon */
            30.0,40.0,50.0,60.0, /* Coordinates & size */
            1.0,0.0,0.0          /* Note color */ ).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Text to appear in note
- Title (CHARACTER) - Small title for the note header
- Icon (CHARACTER) - Predefined icon set (see below); the character case is important, you have to respect the case of the list given below. Starting with version 5.0, pdfInclude will automatically force the correct character case before writing the note into the pdf file.
- LLX (DECIMAL) - Lower Left X Coordinate
- LLY (DECIMAL) - Lower Left Y Coordinate
- URX (DECIMAL) - Upper Right X Coordinate
- URY (DECIMAL) - Upper Right Y Coordinate
- Red (DECIMAL) - decimal value representing Red colour value
- Green (DECIMAL) - decimal value representing Green colour value
- Blue (DECIMAL) - decimal value representing Blue colour value

Valid icon types

- Note (default)
- Comment
- Insert
- Key
- Help
- NewParagraph
- Paragraph

The icons may differ based on viewer being used.

pdf_stamp (PROCEDURE)

This procedure allows you to add a Stamp at a specific location (actually an area determined by the Lower Left and Upper Right X/Y coordinates) on the current PDF page.
The stamp can contain as much as 32K of associated text.
The stamps icon may differ based on viewer being used.
The sample call creates the stamp you can see on the left of the sample call.

Sample call

```
RUN pdf_stamp("Spdf","My Stamp Text","My Stamp Title",
              "Approved",
              40.0,20.0,70.0,40.0,
              0.0,1.0,0.0).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Text to appear in note
- Title (CHARACTER) - Small title for the note header
- Stamp (CHARACTER) - Predefined icon set (see below); the character case is important, you have to respect the case of the list given below. Starting with version 5.0, pdfInclude will automatically force the correct character case before writing the note into the pdf file.
- LLX (DECIMAL) - Lower Left X Coordinate
- LLY (DECIMAL) - Lower Left Y Coordinate
- URX (DECIMAL) - Upper Right X Coordinate
- URY (DECIMAL) - Upper Right Y Coordinate
- Red (DECIMAL) - decimal value representing Red colour value
- Green (DECIMAL) - decimal value representing Green colour value
- Blue (DECIMAL) - decimal value representing Blue colour value

Valid Stamp types

- Draft (default)
- Approved
- Experimental
- NotApproved
- AsIs
- Expired
- NotForPublicRelease
- Confidential
- Final
- Sold
- Departmental
- ForComment
- TopSecret
- ForPublicRelease

pdf_Markup (PROCEDURE)

This procedure allows you to add a Style and a Tool-tip 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 sample call creates the yellow highlight on the title of this paragraph.

Sample call

```
RUN pdf_markup("Spdf","My Markup Text",
               "My Markup Title","Highlight",
               100.0,540.0,300.0,540.0,100.0,554.0,250.0,554.0,
               1.0,1.0,0.0).
```

Parameters

- stream name (CHARACTER)
- Text (CHARACTER) - Text to appear in markup content
- Title (CHARACTER) - Small title for the markup content
- Style (CHARACTER) - Predefined style set (see below); the character case is important, you have to respect the case of the list given below. Starting with version 5.0, pdfInclude will automatically force the correct character case before writing the text markup annotation into the pdf file.
- X1 (DECIMAL) - Lower Left X Coordinate
- Y1 (DECIMAL) - Lower Left Y Coordinate
- X2 (DECIMAL) - Lower Right X Coordinate
- Y2 (DECIMAL) - Lower Right Y Coordinate
- X3 (DECIMAL) - Upper Left X Coordinate
- Y3 (DECIMAL) - Upper Left Y Coordinate
- X4 (DECIMAL) - Upper Right X Coordinate
- Y4 (DECIMAL) - Upper Right Y Coordinate
- Red (DECIMAL) - decimal value representing Red colour value
- Green (DECIMAL) - decimal value representing Green colour value
- Blue (DECIMAL) - decimal value representing Blue colour value

Valid Markup Styles

- Highlight (default)
- Underline
- Squiggly (squiggly or jagged underline)
- StrikeOut

Transaction mechanism

Starting with version 4.1, the transaction mechanism allows to undo some additions done to the current pdf stream based on any condition.

For example, it is used in the program generating this very document (readme.p) in order to make titles sticky to paragraphs:

- when a title is encountered, a transaction is started, then the title is written to the pdf
- when writing the next paragraph, if it is not located on the same page, then we undo the transaction, create a new page, and start over. Else we commit the transaction. Thus, if we were to have a page break between the title and its first paragraph, both title and its first paragraph will be displayed on the next page.

It is also used in the table tool (pdftool.p) to ensure no page break happens within a table line, and in the matrix tool, to buffer the text, draw the cell's rectangle, then commit the text, so that it is written on top of the rectangle.

A transaction is started with 'pdf_transaction_begin'. Then you call some pdfInclude APIs to display text and graphical elements. Then you commit with 'pdf_transaction_commit' or cancel with 'pdf_transaction_rollback'.

The transaction mechanism can also be used to "buffer" text in order to write it later to the pdf, using 'pdf_transaction_buffer'. In this case, you call 'pdf_transaction_begin', then when you are done with outputting the text and/or graphical elements you need, you call 'pdf_transaction_buffer'. Then you can output more elements to the pdf file. Finally, you output the buffered elements, calling 'pdf_transaction_commit'.

pdf_transaction_begin (PROCEDURE)

Start a transaction.

Sample call

```
RUN pdf_transaction_begin ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_transaction_rollback (PROCEDURE)

Cancel (roll back) a transaction.

Sample call

```
RUN pdf_transaction_rollback ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_transaction_buffer (PROCEDURE)

Buffer a transaction = remember for later.

Using this procedure you can output text, buffer it (i.e. do not write it to the pdf file yet), then output some more objects to the pdf file, then commit the transaction.

This is useful for example when you need to know the size of the text (number of lines) outputted via pdf_wrap_text_xy, then you can write a filled rectangle in order to print the

text above it. The text needs to be be printed after the rectangle, else it would be below the rectangle, and as thus, hidden.

Sample call

```
RUN pdf_transaction_buffer ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_transaction_commit (PROCEDURE)

write the transaction contents to the pdf

Sample call

```
RUN pdf_transaction_commit ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_in_transaction (FUNCTION)

Returns YES when a transaction is pending (between start and commit), else returns NO.

Return type

LOGICAL

Sample call

```
IF pdf_in_transaction("Spdf") THEN ...
```

Parameters

- stream name (CHARACTER)

Reusable patterns (pdf Xobjects)

Starting with version 4.2, you can define pieces of pdf that can be reused many times, on the same page or on different pages. This allows to repeat many times the exact same piece of pdf, while keeping the pdf file smaller.

A pattern can contain text, pictures or any graphical elements specified by pdf. In the pdf specifications, such reusable patterns are called "Xobjects".

In order to define a pattern, first call pdf_pattern_begin, then issue calls to any pdfInclude APIs to display text, embed a picture or draw a graphic, then once done, call pdf_pattern_end. Later, call pdf_pattern_use each time you need to use the pattern.

You can define as much patterns as you need.

The default header makes use of the reusable patterns.

pdf_pattern_begin (PROCEDURE)

This procedure tells pdfInclude that all the next API calls will display elements to be embedded into a pattern.

Sample call

```
RUN pdf_pattern_begin("Spdf", OUTPUT iXObjectId,
                      pdf_PageWidth(pdfStream),
                      pdf_TopMargin(pdfStream),
                      "myHeader").
```

Parameters

- stream name (CHARACTER)
- XObject ID (INTEGER) - OUTPUT parameter giving the identifier of the pattern (to be reused later in pdf_pattern_use)
- Width (INTEGER) - width of the pattern. Once within the pattern, pdf_PageWidth() will return this value.
- Height (INTEGER) - height of the pattern. Once within the pattern, pdf_PageHeight() will return this value.
- key (CHARACTER) - alternate identifier, allowing you to find your pattern later, in case you don't have the integer identifier handy, using pdf_pattern_search_by_key().

pdf_pattern_end (PROCEDURE)

Tells pdfInclude that you are done with the current pattern creation. From now on you can use the pattern with pdf_pattern_use.

Sample call

```
RUN pdf_pattern_end ("Spdf").
```

Parameters

- stream name (CHARACTER)

pdf_pattern_use (PROCEDURE)

Write the pattern into the pdf document.

Sample call

```
RUN pdf_pattern_use ("Spdf", iXObjectId,
                      0,
                      pdf_PageHeight(pdfStream) - pdf_TopMargin(pdfStream),
                      ?, ?).
```

Parameters

- stream name (CHARACTER)
- XObject ID (INTEGER) - identifier as returned by pdf_pattern_begin
- X value (DECIMAL) - X position where to display the pattern
- Y value (DECIMAL) - Y position where to display the pattern
- Width (DECIMAL) - Width to use to display the pattern. Can be the unknown value '?', in which case the width of the pattern as defined in pdf_pattern_begin will be used; can also be a different value, in which case the pattern will be expanded or shrinked to fit the value.
- Height (DECIMAL) - Height to use to display the pattern. Can be the unknown value '?', in which case the height of the pattern as defined in pdf_pattern_begin will be used; can also be a different value, in which case the pattern will be expanded or shrinked to fit the value.

pdf_pattern_search_by_key (FUNCTION)

Returns the integer identifier of a pattern, using the alternate key to find it.

Return type

INTEGER

Sample call

```
iXObjectId = pdf_pattern_search_by_key("Spdf", cSearchKey).
```

Parameters

- stream name (CHARACTER)
- key (CHARACTER) - alternate identifier for which you want to get the integer identifier

Using an external pdf file

pdfInclude allows you to open an existing pdf file and reuse it into your pdfInclude generated file.

It is 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.

It is also very useful to use an existent pdf as a template, then overlay data onto the document.

When using external pdf files, the following parameters are used:

- UseExternalPageSize: TRUE/**FALSE** - the page size from the external pdf file will be used for the generated page
- reuseExternal: TRUE/**FALSE** - do not delete external temporary files & temp-tables when calling pdf_close. This allows to save a lot of processing time when generating a lot of documents which use the same pdf template: set this parameter to YES, for the first one, call pdf_open_pdf, then do not call it again for all the remaining documents. This allows to perform the external pdf file parsing only once, thus saving a lot of processing time. Appeared in pdfInclude 4.2.
- usePdfCache: **TRUE**/FALSE - makes use of the external pdf files cache (starting with version 5.1). This is a great optimization, as the external pdf file is now parsed only once, cache files being written in SESSION:TEMP-DIRECTORY/pdfcache. If it is reused later, from the same or another program, then the cache will be used instead of parsing the pdf file. However pdfInclude does not detect if the external pdf file has changed (the identifier for the cache being only the file path). In such a case, the developper should call pdf_clear_pdf_cache before pdf_open_pdf.
- parameters related to pdf forms:
  - formFlatten: CAN-DO list of widgets to flatten (default value "*"). If you want to fill a PDF form, while keeping all the widgets, set it to "" (empty string).
  - formFlattenWithDefaultValues: can-do list of widget names for which to retain the default value when flattening the form
  - retainAnnots list annotations subtypes. Annotations which subtype is in the list will be kept in the generated pdf. The list elements are to be taken from the following list: Text, Link, FreeText, Line, Square, Circle, Polygon, PolyLine, Highlight, Underline, Squiggly, StrikeOut, Stamp, Caret, Ink, Popup, FileAttachment, Sound, Movie, Widget, Screen
- debug parameter:
  - drawFormFieldRect: 3 color values separated by a coma, e.g. "1,0,0". If defined, pdfInclude will draw a box around the form fields.

Optimization

Parsing a pdf file is costly, and might use a good percentage of the total generation time. In order for this time to be the smallest possible, pdfInclude performs some optimization:

- When a pdf file has been parsed for stream e.g. "Spdf1", and the program asks to parse it again for "Spdf2" (before closing "Spdf1"), then instead of parsing it again, it will be reused directly (starting with version 5.1)
- When program generates various pdf files using the same template with the same ID within the same session, using the "reuseExternal" parameter makes pdfInclude remember the parsed data, and reuse it subsequently (starting with version 5.0)
- Finally, using the parameter "usePdfCache" makes the parsed data persistent between sessions (disk cache), thus making the global generation much faster.

It is better - if possible for your use case - to use the first two methods, as it is faster (memory to memory copy) than the disk cache. In any case, the use of the pdf cache is

highly recommended (since version 5.1), as it it way faster than parsing the external pdf file.

pdf_open_PDF (PROCEDURE)

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).

Starting with pdfInclude 5.0, pdf_open_PDF searches the external file in the PROPATH.

Starting with pdfInclude 5.1, pdf_open_PDF searches for the external file cache before parsing it, according to the value of the "usePdfCache" parameter. If it is found, then the contents of the external pdf file is loaded from the cache, instead of parsing it.

Sample call

