PDFX

Developer guide

Writing PDF files in pure AutoIt, with no dependency

Version 1.4.0 · AutoIt 3.3.16.1 · ISO 32000-1 (PDF 1.7)

Numeric1 · https://www.autoitscript.com/forum/

PDFX writes the file itself — objects, cross-reference table, trailer — from a single #include. No COM component, no printer driver, no DLL.

42	4	1	0
public functions	worked examples	include file	external dependencies

IN THIS GUIDE

- The document model and the coordinate system
- Text with exact AFM metrics, colours, shapes, images and tables
- Navigation: bookmarks, forward links and opening mode
- Saving, compression, validation and the complete API reference

Produced by the engine it documents: contents, bookmarks and links are generated by PDFX.

Contents

Every entry is clickable; every heading is also a bookmark in the reader pane.

01 **Why PDFX** 03

02 **Quick start** 04

03 **The document model** 05

04 **Coordinates and units** 07

05 **Text and fonts** 08

06 **Colour, stroke and opacity** 09

07 **Shapes** 10

08 **Images** 11

09 **Tables** 13

10 **Header, footer and metadata** 14

11 **Navigation** 15

12 **Saving and validity** 17

13 **Compression** 18

14 **Testing and validation** 20

15 **Performance and limits** 21

16 **API reference** 22

17 **References and licence** 26

Chapters 1 to 15 build the mental model; chapter 16 is the reference to keep at hand.

CHAPTER 01

Why PDFX

A PDF writer for AutoIt, written from the specification.

AutoIt handles text files, CSV, INI and GUIs very well. When the result has to be printed, e-mailed or archived with a stable layout, the format that opens everywhere is PDF. PDFX produces that file directly, from the script.

The usual routes to a PDF are indirect: drive Word or Excel through COM, install a virtual printer, or ship a third-party DLL. Each one adds a dependency that may be missing on the target machine, a licence to review, or a version to maintain. PDFX has none of them: one include file, no registration, no temporary document.

What the engine covers

Area	What PDFX provides	What it does not
Text	Core 14 fonts, WinAnsi, exact metrics, wrap and align	TTF/OTF embedding
Vector graphics	Lines, rectangles, ellipses, polylines, Bezier	Gradients, clipping paths
Images	JPEG passthrough, PNG/BMP/GIF via GDI+	JPEG 2000, SVG
Layout	Paginated tables, headers, footers	Floating boxes, columns
Navigation	Bookmarks, internal and external links	Named destinations, JavaScript
Metadata	Title, author, subject, keywords, UTC dates	XMP, PDF/A
Output	PDF 1.7, FlateDecode, deterministic /ID	Encryption, signatures, forms
Reading	Nothing — the engine only writes	Merge, extract, fill

Design rules

- One handle, no globals: every call receives the document and changes it. Two documents can be built at the same time.
- Top-left coordinates: the number you write is the position you see; the conversion to PDF space happens once, at emission.
- Exact metrics: text is measured on the real AFM tables of the standard fonts, never estimated from a character count.
- Bounded inputs: a colour component or an opacity outside its range is clamped and reported with @error = 1, never silently wrapped.
- Deterministic bytes: the document identifier is a fingerprint of the content, so two runs produce the same file and a regression test can compare bytes.

SCOPE

PDFX is a writer, not a toolkit. Chapters 15 and 17 state the limits precisely: no reading, no encryption, no signatures, no PDF/A. Within that scope the output is complete and standards-compliant, and it opens identically in Acrobat, Chrome, Edge, Firefox, SumatraPDF and Preview.

CHAPTER 02

Quick start

Five lines from an empty folder to a file that opens.

Copy PDFX.au3 next to your script, or into an include folder. AutoIt 3.3.16.1 is the reference version. Crypt.au3 and GDI+ ship with AutoIt and are used only to compute the document identifier and to convert non-JPEG images; nothing else is required.

```
#include "PDFX.au3"

Global $pdf = _PDFX_New("A4", "P", 20)         ; A4 portrait, 20 mm margins
_PDFX_SetFont($pdf, "Helvetica", 18, "B")
_PDFX_Text($pdf, _PDFX_Mm(20), _PDFX_Mm(30), "Hello from PDFX")

_PDFX_Save($pdf, @ScriptDir & "\hello.pdf")
_PDFX_Close($pdf)
```

_PDFX_New creates the document and its first page. The drawing calls append to the current page; _PDFX_Save freezes the page, applies the header and footer, and writes the file. _PDFX_Close releases GDI+ if the engine started it, which matters only at the end of a script that uses images. At the top level of a script the variable is Global; inside a function, declare it Local.

Check the result

The Tools folder contains three independent Python scripts with no third-party dependency. The first one reads the file structure and reports anything invalid:

```
python Tools/pdfcheck.py hello.pdf
hello.pdf: 1035 bytes, 7 objects, 1 stream (0 compressed), 0 bookmarks, 0 internal links, OK
```

The counts vary with the content. OK at the end means the file is consistent.

IF NOTHING HAPPENS

A script that closes with no output has usually hit a syntax error or a duplicated function name. Run Au3Check.exe PDFX.au3 (or press Ctrl+F5 in SciTE) to see the line. An out-of-range array index stops AutoIt silently with exit code 1: check the row and column indices of your tables.

The four examples

File	What it demonstrates	Pages
01_Report.au3	Cover band, KPI cards, bar chart, paginated table, shapes, alpha	2
02_Invoice.au3	Business document, five-column table, totals block, link	1
03_Images.au3	JPEG passthrough, PNG conversion, three sizing rules, dedup	1
04_Manual.au3	Contents written before chapters, forward links, 11 bookmarks	7

Each example is a complete script of fewer than 200 lines. Reading 01_Report.au3 after this guide is the fastest way to see the API in context.

CHAPTER 03

The document model

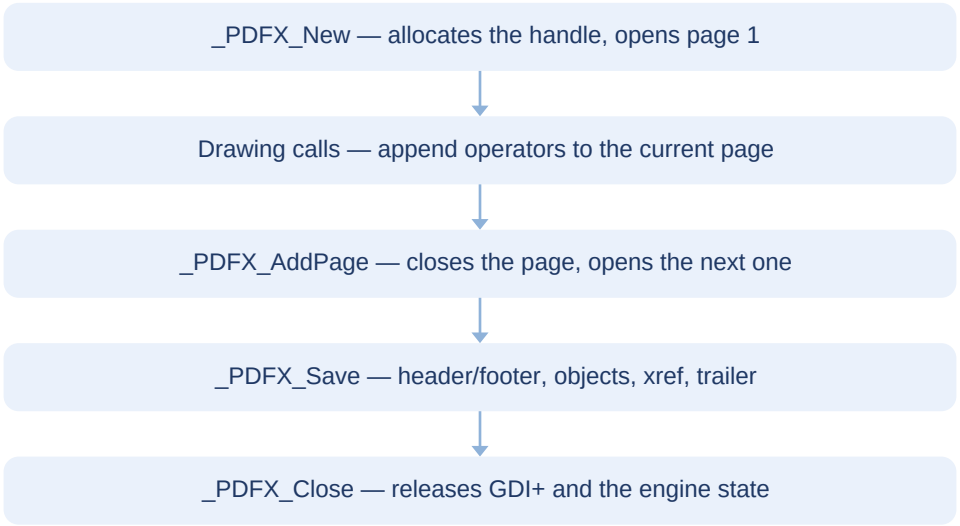
One array holds everything; every function receives it.

A document is a plain AutoIt array. It contains the pages, the font and image resources, the current graphic state, the bookmarks, the pending links and the metadata. AutoIt passes arrays by reference, so no ByRef is needed at the call site: the handle is always the first argument.

```
Local $a = _PDFX_New("A4")   ; page 1 exists
Local $b = _PDFX_New("A5", "L") ; a second, independent document
_PDFX_Text($a, 56, 56, "In A")
_PDFX_Text($b, 40, 40, "In B")
```

The document model (continued)

Lifecycle



Nothing reaches the disk before _PDFX_Save. A document is written only once: after a successful save, _PDFX_AddPage and a second _PDFX_Save return @error = 2 instead of producing an inconsistent second file. If the file cannot be created, nothing is frozen and the call can be retried.

Error convention

A failed call returns False (or -1, or 0, depending on its normal return type) and sets @error. The two values used throughout the API are:

@error	Meaning	Typical call
1	Input was clamped or is invalid; the call still succeeded	_PDFX_SetAlpha($d, 1.4) ? opacity 1
2	The document is already saved and is therefore frozen	_PDFX_AddPage($d) ? -1
0	Success	—

WHY A FROZEN DOCUMENT

Allowing a second save would silently produce two files whose xref tables claim the same object numbers. Returning an error is the cheapest way to make the mistake visible. Build a new document with _PDFX_New when the content changes.

CHAPTER 04

Coordinates and units

The origin is at the top left, where you expect it.

PDF itself places the origin at the bottom-left corner and grows upwards. That convention comes from printing; it is awkward when a script thinks in screen coordinates. PDFX takes every Y coordinate from the top of the page and converts it once, when the operator is written.

The unit is the point, 1/72 inch, in both worlds. _PDFX_Mm converts millimetres; _PDFX_Mm(20) is 56.69 points. The accessors _PDFX_PageWidth(), _PDFX_PageHeight() and _PDFX_Margin() return points as well.

Text is anchored by its top

_PDFX_Text($pdf, x, y, text) places the top of the text at y, not the baseline. The visible top of a 12-point line lands where you asked, so blocks of text stack predictably: to place the next paragraph, add the leading to y, or simply use the value returned by _PDFX_TextWrap.

```
Local $y = _PDFX_Mm(40)
$y = _PDFX_TextWrap($pdf, _PDFX_Mm(20), $y, _PDFX_Mm(170), $intro, 10)
$y = _PDFX_TextWrap($pdf, _PDFX_Mm(20), $y + _PDFX_Mm(4), _PDFX_Mm(170), $body, 10)
```

Page formats

Format	Millimetres	Points	Note
A4	210 × 297	595.28 × 841.89	default
A3	297 × 420	841.89 × 1190.55	
A5	148 × 210	419.53 × 595.28	
Letter	215.9 × 279.4	612 × 792	
Legal	215.9 × 355.6	612 × 1008	
Custom	"WxH" in mm	converted	e.g. "120x200"

The third argument of _PDFX_New is the margin, in millimetres. It is used by the header, the footer, and by the automatic page break of _PDFX_TextWrap and _PDFX_Table. It does not clip: a rectangle drawn at x = 0 covers the whole width, which is how a full-bleed cover band is made.

Orientation and custom sizes

Pass "P" (portrait) or "L" (landscape) as the second argument. _PDFX_AddPage accepts an explicit width and height in points for mixed-format documents: _PDFX_AddPage($pdf, _PDFX_Mm(148), _PDFX_Mm(210)) adds an A5 page after an A4 one.