```
RUN pdf_open_pdf ("Spdf","c:\templates\myInvoice.pdf","INV").
```

Parameters

- stream name (CHARACTER)
- PDF Name (CHARACTER) - PDF File Name and Path (must exist)
- PDF ID (CHARACTER) - Unique PDF Identifier (e.g.: Invoice). This call defines the PDF identifier.

pdf_use_PDF_page (PROCEDURE)

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

For pdfInclude versions less or equal to 4.0, 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). Starting with version 4.1, it is possible to embed an existing page many times from each new page. These pages may come from different existing pdf files, or from different pages from the same existing pdf file.

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.

The text position is reset to the top of page. When the parameter "UseExternalPageSize" has been set to "TRUE", the document page size is updated to that of the external pdf.

Sample call

```
RUN pdf_use_pdf_page ("Spdf", "INV", 1).
```

Parameters

- stream name (CHARACTER)
- PDF ID (CHARACTER) - Unique PDF Identifier
- Page (INTEGER) - Number of the page from the existing pdf to use

pdf_place_pdf_page (PROCEDURE)

This procedure, which appeared in pdfInclude 5.0, is similar to pdf_use_PDF_page, with an extra "options" parameter, allowing to rotate, scale and position the external pdf document within the page. It is equivalent to pdf_place_image, but for pdfs.

Sample call

```
RUN pdf_place_pdf_page ("Spdf", "INV", 1, "Rotate=90,Scale=0.25").
```

Parameters

- stream name (CHARACTER)
- PDF ID (CHARACTER) - Unique PDF Identifier
- Page (INTEGER) - Number of the page from the existing pdf to use
- Options (CHARACTER) - options list. The list can of course be left empty. Currently supports the following options:
  - *Rotate* to rotate the template within the document page (decimal angle in degrees)
  - *Scale* to scale the template within the document page (decimal scale). This can be useful e.g. to create a pdf document with thumbnails created from other pdf documents.
  - *X*: position on the X axis where to position the external pdf document left-bottom corner (before rotation). If not present, it will be horizontally centered within the page.
  - *Y*: position on the Y axis where to position the external pdf document left-bottom corner (before rotation). If not present, it will be vertically centered within the page.
  - *Background*: as most of the pdf files have no background, they result in being transparent. This parameter allows you to set a solid background. It is a R,G,B triplet (decimals between zero and one). Default: no background.
  - *Border*: this parameter allows you to draw a border around the pdf page. It is a R, G,B triplet (decimals between zero and one).
  - *BorderWeight*: this parameter defines the border weight. Default: no border.
  - *UsePdfPage*: allow pdf_place_pdf_page to behave like pdf_use_pdf_page (page size, reset text position, use form fields)

Example

The two thumbnails below are generated using the following code:

```
RUN pdf_open_pdf ("Spdf","samples/support/POFormLOo.pdf","ext").
RUN pdf_place_pdf_page ("Spdf","ext",1,"Scale=0.3,X=120,Y=86").
RUN pdf_place_pdf_page ("Spdf","ext",1,"Rotate=-12,Scale=0.3,X=330,Y=106").
```
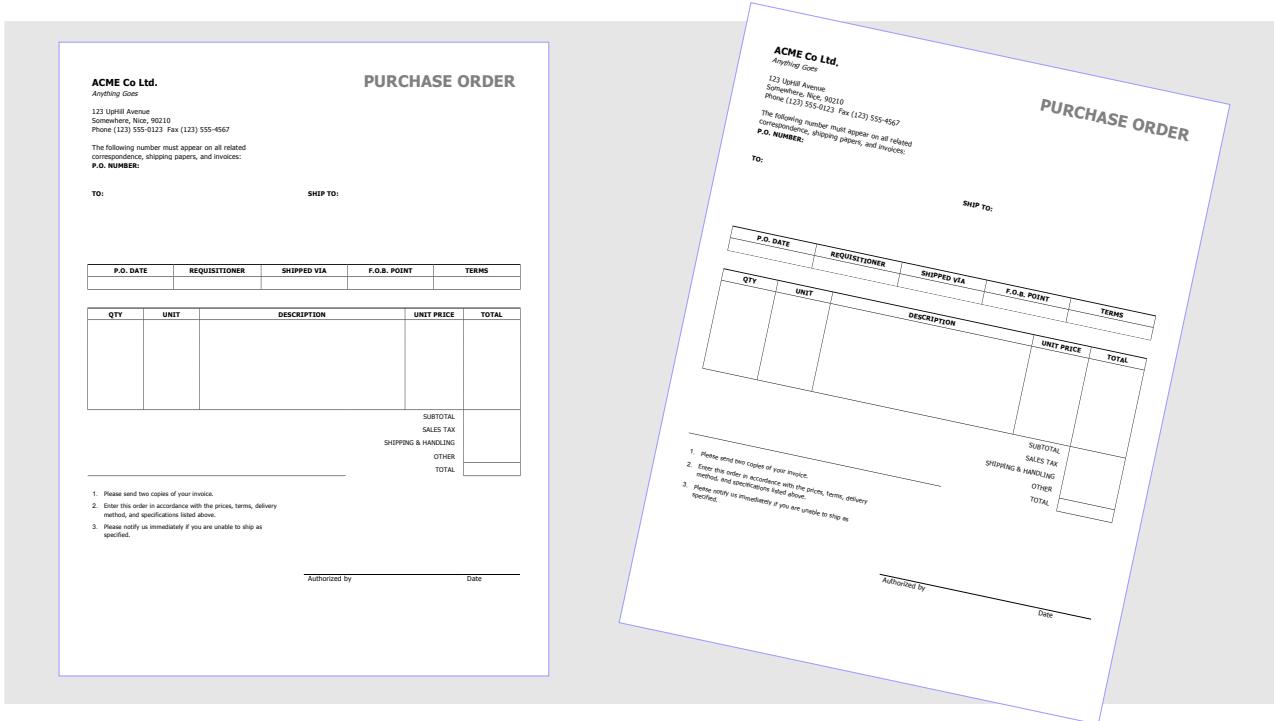

pdf_fill_text (PROCEDURE)

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

If you've opened an 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. See below for a description of the possible field options.

Before version 4.0, only text fields (fill-ins) were supported. Starting with version 4.0, pdfInclude supports filling of:

- fill-ins
- radio buttons
- check boxes
- combo boxes
- lists

Also starting with version 4.0, pdfInclude uses the font, font size, alignment & multiline flag defined in the pdf template, unless specified a different value through the Field options.

Sample call

Fill a text field, using default alignment and font (as defined in the template):
```
RUN pdf_fill_text ("Spdf", "InvoiceNumber","00001","").
```
Fill a text field, forcing right alignment and font:
```
RUN pdf_fill_text ("Spdf", "InvoiceNumber","00001","align=right,font=Helvetica").
```

Tick a checkbox:
```
RUN pdf_fill_text ("Spdf", "FOB","YES","").
```
Tick a radio set (where "choice1" is the value defined in the template):
```
RUN pdf_fill_text ("Spdf", "FOB","choice1","").
```
Select the third entry in a combo box:
```
RUN pdf_fill_text("Spdf", "ComboBox1","3","").
```
Fill the combo box with a centered free text (not defined in the templateas a possible value for the combo):
```
RUN pdf_fill_text("Spdf", "ComboBox2","Free text","text,align=center").
```
Select various entries in a list box:
```
RUN pdf_fill_text("Spdf", "ListBox3","6,8,12","").
```

Parameters

- stream name (CHARACTER)
- Form Field Name (CHARACTER) - Name of a field in the pdf form
- Field Value (CHARACTER) - Text to be written in the field (for fill-ins) or value to be set to the widget.
- Field Options (CHARACTER) - Field options

Valid Field Options

Field Options is a coma separated list of key=value pairs.

- align: LEFT/CENTER/RIGHT - The 'Align' option allows you to programmatically set how you want the field contents aligned when the field value is displayed, instead of using the alignment defined in the pdf template.
- multiline: YES/NO - 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. Usually the multiline option is defined directly in the pdf template, but you can override it using this option.

Starting with version 4.1, the following options are available:

- font: name of a font known to pdfInclude - if specified, will replace the font defined in the field.
- fontSize: (DECIMAL) font size to fill the field with - if specified, will replace the font size defined in the field.
- text: YES/NO - for combo widgets, tells if the value is free text (YES), or the index number of a predefined value of the combo.

pdf_clear_pdf_cache (PROCEDURE)

This procedure (starting with version 5.1) clears the cache for a given pdf file.

Sample call

```
RUN pdf_clear_pdf_cache ("Spdf", "c:\templates\myInvoice.pdf").
```

Parameters

- stream name (CHARACTER)
- pdf name (CHARACTER): external pdf file path

parseText (FUNCTION)

This function parses a string directly issued from an existent pdf file into a normal string, stripping extra characters and performing the needed conversions.

Return type

CHARACTER

Sample call

```
cValue = parseText (cMyPdfString).
```

Parameters

- stream name (CHARACTER)
- pdf string (CHARACTER): a string extracted from an existent pdf file (such as a widget's value or name, or a string exported using the query APIs, see below).

pdf_get_widgets (PROCEDURE)

This procedure returns information about form widgets. This is useful for debugging an external pdf form, or to know the names of the widget of such a form to be able to fill them, using pdf_fill_text.

Sample call

```
RUN pdf_get_widgets ("Spdf", OUTPUT cWidgets).
```

Parameters

- stream name (CHARACTER)
- widgets (CHARACTER), OUTPUT - a CHR(1) delimited list with information about the form fields and widgets. Each element of the list is itself a CHR(2) delimited list, containing:
  - widget name,
  - widget page,
  - widget type,
  - widget rectangle,
  - widget value,
  - widget values (for multiple values widgets like lists).

Querying an existing pdf file

Introduction to pdf query

Prior to version 4.1, the only way to get information from a pdf file was to use:

pdf_get_pdf_info (FUNCTION)

This function allows you to get the values from the external pdf document properties (Author, Title, Subject, Keywords...).

Return type

CHARACTER

Sample call

```
cAuthor = pdf_get_pdf_info("Spdf", "INV", "Author").
```

Parameters
- stream name (CHARACTER)
- PDF ID (CHARACTER) - Unique PDF Identifier
- property name (CHARACTER) - one of:
  - pdf properties: author, title, subject, keywords, creator, producer, modDate, modTime, creationDate, creationTime
  - calculated properties: pages

Starting with version 4.1, pdfInclude allows you to query an existing pdf file. This can be useful for example to open an existing pdf file and fetch some information from it (like title, author, number of pages).

It is also possible to perform the queries without creating a new document: pdf_new accepts the unknown value '?' as an argument for the file name, in which case no file will be generated.

The "pdf path"

In order to query a pdf file, you have to determine the "pdf path" to query (this requires some knowledge of the pdf format, or just open your pdf in a notepad and take inspiration in the examples below).

A "pdf path" is the way to describe how to find the data within the file. As pdf is structured using scalar values (numbers, strings) and dictionaries (which can contain named scalar values or named dictionaries or named arrays) and arrays (array of scalar, dictionaries or arrays), it is possible to define a path representing the way we travel in the pdf file structure to fetch the data.

Example of a pdf path: "/Root/Pages/Count"

There are two starting point objects:
- /Info: the information dictionary, containing the definition of the Author, Title...
- /Root: the root of the document, starting point for any other information.

The pdf path is a list of pdf names (names of the pairs within the dictionaries) separated by "/". Array elements are accessed through the classical array notation [n], e.g.

"/Kids[3]" meaning the third element of the array named "/Kids".

If you open a pdf file using a notepad, a typical Info dictionary will look this way:

```
1 0 obj
<<
 /Author (Jicé)
 /CreationDate (D:20150306110837+01'00)
 /Producer (pdfInclude v5.0)
 /Title (Example pdf document)
>>
endobj
```

Thus, the path to query the author of the document will be "/Info/Author".

In a notepad, a typical Root dictionary will look this way:

```
2 0 obj
<<
 /Type /Catalog
 /Pages 3 0 R    % This is a pointer to the pages object (see below)
 /PageMode /UseNone
 /PageLayout /SinglePage
>>
endobj
```

The "3 0 R" notation above is like a pointer to object number 3, which name is "/Pages", and can be found elsewhere in the pdf file. The path to this object will be "/Root/Pages". The /Pages array looks this way:

```
3 0 obj
<<
 /Type /Pages
 /Count 7
 /Kids [ % Pointers to each page object
  175 0 R
  178 0 R
  181 0 R
  184 0 R
  187 0 R
  190 0 R
  193 0 R
 ]
>>
endobj
```

In the Kids array, you can see 7 pointers to each of the pages. The second page object (which path is "/Root/Pages/Kids[2]") is:

```
178 0 obj
<<
 /Type /Page
 /Parent 3 0 R
 /Resources 174 0 R
 /Rotate 90
 /MediaBox [0 0 595.276 841.89]
 /CropBox [0 0 595.276 841.89]
 /Contents 179 0 R
>>
endobj
```

Finally, if you would like to know what rotation has been applied to the second page, you would use the following path: "/Root/Pages/Kids[2]/Rotate", which would give you the value "90".

Example

```
{pdf_inc.i "THIS-PROCEDURE"}
DEFINE VARIABLE cFile        AS CHARACTER    NO-UNDO.
DEFINE VARIABLE iPageCount AS INTEGER       NO-UNDO.
/* ... more variables definitions ... */
cFile = "path/to/the/file.pdf".
RUN pdf_new("s", ?).
RUN pdf_open_pdf("s", cFile, "ext").
RUN pdf_ext_get_nb_pages("s", "ext", OUTPUT iPageCount).
RUN pdf_ext_get_path("s", "ext", ?, "/Info/Author",
                     OUTPUT cPath, OUTPUT cType, OUTPUT cValueAuthor).
RUN pdf_ext_get_path("s", "ext", ?, "/Root/Pages",
                     OUTPUT cPath, OUTPUT cType, OUTPUT cValuePages).
RUN pdf_ext_get_path("s", "ext", ?, "/Root/Pages/Kids",
                     OUTPUT cPath, OUTPUT cType, OUTPUT cValueKids).
RUN pdf_ext_get_path("s", "ext", ?, "/Root/Pages/Kids[2]",
                     OUTPUT cPath, OUTPUT cType, OUTPUT cValueKids2).
RUN pdf_close("s") NO-ERROR.
MESSAGE "Pages:" iPageCount SKIP
    "Author:" cValueAuthor "-" parseText(cValueAuthor) SKIP
    "Pages:" cValuePages SKIP
    "Pages/Kids:" cValueKids SKIP
    "Pages/Kids[2]:" cValueKids2
    VIEW-AS ALERT-BOX INFO BUTTONS OK.
```

This will display something similar to:

Pages: 7

Author: (þÿ\000J\000i\000c\000é) - Jicé

Pages: DICT#5 /Count 7 /Kids ARRAY#1 /Type /Pages

Pages/Kids: ARRAY#1 DICT#43 DICT#45 DICT#47 DICT#49 DICT#51 DICT#53 DICT#55

Pages/Kids[2]: DICT#45 /Contents DICT#46 /CropBox ARRAY#41 /MediaBox ARRAY#40 /Parent DICT#5 /Resources DICT#40 /Rotate 90 /Type /Page

pdf_ext_get_path might return 3 types of data:

- a scalar value: it is a simple value, like the number of pages or the author of the document (in the document above, it is Unicode encoded, hence the strange result at first, decoded by the parseText() function).
- a dictionary: in a pdf, a dictionary is a set of key/value pairs. Here, pdfInclude represents it as a CHR(1) separated list, the first element being its identifier (DICT#5), and the following ones the list pairs, separated with a space character: "/Name object".
- an array: in a pdf, an array is an ordered table of elements. Here, pdfInclude represents it as a CHR(1) separated list, the first element being its identifier (ARRAY#1), and the following ones the array elements (which in this case are dictionaries, represented through their identifier). To access an array element in the pdf path,

pdf_ext_get_path (PROCEDURE)

This procedure allows to get the value or object pointed by a pdf path.

Sample call

See above

Parameters

- stream name (CHARACTER)
- pdf ID (CHARACTER) - ID of the existing pdf, opened using pdf_open_pdf.
- start point (CHARACTER)
  - '?' (unknown value) when the path starts with "/Root" or "/Info"
  - a dictionary identifier (such as DICT#12) as a starting point for the pdf path
- pdf path (CHARACTER) - pdf path to follow in order to find the value. If the start point is '?' then it must start with one of the two top level pdf objects: "/Root" or "/Info".
- found path (CHARACTER) - OUTPUT: if the value has been found, will be equal to the input pdf path. If the value is not found, this value will be the path until the next value was not found.
- found type (CHARACTER) - OUTPUT: one among SCALAR,ARRAY,DICT. '?' if not found.
- found value (CHARACTER) - OUTPUT: found value (see above for a description of the format). '?' if not found.

pdf_ext_get_page (PROCEDURE)

This procedure is a shortcut for "/Root/Pages/Kids[n]". It returns a Page dictionary.

Sample call

```
RUN pdf_ext_get_page ("Spdf", "ext", 2, OUTPUT cPageDictID).
```

Parameters

- stream name (CHARACTER)
- pdf ID (CHARACTER) - ID of the existing pdf, opened using pdf_open_pdf.
- page number (INTEGER)
- page dictionary (CHARACTER) - OUTPUT: the page dictionary. '?' if not found.

pdf_ext_get_nb_pages (PROCEDURE)

This procedure is a shortcut for "/Root/Pages/Count". It returns the count of pages of the external pdf document.

Sample call

```
RUN pdf_ext_get_nb_pages("s", "ext", OUTPUT iPageCount).
```

Parameters

- stream name (CHARACTER)
- pdf ID (CHARACTER) - ID of the existing pdf, opened using pdf_open_pdf.
- page count (INTEGER) - OUTPUT: number of pages in the external pdf document.

Using XML as an input file

pdfInclude provides some calls to load an xml file, and access nodes' values. This is useful e.g. to read an xml file containing invoice data and process it to generate the corresponding documents.

See xml.p in the samples/super directory for an example.

pdf_load_xml (PROCEDURE)

This procedure allows you to load an XML file. The nodes' data can then be accessed by referencing directly the TT_pdf_xml TEMP-TABLE. Then you can use any of pdfInclude calls or tools to generate the corresponding pdf chunk.

Sample call

```
RUN pdf_load_xml ("Spdf", "/path/to/file.xml").
```

Parameters

- stream name (CHARACTER)
- xml file (CHARACTER)

GetXMLNodeValue (FUNCTION)

This function returns the value of a node (after having loaded the xml file using pdf_load_xml). The node is accessed using its parent's xml path, and the node name. The parent xml path is a "/" separated list of node names, leading from the xml file root to the parent node.

For example, the sample call below, having loaded the following xml file, will return the string "ACME Corp.".

```
<Invoice>
  <HeaderInfo>
    <Company>ACME Corp.</Company>
    ...
  </HeaderInfo>
  ...
</Invoice>
```

Return type

CHARACTER

Sample call

```
RUN pdf_text("Spdf", GetXMLNodeValue("/Invoice/HeaderInfo","Company")).
```

Parameters

- parent XML path (CHARACTER)
- node (CHARACTER)

Miscelaneous calls

pdf_set_xy_offset (PROCEDURE)

This procedure allows to shift the position of all subsequent calls, be it text or graphic position calls. The use of an external pdf page will also be shifted by the specified value. This can be useful e.g. to cancel the effect of a cropbox that otherwise would shift the template within the document page - when "UseExternalPageSize" is set to "TRUE".

Sample call

```
RUN pdf_set_xy_offset ("Spdf", 10.0, 20.0).
```

Parameters

- stream name (CHARACTER)
- X shift value (DECIMAL)
- Y shift value (DECIMAL)

pdf_LastProcedure (FUNCTION)

This procedure allows to define a procedure that will be run by pdf_close, at the very end of the generation of the pdf file, just before closing the file.

Return type

LOGICAL (always TRUE)

Sample call

```
pdf_LastProcedure ("Spdf", hMyProc, "MyProc").
```

Parameters

- stream name (CHARACTER)
- procedure handle (HANDLE)
- internal procedure name (CHARACTER)

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 document you need call the 'pdf_add_tool' procedure.

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

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.

General

pdf_tool_add (PROCEDURE)

This procedure allows to start to use a tool.

Sample call
```
RUN pdf_tool_add ("Spdf","CustList", "TABLE",
                 TEMP-TABLE TT_mydata:HANDLE).
```
or
```
RUN pdf_tool_add ("Spdf","cal2015", "CALENDAR", ?).
```

Parameters
- stream name (CHARACTER)
- Tool name (CHARACTER) - identifier of the tool, created by this very call
- Tool type (CHARACTER) - one of "TABLE", "MATRIX" or "CALENDAR"
- Tool data (HANDLE) - only used for "TABLE", a temp-table containing the data to be displayed. For other tool types, it must be the unknown value '?'.

pdf_set_tool_parameter (PROCEDURE)

This procedure allows to define parameters for a tool previously added with 'pdf_tool_add'.

The parameters used for each tool are listed below.

Sample call
```
RUN pdf_set_tool_parameter("Spdf", "CustList", "Outline", 0, ".5").
```

Parameters

- stream name (CHARACTER)
- Tool name (CHARACTER) - identifier of the tool, as defined in 'pdf_tool_add'
- Parameter name (CHARACTER) - identifier of the parameter
- Column (INTEGER) - column for which to apply the parameter. 0 if not used (Calendar, Table or Matrix general parameters)
- Parameter value (CHARACTER) - value of the parameter

pdf_get_tool_parameter (FUNCTION)

Get a tool's parameter value.

Return type

CHARACTER

Sample call

```
iCol1 = INTEGER(pdf_get_tool_parameter("Spdf","CustList",
                                        "ColumnWidth", 1)).
```

Parameters

- stream name (CHARACTER)
- Tool name (CHARACTER) - identifier of the tool, as defined in 'pdf_tool_add'
- Parameter name (CHARACTER) - identifier of the parameter
- Column (INTEGER) - column for which to apply the parameter. 0 if not used (Calendar, Table or Matrix general parameters)

pdf_get_tool_parameter2 (FUNCTION)

Same as pdf_get_tool_parameter, but with a default value in the call which will be returned if the parameter has not been defined previously.

Return type

CHARACTER

Sample call

```
cCol3 = pdf_get_tool_parameter2("Spdf","CustList",
                                "ColumnHeader", 3, "Col3").
```

Parameters

- stream name (CHARACTER)
- stream name (CHARACTER)
- Tool name (CHARACTER) - identifier of the tool, as defined in 'pdf_tool_add'
- Parameter name (CHARACTER) - identifier of the parameter
- Column (INTEGER) - column for which to apply the parameter. 0 if not used (Calendar, Table or Matrix general parameters)
- Default value (CHARACTER) - value to return when the parameter has never been defined

Table tool

The Table tool allows you to easily generate a multi-page columnar report. For example, it can help to produce a Customer Listing (as illustrated below):

Example of report generated using the TABLE tool

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. e.g.: 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.
WrapText *(starting with version 4.0)*	For each column you can specify if its content can wrap (more than one line of text can be generated in the cell). The value to be defined is the maximum number of lines in the cell. Set to "0" for unlimited. If there would be more lines than the maximum number defined here, then the text would be truncated.
DetailBGColor	This represents the background colour for each of the detail lines in the columnar output (e.g.: the gray background in the table sample). Since this is a table specific parameter the Indicator must be zero (0).

Parameter Name	Description
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 (e.g.: 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).

Example of implementation

```
DEFINE VARIABLE h_PDFtable  AS HANDLE NO-UNDO.
DEFINE VARIABLE h_TT        AS HANDLE NO-UNDO.