CHAPTER 05

Text and fonts

Three families, fourteen fonts, exact measurements.

The engine uses the fourteen standard PDF fonts. They are not embedded: every reader already has them, so the file stays small and no licence has to be managed. The default font is Helvetica 10, black, left aligned.

Families and styles

Family	Aliases	Styles	Use
Helvetica	Arial, sans	normal, B, I, BI	running text, UI
Times	serif	normal, B, I, BI	body copy, quotes
Courier	mono	normal, B, I, BI	code, columns

```
_PDFX_SetFont($pdf, "Times", 11, "I")   ; family, size, style
_PDFX_Text($pdf, 56, 72, "Text with a top-anchored Y")
_PDFX_TextAlign($pdf, 56, 100, _PDFX_Mm(80), "Centred", "C")
Local $w = _PDFX_TextWidth($pdf, $label, 11) ; exact width, in points
```

Encoding

Text is written in WinAnsi, the Windows code page 1252. French, German, Spanish and Italian accented letters pass through unchanged, with the guillemets, the euro sign, the degree sign, the plus-minus sign, the middle dot, the ellipsis and the em dash. A character outside WinAnsi (CJK, Cyrillic, emoji) becomes a question mark rather than corrupting the file. Save the .au3 file as UTF-8 so that accented literals arrive intact.

Three text functions

- _PDFX_Text writes one line at a given point. No wrapping, no clipping: the caller decides where the line ends.
- _PDFX_TextAlign writes one line inside a horizontal box, aligned left, centre or right. It is the tool for table-like layouts.
- _PDFX_TextWrap wraps a paragraph at a width and returns the top of the next free line, which makes chained paragraphs natural.

_PDFX_TextWrap accepts a leading (0 means 1.2 times the size), an alignment, and a page-break flag. With the flag set to True, a paragraph that reaches the bottom margin adds a page and continues on it. The continuation starts at the document margin: when the page carries a running header, leave the flag off and manage the break yourself, as the examples do.

WHY AFM TABLES

A width estimated as characters × 0.5 em drifts by several points per line: right-aligned columns stop lining up and centred titles look off-centre. PDFX carries the AFM widths of the fourteen standard fonts, so _PDFX_TextWidth returns the exact advance needed by the alignment and by the word wrapper.

CHAPTER 06

Colour, stroke and opacity

A small graphic state, changed by explicit calls.

Fill and stroke are two separate colours. _PDFX_SetFill paints the interior of shapes and the glyphs of text; _PDFX_SetStroke paints lines, outlines and table borders. Both accept the same four forms: an integer 0xRRGGBB, a colour name, an array [R, G, B], or the string "R,G,B".

```
_PDFX_SetFill($pdf, 0x1F3864)          ; hex
_PDFX_SetStroke($pdf, "navy")           ; name
_PDFX_SetFill($pdf, "46,117,182")       ; string components
Local $aNavy[3] = [31, 56, 100]
_PDFX_SetStroke($pdf, $aNavy)           ; zero-based array
```

An array is read from index 0, so a StringSplit() result is not suitable: its first element is the count. Build the array explicitly, as above. Names are matched case-insensitively: black, white, red, green, blue, gray (or grey), lightgray, navy, teal, orange, purple (or violet), yellow, silver, maroon, olive, lime, aqua (or cyan),

fuchsia (or magenta). An unknown name is treated as an integer and gives black.

Clamping

Components outside 0..255, or an opacity outside 0..1, are clamped into range and reported once: the call succeeds, @error is 1. The result is always a valid colour; the error tells you that the value you passed was not what you meant.

Width, dash and opacity

Call	Effect	Default
_PDFX_SetLineWidth($d, $w)	Stroke width in points; 0 or less becomes 0.1	1
_PDFX_SetDash($d, $on, $off)	Dash pattern; $off = 0 reuses $on; $on = 0 restores solid	solid
_PDFX_SetAlpha($d, $a, $stroke)	Fill and stroke opacity; negative stroke = same as fill	1
_PDFX_SetDash($d, 0)	Emits [] 0 d and restores a solid line	solid

Opacity is implemented with an ExtGState entry, /ca for the fill and /CA for the stroke. The engine creates one object per distinct pair of values and reuses it, so repeating the same opacity costs nothing. Set it back to 1 before drawing content that must stay opaque.

STATE IS SEQUENTIAL

There is no stack of graphic states to push and pop. Each call changes the current state until the next call changes it, exactly like a drawing program. Set the colour immediately before the shape that needs it.

CHAPTER 07

Shapes

Seven primitives, one style flag.

Every shape receives a style: $PDFX_STROKE draws the outline, $PDFX_FILL paints the interior, $PDFX_BOTH does both. The fill colour is the one set by _PDFX_SetFill, the outline uses _PDFX_SetStroke, the current width and the current dash pattern.

Shapes (continued)

Call	Geometry	Style
_PDFX_Line($d, x1, y1, x2, y2)	Segment; stroke only	—
_PDFX_Rect($d, x, y, w, h, $style)	Axis-aligned rectangle	all
_PDFX_RoundRect($d, x, y, w, h, $r, $style)	Rectangle with a corner radius	all
_PDFX_Ellipse($d, cx, cy, rx, ry, $style)	Centre and two radii	all
_PDFX_Circle($d, cx, cy, $r, $style)	Centre and radius	all
_PDFX_Polyline($d, $points, $close, $style)	[n][2] array of x, y pairs	all
_PDFX_Bezier($d, x1, y1, c1x, c1y, c2x, c2y, x2, y2, $style)	Cubic curve, two control points	all

```
_PDFX_SetFill($pdf, 0xE8EEF7)
_PDFX_SetStroke($pdf, 0x1F3864)
_PDFX_SetLineWidth($pdf, 1.2)
_PDFX_RoundRect($pdf, _PDFX_Mm(20), _PDFX_Mm(40), _PDFX_Mm(55), _PDFX_Mm(22), _PDFX_Mm(4), $PDFX_BOTH)

Local $aPoints[4][2] = [[10, 10], [40, 10], [40, 30], [10, 30]]
_PDFX_Polyline($pdf, $aPoints, True, $PDFX_FILL)
```

Coordinates always describe the bounding box or the centre in the top-left system, so a shape placed at y = 40 mm starts 40 mm from the top edge. Curves are written with the native PDF operator c; there is no approximation by line segments.

TEXT USES THE FILL COLOUR

There is no separate text colour. _PDFX_SetFill sets the colour of both text and shape interiors, which keeps the mental model small: one fill, one stroke, shared by everything drawn afterwards.

CHAPTER 08

Images

JPEG bytes are copied, not re-encoded.

_PDFX_Image inserts a JPEG, PNG, BMP or GIF file. The path can be relative to the script; build it from @ScriptDir when the document is produced from another working directory.

Images (continued)

Two paths, one object

Input	Path through the engine	Quality	Speed
JPEG	Stream copied byte for byte, /DCTDecode	unchanged	instant
PNG, BMP, GIF	Drawn on white by GDI+, encoded as JPEG	JPEG q?85	slower
TIFF	Same GDI+ path when the codec is installed	JPEG q?85	slower

A JPEG is never decoded. The engine reads its SOF marker to learn the pixel size and the number of components: 1 is greyscale, 3 is RGB, 4 is CMYK. A CMYK file carrying the Adobe APP14 marker is declared with /Decode [1 0 1 0 1 0 1 0], because Adobe stores inverted values; without the marker the PDF order is used. The result is a PDF whose embedded image is exactly the original file.

PNG, BMP and GIF files are converted through GDI+ to JPEG on a white background. The alpha channel is flattened: a transparent logo becomes white, which is the expected behaviour for print. A PNG image is therefore stored approximately at the size of an equivalent JPEG, not at its original PNG size.

Sizing rules

Arguments	Result	Example
width, height = 0	Natural size at 96 dpi	0, 0
width only	Height deduced from the aspect ratio	48 mm, 0
height only	Width deduced from the aspect ratio	0, 30 mm
both	Exact rectangle: proportions are not preserved	48, 24 mm

```
_PDFX_Image($pdf, $jpg, _PDFX_Mm(20), _PDFX_Mm(40), _PDFX_Mm(48), 0)   ; height inferred
_PDFX_ImageFit($pdf, $logo, _PDFX_Mm(80), _PDFX_Mm(40), _PDFX_Mm(50), _PDFX_Mm(50), "C")
Local $a = _PDFX_ImageInfo($pdf, $jpg)
ConsoleWrite($a[0] & " x " & $a[1] & " px, " & $a[4] & " bytes, " & $a[5] & @CRLF)
```

_PDFX_ImageInfo returns width and height in pixels (0 and 1), the natural size in points at 96 dpi (2 and 3), the file size in bytes (4) and the extension in upper case (5). It registers the image in the document even if nothing is drawn, which is harmless.

Deduplication

Images are cached by absolute path. Inserting the same file ten times creates one XObject and ten references to it: the file size grows once. Copying a file to a second name defeats the cache, because the key is the path.

Images (continued)

PROGRESSIVE JPEG

Progressive files are accepted but rendered unevenly by some readers. Saving the image as baseline JPEG (GDI+ does this by default) removes the ambiguity. The engine does not re-encode a JPEG, so it cannot convert one for you.

CHAPTER 09

Tables

Columns as fractions, pagination included.

_PDFX_Table draws a header row, a grid and the data, and paginates by itself. Columns are described by an [n][3] array: title, width fraction, alignment. Fractions are normalised, so 0.6 / 0.15 / 0.25 and 6 / 1.5 / 2.5 produce the same layout.

```
Local $aCols[3][3] = [ _
    ["Item",  0.60, "L"], _
    ["Qty",   0.15, "C"], _
    ["Total", 0.25, "R"]]

Local $aRows[2][3] = [ _
    ["Design", "1", "1,450.00"], _
    ["Print",  "2",   "640.00"]]

Local $y = _PDFX_Table($pdf, _PDFX_Mm(20), _PDFX_Mm(80), _PDFX_Mm(170), $aCols, $aRows, _
    "padding=5;fontsize=9;zebra=1;headfill=0x1F3864;headcolor=0xFFFFFF")
```

Tables (continued)

Options

Option	Default	Role	Range
padding	4	Inner margin of a cell, in points	0..
fontsize	current	Body size; the header uses the same size in bold	> 0
headfill	0xE8EEF7	Header background; -1 removes it	colour
headcolor	0x1F3864	Header text colour	colour
zebra	0	1 alternates row backgrounds	0 / 1
zebracolor	0xF7F9FC	Background of odd rows	colour
border	0.5	Grid line width; 0 removes the lines	0..
bordercolor	0xBFC9D9	Grid colour	colour
textcolor	0x000000	Body text colour	colour
pagebreak	1	0 disables the automatic page break	0 / 1

The function returns the Y coordinate of the bottom of the last row, ready for the following block. With pagebreak = 1, a row that would cross the bottom margin starts a new page and the header is redrawn at the top of it. The break happens at the document margin, as with _PDFX_TextWrap, so a table on a page with a running header should either fit on one page or accept that the repeated header starts at the margin.