DEFINE TEMP-TABLE TT_mydata NO-UNDO
    FIELD Custno    LIKE Customer.CustNum
    FIELD CustNameW LIKE Customer.NAME
    FIELD CustName  LIKE Customer.NAME
    FIELD PhoneNo   LIKE Customer.Phone
    FIELD State     LIKE Customer.State.

/* Build my Temp Table Data */
FOR EACH Customer NO-LOCK:
  CREATE TT_mydata.
  ASSIGN
    TT_mydata.CustNo    = Customer.CustNum
    TT_mydata.CustNameW = "<i>" + Customer.NAME + "</i>"
    TT_mydata.CustName  = Customer.NAME
    TT_mydata.PhoneNo   = "<color=green>" + Customer.Phone + "</color>"
    TT_mydata.State     = Customer.State.
END.

{ pdf_inc.i "THIS-PROCEDURE" }

RUN pdf_new("Spdf",cpdfIncludePath + "samples/super/table.pdf").

/* It is possible to make use of the tags - only in the "Wrap" columns,
   see the "WrapText" parameter below */
DEFINE VARIABLE cFont AS CHARACTER INITIAL "Helvetica" NO-UNDO.
RUN pdf_set_parameter("sPDF","UseTags","TRUE").
RUN pdf_set_parameter("sPDF","TagColor:Black","0,0,0").
RUN pdf_set_parameter("sPDF","TagColor:Red","255,0,0").
RUN pdf_set_parameter("sPDF","TagColor:Green","0,200,0").
RUN pdf_set_parameter("sPDF","TagColor:Blue","0,0,200").
RUN pdf_set_parameter("sPDF","BoldFont",cFont + "-Bold").
```

```
RUN pdf_set_parameter("sPDF","ItalicFont",cFont + "-Oblique").
RUN pdf_set_parameter("sPDF","BoldItalicFont",cFont + "-BoldOblique").
RUN pdf_set_parameter("sPDF","DefaultFont",cFont).
/* should match the DetailTextColor parameter defined below */
RUN pdf_set_parameter("sPDF","DefaultColor","Blue").

/* Set Page Header procedure */
pdf_PageHeader ("Spdf",
                THIS-PROCEDURE:HANDLE,
                "PageHeader").

/* Set Page Header procedure */
pdf_PageFooter ("Spdf",
                THIS-PROCEDURE:HANDLE,
                "PageFooter").

/* Don't want the table to be going until the
   end of the page or too near the side */
RUN pdf_set_LeftMargin("Spdf",20).
RUN pdf_set_BottomMargin("Spdf",50).

/* Link the Temp-Table to the PDF */
h_TT = TEMP-TABLE TT_mydata:HANDLE.
RUN pdf_tool_add ("Spdf","CustList", "TABLE", h_TT).

/* Now Setup some Parameters for the Table */

/* comment out this section to see what the default Table looks like */
RUN pdf_set_tool_parameter("Spdf","CustList","Outline",        0, ".5"             ).
RUN pdf_set_tool_parameter("Spdf","CustList","HeaderFont",     0, "Helvetica-Bold").
RUN pdf_set_tool_parameter("Spdf","CustList","HeaderFontSize", 0, "8"             ).
RUN pdf_set_tool_parameter("Spdf","CustList","HeaderBGColor",  0, "255,0,0"       ).
RUN pdf_set_tool_parameter("Spdf","CustList","HeaderTextColor",0, "255,255,255"   ).

RUN pdf_set_tool_parameter("Spdf","CustList","DetailBGColor",  0, "200,200,200").
RUN pdf_set_tool_parameter("Spdf","CustList","DetailTextColor",0, "0,0,200"    ).
RUN pdf_set_tool_parameter("Spdf","CustList","DetailFont",     0, "Helvetica"  ).
RUN pdf_set_tool_parameter("Spdf","CustList","DetailFontSize", 0, "8"          ).

RUN pdf_set_tool_parameter("Spdf","CustList","ColumnVerticalPadding", 0, "2"    ).
/* end of section */

/* Define Table Column Headers */
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnHeader", 1, "Customer #").
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnHeader", 2, "Name W"    ).
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnHeader", 3, "Name"      ).
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnHeader", 4, "Phone #"   ).
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnHeader", 5, "State"     ).

/* Define Table Column Widths */
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnWidth",  1, "10").
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnWidth",  2, "10").
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnWidth",  3, "10").
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnWidth",  4, "15").
RUN pdf_set_tool_parameter("Spdf","CustList","ColumnWidth",  5, "10").

/* Define Which Table Column Wraps */
/* this parameter is the maximum number of lines to wrap the field's value.
   If the field's value does not fit in this number of lines, then it gets truncated.
   Set to zero for no limit. If not set then the field won't wrap. */
RUN pdf_set_tool_parameter("Spdf","CustList","WrapText",     2, "0").
/* Define the phone # as wrap in order to be able to use the tags */
RUN pdf_set_tool_parameter("Spdf","CustList","WrapText",     4, "1").

/* Now produce the table */
RUN pdf_tool_create ("Spdf","CustList").