CELL CONTENT

Any value is converted with String(), so numbers, booleans and expressions can be placed directly in the array. A cell that is too long for its column wraps onto additional lines and the row grows; alignment applies to every line of the cell.

CHAPTER 10

Header, footer and metadata

Set once, repeated on every page at save time.

_PDFX_SetHeader and _PDFX_SetFooter each take three zones: left, centre and right. An empty string hides its zone, which makes a quiet pagination easy to compose. Three tokens are replaced when the document is saved: {page}, {pages} and {date}.

Header, footer and metadata (continued)

```
_PDFX_SetHeader($pdf, "ACME Ltd", "", "Annual report")
_PDFX_SetFooter($pdf, "", "Page {page} / {pages}", "{date}")

; size, family, style and colour are per document
_PDFX_SetFooter($pdf, "", "Page {page} / {pages}", "", 7, "Times", "I", 0x6B7280)
```

Geometry

The header starts at the document margin: its text is anchored there and a thin rule is drawn below it, at the margin plus 1.2 times the size plus 3 points. The footer sits in the bottom margin, its text about 14 mm above the edge for a 20 mm margin. Both are drawn after the page content, so a coloured band placed at the top of the page does not hide them.

That ordering has a consequence worth knowing: the header is applied to every page, including a cover. If the cover carries a full-bleed coloured band, start the band below the header (around 36 mm) or accept grey text over the colour. The examples and this guide start the band below the header strip.

Metadata

_PDFX_SetMeta fills /Title, /Author, /Subject and /Keywords. ASCII strings are written in literal form; anything else is written in hexadecimal UTF-16BE with a byte-order mark, so a title containing €, œ or a CJK character stays exact in the reader's window. /CreationDate and /ModDate are in universal time and end with Z, which makes the file independent of the machine's time zone.

The document identifier in the trailer is an MD5 fingerprint of the content, the metadata and the navigation data. Two runs over the same document produce the same /ID, so the bytes can be compared in a regression test. Only the timestamps need neutralising, and Tools/regress.py does that.

CHAPTER 11

Navigation

A PDF can carry its own table of contents.

Navigation in PDFX has three parts: bookmarks, which appear in the reader's pane; links, which make a rectangle clickable; and the opening mode, which decides what the reader shows first.

Navigation (continued)

Bookmarks

_PDFX_AddBookmark($pdf, $title, $level, $page, $y) adds an entry. The page is one-based, as everywhere in the API; the level is zero-based. Without a Y, the destination is /Fit and the whole page is shown. With a Y, it becomes /XYZ and the reader places that point at the top of the window.

```
_PDFX_AddBookmark($pdf, "Cover", 0, 1)
_PDFX_AddBookmark($pdf, "Chapter 1", 1, 3)
_PDFX_AddBookmark($pdf, "Section 1.1", 2, 3, _PDFX_Mm(120))
_PDFX_SetPageMode($pdf, "UseOutlines")
```

A bookmark cannot descend more than one level below the previous one. A request that goes deeper is clamped to that maximum without an error, so an outline built in a loop always stays valid. The function returns the ordinal number of the new bookmark, or -1 with @error = 1 when the title is empty or the page is outside the document.

The tree is complete: /Parent, /Prev, /Next, /First, /Last and /Count. The /Count of a node is its number of descendants, and the root counts them all. Counts are positive, so every node is open. Titles that are pure ASCII are written literally; others use hexadecimal UTF-16BE, so an accented title stays exact in the pane. When bookmarks exist and no mode was requested, /PageMode /UseOutlines is added automatically.

Links

_PDFX_Link adds a URL annotation. _PDFX_LinkPage adds an internal link and accepts a target page that does not exist yet: the engine keeps the request pending and resolves it at save time, when every page has its object number. A table of contents can therefore be written before the chapters it points to, without a second pass. A target that never materialises is dropped silently; only a page below 1 is refused immediately.

```
; written on the contents page, before the chapters exist
_PDFX_LinkPage($pdf, $x, $y, $w, $h, 5, _PDFX_Mm(32))

; an external link
_PDFX_Link($pdf, $x, $y, $w, $h, "https://www.autoitscript.com/forum/")
```

Opening mode

Call	Accepted values	Default
_PDFX_SetPageMode	UseNone, UseOutlines, UseThumbs, FullScreen	automatic
_PDFX_SetPageLayout	SinglePage, OneColumn, TwoColumnLeft, TwoColumnRight, TwoPageLeft, TwoPageRight	reader

Case and spaces are ignored in both lists; an empty string restores the automatic choice.