RUN pdf_close("Spdf").
IF RETURN-VALUE > '' THEN
    MESSAGE RETURN-VALUE
        VIEW-AS ALERT-BOX ERROR BUTTONS OK.
```

<u>Note</u>: page header and footer procedures intentionally omitted.

Matrix 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.

Note: starting with version 5.0 a matrix can span accross multiple pages. See the parameters below.

The following illustrates multiple matrices on a single page.

Example of report generated using the MATRIX tool

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. e.g.: The following matrix has 3 rows and 5 columns. The cell number is shown in the cell. 01 - 02 - 03 - 04 - 05 06 - 07 - 08 - 09 - 10 11 - 12 - 13 - 14 - 15

Parameter Name	Description
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
ColumnPadding *(starting with version 4.1)*	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). Default value: **5**.
ColumnVerticalPadding *(starting with version 4.1)*	This represents the space between the horizontal lines and the text. It is set for the table (not per cell or column) therefore the indicator must be zero (0). Default value: **1**.
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 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 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).
PageBreak *(starting with version 5.0)*	Flag (YES/**NO**) enabling a matrix to span on more than one page. If the matrix does not fit onto the page, instead of issuing an error, it will be continued onto the next page, just after the header - that is if one is defined.
Repeat1stRow *(starting with version 5.0)*	Flag (**YES**/NO), only usable is PageBreak = "YES", used to repeat the first line of the matrix in case of a page break. This allows to repeat the header of the table on each pages.
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).
WrapText *(starting with version 4.0)*	For each column you can specify if its content can wrap (more than one line of text can be generated in the cell). The value to be defined is the maximum number of lines in the cell. Set to "0" for unlimited. If there would be more lines than the maximum number defined here, then the text would be truncated.
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.

Note: the tables in this readme file are created using the matrix tool.

Calendar tool

The 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.
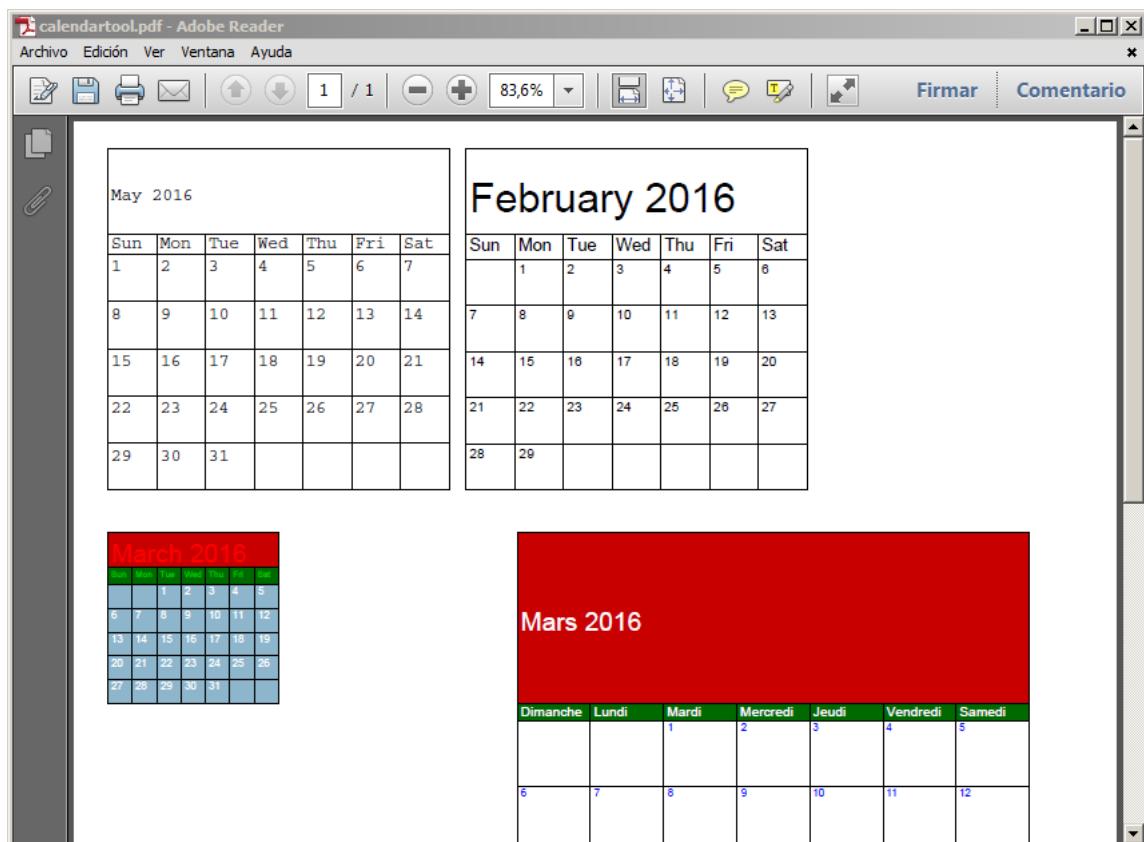
Example of result generated using the CALENDAR tool

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.

Parameter Name	Description
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 (e.g.: 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 (e.g.: 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 (e.g.: 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 (e.g.: 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 (e.g.: 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).

Parameter Name	Description
WeekDays	This parameter allows you to set the Day Labels (e.g.: 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.

API Index

Global list

- GetXMLNodeValue (FUNCTION)
- parseText (FUNCTION)
- pdf_Angle (FUNCTION)
- pdf_Bookmark (PROCEDURE)
- pdf_BottomMargin (FUNCTION)
- pdf_circle (PROCEDURE)
- pdf_clear_pdf_cache (PROCEDURE)
- pdf_close (PROCEDURE)
- pdf_close_path (PROCEDURE)
- pdf_close_path2 (PROCEDURE)
- pdf_curve (PROCEDURE)
- pdf_decr_parameter (PROCEDURE)
- pdf_ellipse (PROCEDURE)
- pdf_Encrypt (PROCEDURE)
- pdf_exec_footer (PROCEDURE)
- pdf_ext_get_nb_pages (PROCEDURE)
- pdf_ext_get_page (PROCEDURE)
- pdf_ext_get_path (PROCEDURE)
- pdf_FillBlue (FUNCTION)
- pdf_FillGreen (FUNCTION)
- pdf_FillRed (FUNCTION)
- pdf_fill_text (PROCEDURE)
- pdf_Font (FUNCTION)
- pdf_FontType (FUNCTION)
- pdf_font_diff (PROCEDURE)
- pdf_Font_Loaded (FUNCTION)
- pdf_GetBestFont (PROCEDURE)
- pdf_GetNumFittingChars (FUNCTION)
- pdf_get_image_info (PROCEDURE)
- pdf_get_info (FUNCTION)
- pdf_get_parameter (FUNCTION)
- pdf_get_parameter2 (FUNCTION)
- pdf_get_pdf_info (FUNCTION)
- pdf_get_tool_parameter (FUNCTION)
- pdf_get_tool_parameter2 (FUNCTION)
- pdf_get_widgets (PROCEDURE)
- pdf_get_wrap_length (FUNCTION)
- pdf_GraphicX (FUNCTION)
- pdf_GraphicY (FUNCTION)
- pdf_ImageDim (FUNCTION)
- pdf_incr_parameter (PROCEDURE)
- pdf_insert_page (PROCEDURE)
- pdf_in_transaction (FUNCTION)
- pdf_LastProcedure (FUNCTION)
- pdf_LeftMargin (FUNCTION)
- pdf_line (PROCEDURE)
- pdf_line_dec (PROCEDURE)
- pdf_link (PROCEDURE)
- pdf_load_font (PROCEDURE)
- pdf_load_font2 (PROCEDURE)
- pdf_load_image (PROCEDURE)
- pdf_load_template (PROCEDURE)
- pdf_load_xml (PROCEDURE)
- pdf_Markup (PROCEDURE)
- pdf_merge_stream (PROCEDURE)

Procedures list

- pdf_decr_parameter
- pdf_ellipse
- pdf_Encrypt
- pdf_exec_footer
- pdf_ext_get_nb_pages
- pdf_ext_get_page
- pdf_ext_get_path
- pdf_fill_text
- pdf_font_diff
- pdf_GetBestFont
- pdf_get_image_info
- pdf_get_widgets
- pdf_incr_parameter
- pdf_insert_page
- pdf_line
- pdf_line_dec
- pdf_link
- pdf_load_font
- pdf_load_font2
- pdf_load_image
- pdf_load_template
- pdf_load_xml
- pdf_Markup
- pdf_merge_stream
- pdf_move_to
- pdf_new
- pdf_new_page
- pdf_new_page2
- pdf_note
- pdf_open_PDF
- pdf_path_add_segment
- pdf_pattern_begin
- pdf_pattern_end
- pdf_pattern_use
- pdf_place_image
- pdf_place_pdf_page
- pdf_rect
- pdf_rect2
- pdf_rectdec
- pdf_ReplaceText
- pdf_reset_all
- pdf_reset_stream
- pdf_rgb
- pdf_set_base14_codepage
- pdf_set_BottomMargin
- pdf_set_dash
- pdf_set_dash_pattern
- pdf_set_FillBlue
- pdf_set_FillGreen
- pdf_set_FillRed
- pdf_set_font
- pdf_set_GraphicX
- pdf_set_GraphicY
- pdf_set_info
- pdf_set_LeftMargin
- pdf_set_linecap
- pdf_set_linejoin
- pdf_set_MinPdfVersion
- pdf_set_Orientation
- pdf_set_Page
- pdf_set_PageHeight
- pdf_set_PageRotate

- pdf_set_PageWidth
- pdf_set_PaperType
- pdf_set_parameter
- pdf_set_RightMargin
- pdf_set_TextBlue
- pdf_set_TextGreen
- pdf_set_TextRed
- pdf_set_TextX
- pdf_set_TextXY
- pdf_set_TextY
- pdf_set_tool_parameter
- pdf_set_TopMargin
- pdf_set_VerticalSpace
- pdf_set_xy_offset
- pdf_skip
- pdf_skipn
- pdf_stamp
- pdf_stroke_color
- pdf_stroke_fill
- pdf_subset_add_range
- pdf_subset_add_string
- pdf_text
- pdf_text_align
- pdf_text_at
- pdf_text_boxed_xy
- pdf_text_center
- pdf_text_char
- pdf_text_charxy
- pdf_text_color
- pdf_text_render
- pdf_text_rotate
- pdf_text_scale
- pdf_text_skew
- pdf_text_to
- pdf_text_xy
- pdf_text_xy_dec
- pdf_tool_add
- pdf_transaction_begin
- pdf_transaction_buffer
- pdf_transaction_commit
- pdf_transaction_rollback
- pdf_use_PDF_page
- pdf_use_template
- pdf_watermark
- pdf_wrap_text
- pdf_wrap_text_x
- pdf_wrap_text_xy
- pdf_wrap_text_xy_dec

Functions list

- GetXMLNodeValue
- parseText
- pdf_Angle
- pdf_BottomMargin
- pdf_FillBlue
- pdf_FillGreen
- pdf_FillRed
- pdf_Font
- pdf_FontType
- pdf_Font_Loaded
- pdf_GetNumFittingChars

- pdf_get_info
- pdf_get_parameter
- pdf_get_parameter2
- pdf_get_pdf_info
- pdf_get_tool_parameter
- pdf_get_tool_parameter2
- pdf_get_wrap_length
- pdf_GraphicX
- pdf_GraphicY
- pdf_ImageDim
- pdf_in_transaction
- pdf_LastProcedure
- pdf_LeftMargin
- pdf_Orientation
- pdf_Page
- pdf_PageFooter
- pdf_PageHeader
- pdf_PageHeight
- pdf_PageNo
- pdf_PageRotate
- pdf_PageWidth
- pdf_PaperType
- pdf_pattern_search_by_key
- pdf_PointSize
- pdf_Render
- pdf_RightMargin
- pdf_ScaleX
- pdf_ScaleY
- pdf_TextBlue
- pdf_TextGreen
- pdf_TextRed
- pdf_TextX
- pdf_TextY
- pdf_text_fontwidth
- pdf_text_fontwidth2
- pdf_text_width
- pdf_text_widthdec
- pdf_text_widthdec2
- pdf_TopMargin
- pdf_TotalPages
- pdf_VerticalSpace

How to

How to Implement Compression

pdfInclude 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:

- Download and install the appropriate binary from www.zlib.net
  Note: zlib1.dll is provided in the pdfInclude download for Windows 32 bits.
- Open the file pdf_pre.i Progress include file in the pdfInclude download. Change the zlib pre-processor to point to the appropriate DLL or Shared Library object.
  E.g.: For Windows installs the zlib pre-processor would have the value of zlib1.dll defined like:
  &GLOBAL-DEFINE zlib zlib1.dll
- 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.
- Run your code and you will see a reduction in your PDF file size.
  pdfInclude does compress images but since they are already compressed, you may find that they aren't further reduced by very much.

How to Implement Encryption

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

pdfInclude uses external binaries (a DLL or Shared Library object) to perform the RC4 encryption. AES encryption is done without any external library. Being more secure, it is thus the recommended choice.

For Windows 32 bits, the DLL (procryptlib.dll) used for RC4 encryption is included in the download.

If you are on a *NIX OS or are using OpenEdge 64 bits on Windows, then you will need to compile the rc4.c object into a Shared Library, resp. a DLL.

Steps to implementing encryption:

- Only for pdfInclude version < 4.1: 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)

  Note: starting with pdfInclude 4.1, we use the built-in MD5-DIGEST ABL function.

- Only for RC4 encryption: 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') - not needed for AES encryption (starting with version 4.1)

- 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.

- By default RC4-40 encryption is used. If you wish to change the algorithm or the key length (starting with version 4.0), use the following parameters:
  - EncryptAlgoritm: "RC4" or "AES"
  - EncryptKey: for RC4 you can choose "40" or "128". For AES, only "128" is currently supported.

- 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.

  See the Native encryption paragraph for more details about the encryption parameters.

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

Note: currently pdfInclude support the following encryption algorithms:

- RC4 40 bits - all versions
- RC4 128 bits - starting with 4.1 (it was implemented in 3.3 but due to some bugs the resulting pdf file could not be read.)
- AES 128 bits - starting with 4.1
- AES 256 bits is not currently supported. If you are interested in sponsoring this development, please contact us!

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.

The Page Footer is generated after anything else, just before a new page is generated, or when the pdf file is closed.

Note: 'pdf_new_page' can be called explicitly, or automatically when the text reaches the end of page.

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:
- activate the default header and/or footer (starting with pdfInclude 4.2),
- or create custom code blocks (procedures) in your program that will generate the appropriate output.

In case the footer does not appear due to some navigating in the file using e.g. 'pdf_set_page', you can force its display using 'pdf_exec_footer'.

This very document uses both the default header - which in turn makes use of a reusable pattern - and a custom footer.

Default headers & footers

```
RUN pdf_set_parameter("Spdf", "defaultHeader", "TRUE").
RUN pdf_set_parameter("Spdf", "defaultFooter", "TRUE").
```

Setting these parameters is enough for the header and/or footer to appear.
- The default header is composed of an optional logo and 2 lines which can be customized, setting some parameters:
  - First line: its default is to show the document title, as set by 'pdf_set_info'.

    It can be changed to any string, setting the "headerLine1" parameter.
  - Second line: its default is to show the document title, as set by 'pdf_set_info'. It is displayed only if the top margin leaves enough space for it to appear.

    It can be changed to any string, setting the "headerLine2" parameter.
  - the optional logo is displayed when the parameter "headerLogo" is set. This parameter is a list, separated by pipes. Its format is: "picture_file_path|picture_width|picture_height|alignment", where alignment is one of "LEFT" or "RIGHT". Example (as used in this very document): "readme/pdfInclude-logo.png|90|30|RIGHT".
  - the default header can be deactivated of the first page of the document, setting the parameter "headerNotOn1stPage" to "TRUE".
- The default footer is composed of one line. Its default value is "page n/m".

    It can be changed to any string, setting the "footerLine1" parameter. For example:

```
RUN pdf_set_parameter("Spdf",
    "footerLine1",
    "page " + pdf_PageNo("Spdf") + " de " + pdf_TotalPages("Spdf")).
```

Custom headers & footers

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_skipn ("Spdf", 2).
  RUN pdf_text_to  ("Spdf",  "Page: "
                            + STRING(pdf_page("Spdf"))
                            + " of " + pdf_TotalPages("Spdf"), 97).
END. /* PageFooter */
```

Nevertheless, this code won't be run unless we communicate this procedure to pdfInclude. 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>, THIS-PROCEDURE:HANDLE, <Procedure Name>).
pdf_PageHeader (<stream>, 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.

Two very useful calls for custom headers and footers are the TotalPages and PageNo functions.

pdfInclude - README.pdf
Jicé

pdfInclude
for Progress OpenEdge
PDF
www.snowwolf-software.com

How to use PDF Forms

Using an external pdf file with form controls, you can generate and populate arbitrary complex forms.

The first step is to create a PDF form or template. This can be achieved using e.g. Adobe Acrobat (not the reader) or free software like LibreOffice, which has a very good "export to pdf" funcionality, including pdf forms support.

The second step is to use pdfInclude APIs to open it, use it, and populate the form widgets, which are place holders for our data. You can also add onto the pdf form any text, graphic or picture, outside of the form controls, using the standard pdfInclude APIs.

This way of creating documents allows to separate the design of the document with its contents, as long as the developer and the user who is creating the template agree on the form widget names. The user can decide both the look-and-feel of the document and the placement of the data.

Of course, it is also possible to use external pdf files without form controls, and statically place the content. This allows the user to control the look-and-feel of the document, but not to change the data placement.

Finally, it is possible to create a complex and good-looking document all from within pdfInclude, without any external pdf template, if for example you don't want the user to be able to modify the look-and-feel of the document.

The data you use to populate your form might come from any source:
* of course the database,
* text files,
* xml files,
* existing text reports (using pdfInclude, you can for example build a post-processor which will transform them into a full fledged pdf document),
* any other data source.

Below I will illustrate how easy it is to create an purchase order template and publish data to it, in order to generate nice pdf documents.

Create a PDF form

To create our template, we will use LibreOffice, which is a powerful office suite with a very interesting feature for us: it can export its documents to pdf, and even create pdf forms. Furthermore, it is Free and Open Source Software.

From the office suite we will use the word processor (Writer) in this example. However, you can use any tool from the suite, for example Draw, with which you can import an existing pdf file with a quite good fidelity, and modify it to make it a usable pdf template and/or form.

Using Writer, you design your purchase order template, then you add some form widgets. In order to add form widgets (or controls):
* open the "form control" tool bar, from the View/Toolbars menu.
* activate the design mode, clicking on the "Design Mode On/Off" button.