After _PDFX_Save, the navigation functions return @error = 2: the document is frozen. The four examples

Navigation (continued)

and this guide all use the deferred mechanism; Tools/outline.py reads the result back without any dependency:

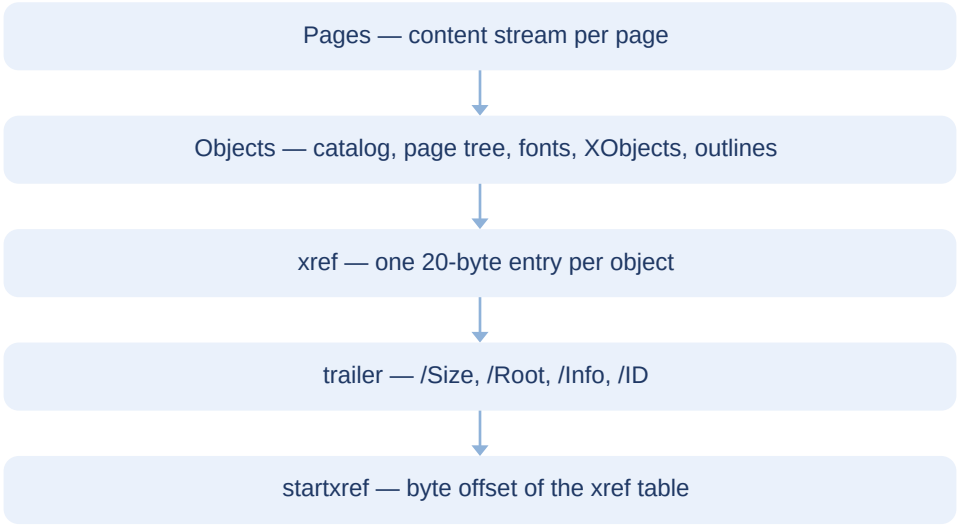
```
python Tools/outline.py Examples/04_Manual.pdf
Cover                              page 1  /Fit
Contents                           page 2  y=739.843
Getting started                    page 3  y=717.165
  Installation                     page 3  y=467.717
---- 11 bookmark(s)
```

CHAPTER 12

Saving and validity

Offsets, xref, trailer: the file is assembled once.

_PDFX_Save is the only function that touches the disk. It freezes the current page, appends the header and footer to every page, serialises the objects, then writes the cross-reference table, the trailer and startxref. The /Length of every stream is the number of bytes actually written.



Saving and validity (continued)

What the reader checks

- The header %PDF-1.7 and the binary comment on the second line.
- startxref points to the literal token xref, and each entry points to the bytes %d 0 obj.
- Each /Length matches the stream between stream and endstream, including the line break.
- The trailer contains /Root and /Size, and /Size equals the number of xref entries plus one.
- The page tree /Count equals the number of /Kids, and every /Kids entry is a page object.

Tools/pdfcheck.py verifies all of this, and inflates every FlateDecode stream with zlib: a corrupt stream or a wrong length is reported, which validates the compressor end to end. It also checks the outline chain, the /Count of each node, the /Annots arrays and every internal destination.

```
python Tools/pdfcheck.py Examples/01_Report.pdf
01_Report.pdf: 5775 bytes, 14 objects, 2 streams (2 compressed), 0 bookmarks, 0 internal links, OK
```

One document, one save

A document is written once. After a successful save, _PDFX_Save and _PDFX_AddPage return @error = 2 instead of writing a second, contradictory file. If the output path cannot be created, nothing is frozen: fix the path and call again. The examples delete the target first when the file may still be open in a reader on Windows.

DETERMINISM

The /ID is an MD5 digest of the version, the metadata, the page content and the navigation data. Two generations of the same document therefore produce the same bytes, timestamps aside. Tests/Reference keeps one PDF per suite, and Tools/regress.py compares them byte for byte after neutralising /CreationDate, /ModDate and the {date} token.

CHAPTER 13

Compression

FlateDecode written in AutoIt, with no DLL.

Text, vectors and table grids produce long content streams full of repeated operators. _PDFX_SetCompression($pdf, $level) writes those streams with FlateDecode: a zlib wrapper around DEFLATE, implemented in the include file itself.

Compression (continued)

```
_PDFX_SetCompression($pdf, 6)   ; 0 = raw, 1..9 = increasing effort
```

Level	Search	Use
0	none — streams stay readable in a text editor	debugging, teaching
1..3	few LZ77 candidates	drafts, interactive generation
6	balanced	recommended default
9	longest chains	archives

How it works

The compressor scans the stream with a 3-byte hash table and a chain of previous positions. The level bounds the number of candidates examined, which is the only difference between levels: the Huffman table is the fixed DEFLATE table, so decompression is the same fast path for every reader. A stream that does not shrink is written uncompressed, so /Filter is not guaranteed on every page.

Only content streams are compressed. JPEG images stay as they are — they are already compressed — and dictionaries, fonts and metadata are too small to gain anything. The two example documents below are text-heavy; the gallery is dominated by a JPEG, hence the small gain.

Document	Raw	Level 6	Gain
01_Report.pdf	18,505 B	5,775 B	about 3.2×
02_Invoice.pdf	10,891 B	3,348 B	about 3.3×
03_Images.pdf	57,906 B	53,220 B	about 1.1× (JPEG)

SPEED

The compressor runs at roughly 30 KB/s, which is the price of pure AutoIt. It reaches the ratio of zlib level 1. For a very large document, lower the level or leave compression off; rendering is identical either way. The output remains deterministic at every level.

CHAPTER 14

Testing and validation

Three tools and four suites, all dependency-free.

A PDF writer is easy to write and hard to trust. PDFX ships with three small Python scripts that read the output independently of the engine, plus four AutoIt test suites that exercise the API.