Then you can add controls:
- fill-ins
- check boxes
- radio buttons
- combo boxes
- list boxes

For each control you must set some properties that will be used later by pdfInclude. To open the property dialog box, double-click on the control, with the form design mode activated.
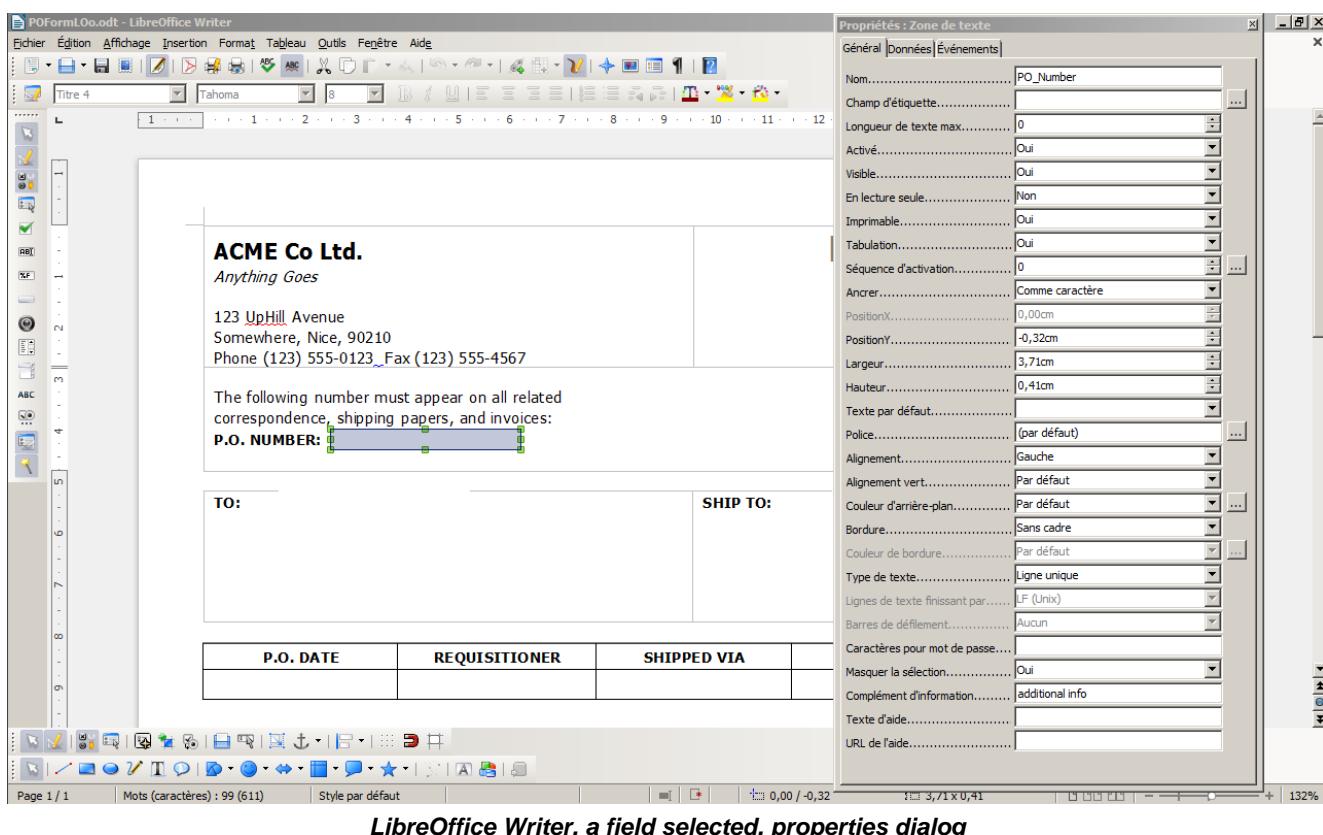
LibreOffice Writer, a field selected, properties dialog

The following properties are used by pdfInclude:

Property Name	Apply to	Description
Name	All	Name of the widget. This will be used later to tell to pdfInclude which widget we are targeting. This is by far the most important property to set for each widget.
Default text	Fill-in, Combo box, List box	Default text for the widget. Can be kept in the pdfInclude generated file if the parameter formFlattenWithDefaultValues is set to TRUE.
Font	All	Defines the font, point size and font colour used to fill the widget.
Alignment	Fill-in	Alignment of the text: left, centre or right
Vertical Alignment	Fill-in	Vertical alignment of the text: top, middle or bottom
Text type	Fill-in	Tells pdfInclude if the fill-in is Single-line or Multi-line.
List entries	Combo box, List box	List of entries in the combo or list box

Other properties are useful, and you will want to use them:

Property Name	Apply to	Description
Border	All	Border to draw around the control. For a pdf template, you will usually want "Without frame" or "Flat".
Background color	All	Background colour ;)

Once the controls are added to the form, their properties properly set, you have to generate the PDF form. For this, go to "File/Export as PDF" then click on the 'Export' button. If you have no widget with duplicated names (see below), you can just use the "Export directly as pdf" button present in the standard tool bar.

Multiple widgets using the same name

In the export dialog, you can notice the "Allow duplicate field names" checkbox.

You can indeed define more than one widget (e.g. fill-in) using the same name. If you do so, pdfInclude will fill all the widgets using the same name with the same value (using only one call to 'pdf_fill_text'). This can be useful for example if the same date is repeated more than once on the same document. When the tasks are divided between a developper and a user creating the forms, this allows him to decide to place the same data more than once on the document, without the need for code adaptations.

When using the same name for multiple widgets, don't forget the check the checkbox above. Of course, also make sure that "Create PDF form" is checked.

Filling the PDF form

In order to demonstrate how to fill the pdf form, we will let you read the following code. You will see that nowhere we specify the position, font, color, alignment, s.o.* but only the values to be set into the different controls we have created.

* Except for some fields, in order to demonstrate the ability to force these values programatically

For this example we are connected to the sports2000 database, and we have saved our PDF form under "samples/support/POForm.pdf".

```
{pdf_inc.i "THIS-PROCEDURE"}
DEFINE VARIABLE i_LineCounter AS INTEGER NO-UNDO.
DEFINE VARIABLE dec_SubTotal  AS DECIMAL NO-UNDO.

RUN pdf_new ("Spdf", "PO.pdf").
/* Set the PageHeader Routine */
pdf_PageHeader ("Spdf",
                THIS-PROCEDURE:HANDLE,
                "PageHeader").
/* Set the PageFooter Routine */
pdf_PageFooter ("Spdf",
                THIS-PROCEDURE:HANDLE,
                "PageFooter").

RUN pdf_open_PDF("Spdf", "samples/support/POForm.pdf", "PO").

RUN ProcessPOs.

RUN pdf_close("Spdf").
IF RETURN-VALUE > '' THEN
    MESSAGE RETURN-VALUE
        VIEW-AS ALERT-BOX ERROR BUTTONS OK.
```

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

PROCEDURE ProcessPOs:
  FOR EACH PurchaseOrder NO-LOCK
    WHERE PurchaseOrder.PoNum >= 8001 AND PurchaseOrder.PoNum <= 8002:

    RUN DoNewPage.

    ASSIGN i_LineCounter = 0
           dec_SubTotal  = 0.

    FOR EACH POLine NO-LOCK OF PurchaseOrder:
      i_LineCounter = i_LineCounter + 1.
      RUN DoPOLine.
      /* If more than 5 lines then create another page */
      IF i_LineCounter > 5 THEN DO:
        RUN DoNewPage.
        i_LineCounter = 0.
      END.
    END.
  END.

END.

PROCEDURE DoNewPage:
  RUN pdf_new_page("Spdf").
  RUN pdf_use_PDF_page("Spdf","PO",1).
END.

PROCEDURE PageHeader:
  /* Get Supplier Info */
  FIND Supplier NO-LOCK WHERE Supplier.SupplierID = PurchaseOrder.SupplierID.
  /* Display 'To' information */
  RUN pdf_fill_text("Spdf","To_Name",Supplier.NAME,
                    "font=Times-Bold,fontSize=12"). /* forced font */
  RUN pdf_fill_text("Spdf","To_Address1",Supplier.Address,"").
  RUN pdf_fill_text("Spdf","To_Address2",Supplier.Address2,"").
  /* Display 'Ship To' Information */
  RUN pdf_fill_text("Spdf","PO_Number",STRING(PurchaseOrder.PONum,"99999"),"").
  RUN pdf_fill_text("Spdf","ShipTo_Name",Supplier.NAME,"").
  RUN pdf_fill_text("Spdf","ShipTo_Address1",Supplier.Address,"").
  RUN pdf_fill_text("Spdf","ShipTo_Address2",Supplier.Address2,"").
  /* Display 'PO Header' Information */
  RUN pdf_fill_text("Spdf",
                    "PO_Date",
                    STRING(PurchaseOrder.DateEntered,"99/99/99"),
                    "").
  RUN pdf_fill_text("Spdf",
                    "Terms",
                    "NET 30",
                    "").
END.

PROCEDURE PageFooter:
  RUN pdf_fill_text("Spdf",
                    "SubTotal",
                    STRING(dec_SubTotal,">,>>9.99"),
                    "").
  RUN pdf_fill_text("Spdf",
                    "PO_Total",
                    STRING(dec_SubTotal,">,>>9.99"),
                    "").
  RUN pdf_fill_text("Spdf",
                    "Mentions",
                    "Lorem ipsum dolor sit amet...",
                    "Font=Helvetica,fontSize=7"). /* forced font */
END.

PROCEDURE DoPOLine:
  RUN pdf_fill_text("Spdf",
                    "Qty_" + STRING(i_LineCounter),
                    STRING(POLine.Qty,">,>>9.99"),
                    "align=center"). /* forced alignment */
  RUN pdf_fill_text("Spdf",
                    "Unit_" + STRING(i_LineCounter),
```

```
                            POLine.ItemNum,
                            "").
      FIND ITEM NO-LOCK WHERE ITEM.ItemNum = POLine.ItemNum.
      RUN pdf_fill_text("Spdf",
                        "Description_" + STRING(i_LineCounter),
                        ITEM.ItemName,
                        "").
      RUN pdf_fill_text("Spdf",
                        "UnitPrice_" + STRING(i_LineCounter),
                        STRING(ITEM.Price,">,>>9.99"),
                        "").
      RUN pdf_fill_text("Spdf",
                        "Total_" + STRING(i_LineCounter),
                        STRING(ITEM.Price * POLine.Qty,">,>>9.99"),
                        "").
      ASSIGN dec_SubTotal = dec_SubTotal
                          + ITEM.Price * POLine.Qty.
    END.
```

The result will be (only the first page is shown):

Note: it is also possible to pre-fill a PDF Form, while still keeping it a PDF Form. In order to achieve this, use the "formFlatten" parameter:

```
RUN pdf_set_parameter("Spdf", "formFlatten", "").
```

See Using an external pdf file for a list of available parameters.

How to create multi-lingual documents

The PDF format, by default, only support Latin-1 characters (Western Europe).
pdfInclude supports various ways of printing other code pages.

Single byte code pages

Base 14 fonts

Starting with version 4.0, pdfInclude allows to replace the Latin-1 characters with other code pages. Currently are supported:
- iso8859-2 (Latin-2, Central Europe)
- 1250 (Microsoft Windows Central Europe, like iso8859-2)
- 1251 (Microsoft Windows Cyrillic)
- 1254 (Microsoft Windows Turkish)

Using pdf_set_base14_codepage, you replace - **for the whole pdf document** - the code page used by the 14 base fonts by the specified code page. It is sufficient to call this procedure; any text printed to the document will be translated to this code page.
```
RUN pdf_set_base14_codepage ('Spdf', '1251').
```

Note: for UTF-8 sessions, it is necessary to set the parameter *CodePage*, in order to tell pdfInclude to which codepage to convert the strings to when using a single byte encoding. Its default value is "iso8859-1".

Using a specific TTF font

It is also possible to find or generate fonts defined specially for a specific code page.
For example http://www.aatseel.org/resources/fonts/windows_central.htm lists some fonts defined specifically for a given code page.
Then you load the font and use it normally.

Multi-byte code pages

Starting with pdfInclude 5.0, full UTF-8 support has been implemented. This allows to load Unicode fonts and output any UTF-8 string to the pdf document, with all the characters supported by the font. As Unicode fonts with a large characters coverage can be huge, we also implemented font sub-setting: this allows to embed only the characters which have been used in the document.

Enabling UTF-8

UTF-8 is enabled for one font, from the moment you load it: 'pdf_load_font2' supports a parameter, telling it that we will use this font to output UTF-8 strings. 'pdf_load_font' also switches to UTF-8 when you use a .ufm file instead of a .afm file when loading the font. It is recommended to use the SUBSET functionality in order to keep your files smaller.
Once loaded with this flag activated, every API call which outputs text can be passed UTF-8 strings (even within single bytes sessions).

Example

For example, the following document uses Chinese characters, weights 138 Kb, whereas the font it uses (Arial Unicode) weights 21 Mb!
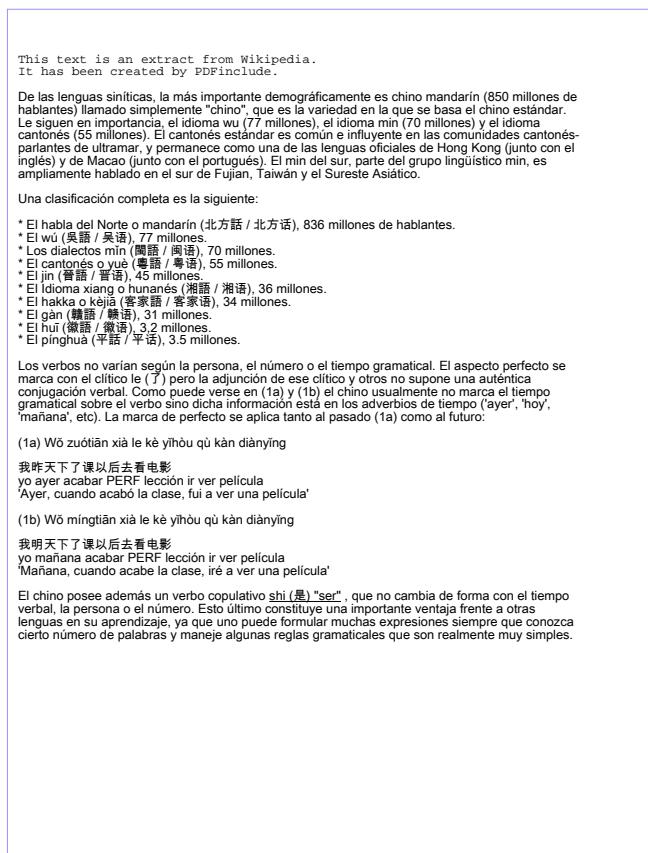
It has been generated with the following code:

```
{pdf_inc.i "THIS-PROCEDURE"}
RUN pdf_new ("Spdf", "chino.pdf").
/* The following call loads a unicode font: cf. YES parameter, using SUBSET */
RUN pdf_load_font2 ("Spdf", "ArialUni", "fonts\arialuni.ttf", YES, "", "SUBSET").
RUN pdf_new_page ("Spdf").
/* We use the built-in Courier font, not UTF-8: */
RUN pdf_set_font ("Spdf", "Courier", 12).
RUN pdf_text ("Spdf", "This text is an extract from Wikipedia.").
RUN pdf_skip ("Spdf").
RUN pdf_text ("Spdf", "It has been created by PDFinclude.").
RUN pdf_skipn ("Spdf", 2).
/* We switch to Arial Unicode, loaded for UTF-8 text: */
RUN pdf_set_font ("Spdf", "ArialUni", 12).
DEFINE VARIABLE cLine AS CHARACTER   NO-UNDO.
DEFINE VARIABLE iMaxY AS INTEGER     NO-UNDO.
/* "samples/support/chinese.txt" contains UTF-8 text */
INPUT FROM VALUE("samples/support/chino2.txt") BINARY NO-MAP NO-CONVERT.
REPEAT:
    IMPORT UNFORMATTED cLine.
    IF cLine > "" THEN
        /* We output UTF-8 text, using Arial Unicode: */
        RUN pdf_wrap_text ("Spdf", cLine, 1, 100, "LEFT", OUTPUT iMaxY).
    RUN pdf_skip ("Spdf").
END.
INPUT CLOSE.
RUN pdf_close ("Spdf").
IF RETURN-VALUE > '' THEN
    MESSAGE RETURN-VALUE
        VIEW-AS ALERT-BOX ERROR BUTTONS OK.
```

More examples

Hello world

```
{pdf_inc.i}
RUN pdf_new ("Spdf", "C:/TEMP/hello.pdf").
RUN pdf_new_page ("Spdf").
RUN pdf_text ("Spdf", "Hello World").
RUN pdf_close ("Spdf").
```

Hello world - compressed and encrypted

```
{pdf_inc.i}
RUN pdf_new ("Spdf", "C:/TEMP/hello-enc.pdf").
RUN pdf_set_parameter ("Spdf", "Compress", "TRUE").
RUN pdf_set_parameter ("Spdf", "Encrypt", "TRUE").
RUN pdf_new_page ("Spdf").
RUN pdf_text ("Spdf", "Hello World").
RUN pdf_close ("Spdf").
```

Hello world - encrypted with AES-128 and passwords

```
{pdf_inc.i}
RUN pdf_new ("Spdf", "C:/TEMP/hello-enc.pdf").
RUN pdf_set_parameter ("Spdf", "Compress", "TRUE").
RUN pdf_set_parameter ("Spdf", "Encrypt", "TRUE").
RUN pdf_set_parameter ("Spdf", "EncryptAlgorithm", "AES").
RUN pdf_set_parameter ("Spdf", "EncryptKey",        "128").
RUN pdf_set_parameter ("Spdf", "MasterPassword", "M4st3r!").
RUN pdf_set_parameter ("Spdf", "UserPassword",    "$Us3r$").
RUN pdf_new_page ("Spdf").
RUN pdf_text ("Spdf", "Hello World").
RUN pdf_close ("Spdf").
```

How to print barcodes

In order for pdfInclude to generate barcodes, one has to use a barcode font.

The following code shows how to generate a report from a database (sports2000 again ;) and to print code39 barcodes. It also illustrates many different options available with pdfInclude, including font embedding, image embedding, and text placements, html-like tags, etc.

```
{pdf_inc.i "THIS-PROCEDURE"}
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", "itemlist.pdf").

/* activate tags to put a link on the item name */
RUN pdf_set_parameter("Spdf","UseTags","TRUE").
/* no need to define the fonts for the tags:
   we are only using <url=> which does not change the font */

/* Load Bar Code Font */
RUN pdf_load_font  ("Spdf", "Code39",
                    "samples/support/code39.ttf",
                    "samples/support/code39.afm","").
```

```
/* Load an image that we will use to show how to place onto the page */
RUN pdf_load_image ("Spdf", "Product", "samples/support/product.jpg").

/* Set Document Information */
RUN pdf_set_info("Spdf","Author","pdfInclude team").
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 5.0").
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:
  Vitems = Vitems + 1.
  RUN display_item_info.
  /* Process Record Counter */
  IF Vitems = 6 THEN
    RUN new_page.
END.

RUN pdf_close("Spdf").
IF RETURN-VALUE > '' THEN
    MESSAGE RETURN-VALUE
        VIEW-AS ALERT-BOX ERROR BUTTONS OK.

/* ------------------- 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 , 110,1.0).

  /* Draw Smaller box (beige filled) to contain Category Description */
  RUN pdf_rect ("Spdf", 350, Vrow + 10, 240, 45,1.0).
  /* 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  , 90,1.0).

  /* Place Link around the Image Box */
  RUN pdf_link ("Spdf",
                20,  pdf_GraphicY("Spdf") - 90 ,
                130, pdf_GraphicY("Spdf"),
                "http://www.example.com?ItemNum=" + STRING(ITEM.ItemNum),
                1, 0, 0, 1, "P").

  /* Display a JPEG picture in the First Box of each Frame */
  IF Vitems = 1 THEN DO:
    RUN pdf_place_image ("Spdf","Product",
                         pdf_LeftMargin("spdf") + 12,
                         pdf_PageHeight("Spdf") - Vrow - 13,
                         pdf_LeftMargin("Spdf") + 95, 86).
  END.