The tools

Script	What it verifies	Dependency
Tools/pdfcheck.py	Header, xref offsets, stream lengths, trailer, page tree, outline, annotations, internal destinations	Python 3
Tools/outline.py	Reads /Outlines, decodes titles, resolves each destination to a page number	Python 3
Tools/regress.py	Compares reference PDFs byte for byte, dates and {date} neutralised	Python 3

```
python Tools/pdfcheck.py Examples/*.pdf
python Tools/outline.py Examples/04_Manual.pdf
python Tools/regress.py Tests/Reference Examples Tests
```

The suites

Suite	What it covers	Checks
Tests/01_Torture.au3	Nine pages of worst cases: long words, accents, every shape, every font	runs to end
Tests/02_Robustness.au3	Clamping, frozen document, invalid arguments, UTF-16 metadata	23/23
Tests/03_Compression.au3	Round-trip of compressed streams, gain, determinism	16/16
Tests/04_Navigation.au3	Bookmark levels, counts, forward links, opening modes	36/36

Tools/regress.py is the safety net for engine changes. Each reference PDF is regenerated by its suite, normalised, and compared byte for byte. A single changed byte prints DIFFERENT with both digests, which makes an accidental change visible before it reaches a user. The rule for the engine is simple: unless a behaviour is intentionally changed, the reference files must stay identical.

ADDING A TEST

Write a small script that saves a PDF to Tests\, run it once, copy the file into Tests\Reference\, and add the regeneration command to the regression run. The /ID makes the new file stable from the first run, so the reference does not need to be regenerated later.

CHAPTER 15

Performance and limits

What to expect, and what the engine deliberately omits.

PDFX keeps the whole document in memory until _PDFX_Save. For the documents it targets — reports, invoices, manuals, certificates — that is a few megabytes at most. The cost of a save is dominated by compression when it is enabled, and by image conversion when PNG or BMP files are used.

Limits

Area	Status	Reason
Reading, merging, splitting	absent	A writer is not a parser; out of scope by design
Encryption, signatures	absent	Requires cryptographic structures and key management
PDF/A, PDF/X	absent	Requires embedded fonts and colour profiles
TTF/OTF embedding	absent	Core 14 fonts cover the supported encodings
Transparency groups, masks	absent	Constant opacity through ExtGState only
Gradients, patterns	absent	Simulate with bands or an image
Named destinations, JavaScript	absent	Links and bookmarks target pages only
Attachments, /GoToR	absent	Navigation stays inside the document
Text extraction	absent	No reading; use a parser

Memory

Everything lives in one array until the last page is written: page content streams, image bytes, font references and the navigation tables. Memory grows with the amount of content, not with its complexity: ten thousand short lines cost less than one full-page photograph.

Images are the part worth planning for. A JPEG is kept as the original bytes, so a 2 MB photograph costs 2 MB. A PNG or BMP is converted once, at insertion, and only the resulting JPEG is kept; the source file is never held in memory. The same absolute path inserted again reuses that object.

Practical advice

- Reuse the font selection: fonts are cached per family, size and style, but switching between many sizes on every line produces many text objects.
- Insert each image once. Deduplication is by absolute path, so the same file inserted ten times costs one stream.
- Prefer JPEG for photographs: it is copied without recompression. Use PNG only for flat graphics, and remember that it becomes a JPEG on white.
- Enable compression for documents that leave the machine, and leave it off while debugging: an uncompressed content stream is readable in a text editor.
- Keep the number of pages reasonable. Each page has its own content stream and resources dictionary; a thousand-page document is possible but slow to build.

A report of two pages with a cover band, cards, a chart and a table takes well under a second to produce without compression, and a few seconds with level 6. The gallery example, which converts a PNG through GDI+, is dominated by the conversion, not by the PDF writer.

CHAPTER 16

API reference

All public functions, grouped by task.

Every public function takes the document handle as its first argument. Pass the variable returned by _PDFX_New to every call: the state travels with it. Optional arguments keep their default when omitted; passing 0 for a size or a width usually means "use the current value" or "infer it".

Signatures are written in their shortest readable form: $doc is the handle, called $oDocument in the source, and parameter names lose their type prefix. AutoIt binds arguments by position, so a call copied from here works as written.

API reference (continued)

Document

`_PDFX_New($format="A4", $orient="P", $margin=15)`

Creates a document and page 1. Format: A3, A4, A5, Letter, Legal or "WxH" in mm. Returns the handle.

`_PDFX_Mm($millimetres)`

Converts millimetres to points. Use it everywhere a coordinate is written.

`_PDFX_AddPage($doc, $w=0, $h=0)`

Adds a page, freezes the previous one, returns its 0-based index. @error = 2 when saved.

`_PDFX_PageCount($doc)`

Number of pages currently in the document.

`_PDFX_PageWidth($doc)` `_PDFX_PageHeight($doc)`

Dimensions of the current page, in points.

`_PDFX_Margin($doc)`

Default margin, in points, as given to _PDFX_New.

`_PDFX_SavePage($doc)`

Freezes the current page content without opening a new page.

Text

`_PDFX_SetFont($doc, $family="Helvetica", $size=0, $style="")`

Selects family, size and style (B, I, BI). Size 0 keeps the current size.

`_PDFX_Font($doc)` `_PDFX_FontSize($doc)`

Current PDF font name and current size, in points.

`_PDFX_TextWidth($doc, $text, $size=0)`

Exact width in points, from the AFM tables of the current font.

`_PDFX_Text($doc, $x, $y, $text, $size=0)`

Writes one line; y is the top of the text. No wrapping.