  /* 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", /* also create a hyperlink on the item name */
      "<url=http://www.example.com?ItemNum=" + STRING(ITEM.ItemNum) + ">"
      + ITEM.ItemName + "</url>", 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 - but not for the first product */
  IF Vitems <> 1 THEN DO:
    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).
  END.
  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 */
```

This will give the following result (only the first page is shown):

Now it's up to you to build the best PDF documents!

<u>Note</u>: in order to make the examples in sample/super work, modify samples/support/config.i to set the location where pdfInclude is installed on your machine, and the Windows root path.

Annex: pdfInclude changes

v5.1

Pdf_inc.p

- pdf_PageFooter: if called with "" as the procedure, then consider that the page has a footer not to add one later in pdf_close: it is the dev intention not to have a footer
- pdf_open_pdf: if the same pdf has already been opened for another stream, then no need to parse it again, we just copy it to the other stream -> huge optimization for this use case
- pdf_open_pdf: implemented a cache for external pdfs! New parameter "usePdfCache" (TRUE by default). Whenever a given pdf has been opened once, it will subsequently be loaded from the cache. -> huge optimization!
- pdf_clear_pdf_cache: new API to remove the cache files for a given pdf template.
- recursivelyAssign* & Export*: only export the same dictionnary or array once. Useful in case of merged documents: this makes them smaller by deduplicating the common objects.

Bug fixes

- pdf_reset_stream: at pdf close time, do not delete external pdf resources when they are still used in another non closed stream. This allows to use the same pdf template in various streams generated in parallel.
- pdf_merge_stream_external: TT_Info was not copied. Fixed.

Pdfextract.p

- Bug fix: LoadObjectPointers: XREF streams /Index array was not correctly taken into account when it specified more than one range of objects. Also better manage files with many versions of the same object.

v5.0

Pdf_inc.p

- Lots of refactoring, de-duplication of code, remove dead code, move some code to sub-folders inc and lib, s.o.
- UTF-8 support (see also Fonts below)

    Utf-8 support for text: all the API which take as parameter some text to display on the pdf file can use UTF-8 strings when the loaded font in Unicode (loaded using a .ufm file or loaded through pdf_load_font2)
- Fonts
    - Manage loading of Unicode fonts through an .ufm file instead of .afm (extracted using ttf2ufm.exe) with pdf_load_font()
    - new API: pdf_load_font2(): load a font without the need of an afm/ufm file. This will become the preferred way of loading fonts for use with PDFInclude.

        Allows to better manage damaged font files and to integrate with our font sub-setting functionality.

Allows to load a font for Unicode (UTF-8) text or standard text, and to specify if the font must be embedded or subset.

The font parsing is done fully in ABL (in lib/pdf_parse_font.p).

Implemented a cache in SESSION:TEMP-DIRECTORY/pdfcache)

- ○ Font sub-setting: add "|SUBSET" to the font name in pdf_load_font() or the option "SUBSET=YES" when calling pdf_load_font2(), and enjoy the pdf file size :)

  Make sub-setting work with Unicode fonts, wherever the text to displays come from. The embedded font will only contain the characters used in the pdf file.

  2 new APIs to add some more characters to the font subset: pdf_subset_add_string and pdf_subset_add_range (in case you wish the pdf file to contain more than the used characters)

  Make font sub-setting work with non-Unicode fonts

  <u>Note</u>: Sub-setting should not be used if the pdf file is to be modified later, because the font will be missing characters. It still can be modified if the font is installed on the computer where the modification is done.

- Pictures
  - ○ Support for more image types (all done in ABL): BMP (1, 4, 8 & 24 bits tested), GIF (only 8 bits, non-interlaced)
  - ○ Implemented a cache for images (cache files go into SESSION:TEMP-DIRECTORY/pdfcache) for better performance
- Other enhancements
  - ○ New tag when using html like tags: . Let you specify different fonts. Usage: <font=myFont>some text, where "myFont" has been loaded previously.
  - ○ Filling forms: the caller can now specify which font and size to use, thus not using the font defined in the template (option font=myFont,fontSize=nn of pdf_fill_text()). This also works with Unicode fonts (which of course must have been loaded previously) and font subsets.
  - ○ Add pdf_(set_)Stroke(Red|Green|Blue)
  - ○ default header: added parameter headerNotOn1stPage
  - ○ pdf_close now checks that every page has got one footer, and create the footer when it is missing
  - ○ pdf_set_dash now resets the dash when called with two zeroes
  - ○ pdf_insert_page now publishes "InsertPDFPage"
  - ○ new parameter insertPageMode: append,insert,next. By default append. When we go past the page footer, instead of always creating a new page (append), we can now insert a page, or just go to the next page.
  - ○ pdf_set_GraphicX & Y: allow negative values
  - ○ new parameter headerLogo "picture_file_path|picture_width|picture_height|alignment" for the default header
  - ○ new APIs:
    - ○ pdf_close_path2 (close a path without filling it)
    - ○ pdf_set_dash_pattern (allows to set the pattern of dashed lines)
    - ○ pdf_place_pdf_page: like pdf_use_pdf_page but with an extra options parameter with values for Scale,X,Y,Rotate,Background,Border,BorderWidth; allows to embed an external pdf file as if it were a picture
    - ○ pdf_pattern_search_by_key
    - ○ pdf[_set]_RightMargin
    - ○ pdf_wrap_text_x (same as pdf_wrap_text but with pdf points instead of chars)
  - ○ pdf_move_to now take DECIMAL parameters; this allow e.g. circles with decimal coordinates to close properly ;)

Bug fixes

- using <url> changed the current stroke color
- bookmarks/links were sometimes corrupted
- when using tags and mixing Unicode and non-Unicode fonts, the text could be corrupted. Fixed by moving pdf_replace_text() at the "right place"
- better handling of rotated pages
- make sure the pages are output in the right order!
- handle chr(0) in changePageText
- spaces in <url> links can lead to strange behaviour: the caller must replace them with %20
- transaction rollback: did not rollback temp-tables (tt_pdf_object, TT_pdf_annot, TT_pdf_bookmark & TT_pdf_FillTxt)

Tools

- Matrix: 2 new parameters:
  - PageBreak: Flag enabling a matrix to span on more than one page.
  - Repeat1stRow: Flag, only usable is PageBreak = "YES", used to repeat the first line of the matrix in case of a page break.

Pdfextract.p

- Implemented a test suite, in order to test the extraction and re-creation of a pdf file with the same content on a large corpus of source pdf files.

Bug fixes

- some fonts were missing on multiple document pages
- the xref stream was incorrectly extracted in some cases

Other

- ttf2ufm.p: TTF font parser to generate .ufm files from a TTF file; entirely done in ABL, does not need an external .exe file
- PDFInclude toolkit: graphical interface to extract pages from source pdf files, rotate them, concatenate them, add a watermark, an overlay, header and footer, pdf properties (Author, title…) – à la pdftk.
- New library to perform syntax coloring of ABL code with PDFInclude

v4.2

Pdf_inc.p

- Other enhancements
  - Patterns (pdf xobjects)

    You can define blocks of text, graphic, images, etc. and reuse them many times. The new API is: pdf_pattern_begin, pdf_pattern_end and pdf_pattern_use. Example implementation in pdf_default_header.
  - default page header and/or footer

    (new boolean parameters "defaultHeader" and "defaultFooter") with some optional

customization for the header (the "headerLine1" & "headerLine2" parameters - default values are Title and Author) and the footer (parameter "footerLine1" - default value "page n/m"). 2 parameters ("headerSeparator" & "footerSeparator", default value: "TRUE") allow to draw a line between header or footer and the rest of the page. <u>Note</u>: in order to optimize the pdf file size, the header makes use of the new functionality of "patterns"; if one of the parameters the header uses change between two headers (Title, Author, headerLine1, headerLine2, headerSeparator), more patterns will be generated, allowing still to have the smallest pdf file while updating the header contents on the fly.

- ○ Dynamic justification

  When using special "@@" tags (@@PageNo and @@TotalPages) with right or center alignment, they are now correctly positioned. Works with pdf_text_center, pdf_text_align & pdf_text_boxed_xy. Useful e.g. for headers and footers.

- ○ pdf_text_boxed_xy now takes into account its justification parameter to right align or center the text. pdf_text_boxed_xy last parameter (line weight) is now DECIMAL.

- ○ new parameter "reuseExternal" to enable many uses of an external pdf file opened once. That is to say that the first generated document will be done using pdf_new/pdf_open_pdf/pdf_use_pdf_page/pdf_close whereas all subsequent calls will be done without parsing again the template, like: pdf_new/pdf_use_pdf_page/pdf_close - see adobe-example.p sample. This allows to decrease the time needed to generate many similar documents based on the same template.

- ○ Performance optimizations

- ○ when pdf_wrap_text_xy* triggers a page change, wrap nicely at the top of the next page

- ○ TT_pdf_page.page_width & height are now DECIMAL instead of INTEGER (in particular it allows to "UseExternalPageSize" correctly).

API changes

None

Pdfextract.p

Bug fixes

- better support for multi-level encoded streams (e.g. hex encoded then compressed)

v4.1

pdf_inc.p

new functionalities

- Pictures:
- ○ added support for CMYK & Gray scale JPEG pictures (before, only RGB was supported)
- Encryption:
- ○ rewrote and optimized encryption code. Make use of new built-ins MD5-DIGEST, HEX-ENCODE & HEX-DECODE for PROVERSION >= 10.

  This rewrite allowed to make it work also in UTF-8 sessions, and to make RC-4 128 encryption work (it had never worked before).

- AES-128 encryption is now supported natively. It does not make use of any external dll, but OpenEdge >= 10 is needed.

  A new parameter "EncryptAlgorithm" has been added, possible values are RC4 & AES.
- encryption now works when using external pdf templates: stream and strings from the external file are properly encrypted.

- pdf_merge_stream: support merging streams using external pdf files. Modified the clean-up code in order not to delete temp files (png images, pdf templates) when they are still in use, in the original or the merged file.

- Getting info from an existing pdf file:
  - pdf_new() can now be called with ? as a filename, thus allowing to write code to get information about an external pdf without the need of producing any pdf file (see pdfinfo.p).
  - new API pdf_ext_get_path allows to get any information from an external pdf, using "pdf paths" like "/Root/Pages/Kids[1]/Cropbox" or - more useful - /Root/Pages/Count.
  - new API pdf_ext_get_page returns a string representation of a page object. This page object can be used as a start for the path in pdf_ext_get_path (this is a shortcut for "/Root/Pages/Kids[n]")

bug fixes

- fix a regression in pdf_set_TextRed/Green/Blue and pdf_set_FillRed/Green/Blue introduced in earlier optimizations.
- pdf_text_to was not positioning the text correctly for proportional fonts. Changed FILL(" ",...) to FILL("E",...) to match pdfInclude 3 behaviour.
- pdf_text_at also was sometimes not positioning the text correctly. Make use of Tm instead of TD to place the text.
- fix a stupid bug in pdf_text_to which had been introduced at the output optimization time
- fix font widths for all variable fonts. Some (e.g. accentuated) characters did not have any width defined.

  Make Symbol and ZapfDingbats variable and add proper widths.
- when using pdf_load_template, some of the specified fonts were forgotten in the pdf file.

API changes

None

Pdfextract.p

- Modified loadDictionary in order to support malformed pdf files where the same key might happen more than once in a dictionary.

Bug fixes

- in ReadWord(), added UNFORMATTED after each IMPORT (else a simple dot in the file would make the load to stop!)
- resolveIndirects: do not treat anymore unknown indirect objects as an error as per the PDF spec.
- added obj_stream & pdf_id fields to TT_ObjStms, and fixed queries for TT_Object & TT_ObjStms which were missing the test for obj_stream = pStream AND pdf_id = pID.
- The xref stream was incorrectly extracted in some cases. This has been fixed both

in outputStream & LoadObjectPointers.
- make LoadObjectPointers respect better the spec
- There was a buffer leak (TT_Object) in LoadObjectPointers which in some cases prevented the code following to work properly.

v4.0

pdf_inc.p

new functionalities

- Pictures:
  - PNG support, including transparency, either one transparent color or a full alpha channel

    new private procedure pdf_extract_image_info replaces pdf_get_image_wh.
  - pdf_place_image now uses the image dimensions if they have not been specified as parameters
- by default, the generated PDF is v1.4. Calling new procedure pdf_set_MinPdfVersion allows to set a minimum version.

  Currently used when using 16 bits png pictures (supported starts with PDF v1.5)
- Added pdf_get_widgets API to return the list of fillable fields in a pdf form
- Added support for 1254 code page (Turkish) - new file 1254.diff
- New transaction mechanism, using pdf_transaction_begin/rollback/commit. The use of transactions is demonstrated in pdftool.p, for the TABLE tool. The MATRIX tool demonstrate the use of pdf_transaction_buffer.
- pdf tools: implement wrap in cells for the TABLE & the MATRIX tools.
- the tag functionality now allows for underline <u>, strike <s> and links <url=>. This also works for rotated text.

  <url=> can be used for http:// links but also to create internal links. In order to achieve this, create a bookmark (named for example "my_bookmark"), then use <url=#my_bookmark>my link</url> whenever you want to create a link.
- add pdf_get_parameter2, same as pdf_get_parameter with a default value (also added pdf_get_tool_parameter2)
- pdf_wrap_text_xy[_dec] now returns (as return-value not to break the API) the Y coordinate of the last line
- new APIs pdf_incr_parameter and pdf_decr_parameter to increment/decrement any integer parameter
- Fill pdf forms
  - take into account the field justification (left, center, right) as defined in the template; also make use of the "multiline" flag defined in the template. Adjust the text position within the fields.
  - add the ability to fill check-boxes and radio buttons.

    Only one radio button may be checked within one given radio group.
  - add support for combo and list boxes.
  - it is now possible to retain the original form in the created document! This allows e.g. the creation of pre-filed forms. See the "retainAcroForm" parameter.
  - new parameter "formFlattenWithDefaultValues" allows to keep the form default values when flattening the widgets (can-do list matching widget names)
  - new parameter formFlatten (obsoletes retainAcroForm): CAN-DO list of form widgets' names to flatten. Set to "" not to flatten anything (same as

retainAcroForm = "TRUE").

If is now possible to selectively flatten some widgets and keep some others in the resulting pdf.

- It is now possible to retain the annotations from the original pdf document (e.g. links, sticky notes...) using the "retainAnnots" parameter.
- integration with PLOP (PDF Linearization, Optimization, Protection), a commercial product from PDFlib GmbH (PSP has been made obsolete), using pdf_Encrypt.

Using PLOP, AES-128 & AES-256 encryption is now available (along with RC4-128).

bug fixes

- pdf_text_widthdec2 had bugs when the string contained a parenthesis (because it was protected by a backslash).
- if something went wrong in pdfextract.p and the file for an external page has not been generated, does not crash anymore
- pdf_load_external: fix bug for UTF-8 sessions
- LoadExternalXObjects: fix bug when the external pdf template contains two images with the same name
- pdf_text_width[2] & pdf_font_width[2] now remove the tags before computing the text width, using new private procedure pdf_strip_tags. Thus the width is now correct.
- pdf_wrap_text_xy[_dec]: when there was too much text to fit in the given box, one line too much was printed.
- compression & decompression using zlib: trap and manage return codes <> 0 (zlib error)

optimizations

- big rewrite to optimize the output of the text operators, up to 40% of reduction in the size of the generated pdf (setTextOperator)
- idem for graphic operators (setGfxOperator)

API changes

- pdf_curve, pdf_circle & pdf_ellipse now take their parameters as DECIMALs
- [de]compressfile return value is now integer instead of logical; it returns zlib return code.
- pdf_encrypt API has changed to reflect changes between PSP & PLOP

Also the permission list is now separated by "," instead of ";".

Pdfextract.p

new functionalities

- **Almost totally rewritten!**

Now, should supports every pdf even with version > 1.4, containing XRef streams and/or Object streams (with a few limitations concerning deprecated pdf functionalities).

The parser is way much more tolerant to the differences encountered in different pdf file from different producers.

The code is much smaller (as is pdf_inc.p) because with the new approach, it is not needed to "understand" what is in the objects (especially fonts).

- Added filters for /AsciiHexDecode and /Ascii85Decode.

- Get from the field definition its justification (left, center, right) and whether the field is multiline or not.
- Get all the needed info in TT_Widget so that we can fill check-boxes, radio buttons, combo and list boxes (done in pdf_inc.p)
- Add code to support for retaining the original AcroForm, and any type of annotation (link, popup...)
- Take into account new parameter FormFlatten (which replaces retainAcroForm)

Bug fixes

- mediaBox and cropBox did not work properly when the values were decimal
- fix error propagation for doc-encrypted
- when readWord encounters a CHR(0), replace it with \000 instead of changing all the string to hexa
- LoadPointers: ensure we do not create twice the same object (same obj)

v3.3

This is the old version, available on OEHive.org/pdfInclude as free software (Eclipse licence).