`_PDFX_TextAlign($doc, $x, $y, $w, $text, $align="L", $size=0)`

Writes one line aligned in a box: L, C or R.

`_PDFX_TextWrap($doc, $x, $y, $w, $text, $size=0, $lead=0, $align="L", $brk=False)`

Wraps a paragraph and returns the top of the next free line. $brk adds pages at the margin.

Graphic state

`_PDFX_SetFill($doc, $colour)`

Fill colour for shapes and text: 0xRRGGBB, name, [r,g,b] or "r,g,b". @error = 1 when clamped.

`_PDFX_SetStroke($doc, $colour)`

Stroke colour, same accepted forms as _PDFX_SetFill.

`_PDFX_SetLineWidth($doc, $width)`

Stroke width in points; 0 or less becomes 0.1.

API reference (continued)

`_PDFX_SetDash($doc, $on, $off=0)`

Dash pattern in points; $on = 0 restores a solid line, $off = 0 reuses $on.

`_PDFX_SetAlpha($doc, $alpha, $stroke=-1)`

Fill and stroke opacity in 0..1, through an ExtGState; negative stroke = same as fill.

Shapes

`_PDFX_Line($doc, $x1, $y1, $x2, $y2)`

Straight segment, stroke only.

`_PDFX_Rect($doc, $x, $y, $w, $h, $style=$PDFX_STROKE)`

Rectangle; style is $PDFX_STROKE, $PDFX_FILL or $PDFX_BOTH.

`_PDFX_RoundRect($doc, $x, $y, $w, $h, $r, $style=$PDFX_STROKE)`

Rectangle with rounded corners of radius $r.

`_PDFX_Ellipse($doc, $cx, $cy, $rx, $ry, $style=$PDFX_STROKE)`

Ellipse from its centre and two radii.

`_PDFX_Circle($doc, $cx, $cy, $r, $style=$PDFX_STROKE)`

Circle from its centre and radius.

`_PDFX_Polyline($doc, $points, $close=False, $style=$PDFX_STROKE)`

Polyline or polygon from an [n][2] array of x, y pairs.

`_PDFX_Bezier($doc, $x1, $y1, $c1x, $c1y, $c2x, $c2y, $x2, $y2, $style)`

Cubic Bezier curve with two control points.

Images

`_PDFX_Image($doc, $path, $x, $y, $w=0, $h=0)`

Inserts a JPEG, PNG, BMP or GIF. One missing dimension is inferred; both missing = natural size at 96 dpi.

`_PDFX_ImageFit($doc, $path, $x, $y, $w, $h, $align="C")`

Scales an image down to fit a box without distortion; align C, or L/R/T/B.

`_PDFX_ImageInfo($doc, $path)`

Returns [pxW, pxH, ptW, ptH, bytes, EXT] and registers the image in the document.

Tables

`_PDFX_Table($doc, $x, $y, $width, $columns, $rows, $options="")`

Paginated table with repeated header. Columns: [title, fraction, L/C/R]. Returns the bottom y.

API reference (continued)

Navigation

`_PDFX_Link($doc, $x, $y, $w, $h, $url)`

Clickable external URL over the given rectangle.

`_PDFX_LinkPage($doc, $x, $y, $w, $h, $page, $yTarget=-1)`

Internal link; the target page may not exist yet, it is resolved at save time.

`_PDFX_AddBookmark($doc, $title, $level=0, $page=-1, $y=-1)`

Adds an outline entry; returns its ordinal, or -1 with @error = 1. Without y, /Fit; with y, /XYZ.

`_PDFX_SetPageMode($doc, $mode="UseOutlines")`

Opening pane: UseNone, UseOutlines, UseThumbs or FullScreen.

`_PDFX_SetPageLayout($doc, $layout)`

Page layout: SinglePage, OneColumn, TwoColumnLeft/Right, TwoPageLeft/Right.

Metadata and page furniture

`_PDFX_SetMeta($doc, $title="", $author="", $subject="", $keywords="")`

Fills the /Info dictionary; non-ASCII text is written in UTF-16BE.

`_PDFX_SetHeader($doc, $l="", $c="", $r="", $size=8, $font="", $style="", $col=-1)`

Header on every page, with {page}, {pages}, {date}; a thin rule is drawn below it.

`_PDFX_SetFooter($doc, $l="", $c="", $r="", $size=8, $font="", $style="", $col=-1)`

Footer on every page; the size is shared with the header.

Output

`_PDFX_SetCompression($doc, $level=6)`

FlateDecode for content streams, 0..9; 0 leaves them raw. Applied at save time.

`_PDFX_Save($doc, $file)`

Writes the PDF 1.7 file. @error = 1 if the file cannot be created, 2 if already saved.

`_PDFX_Close($doc)`

Releases GDI+ if the engine started it. Call after the save.

CHAPTER 17

References and licence

Where the conventions come from.

PDFX is written from ISO 32000-1:2008, the specification of PDF 1.7, and from the public AFM metrics of the Core 14 fonts. The clauses that matter most to a reader of the source are listed below.

Clause	Subject	Used in
7.5	Objects, file structure, cross-reference table, trailer	save pipeline
7.5.8	Cross-reference table, 20-byte entries	pdfcheck
8.2	Graphic operators and state	shapes, colour
8.6	Images and image XObjects	JPEG, GDI+ conversion
9.4	Text objects, showing and positioning	text emission
9.6	Standard 14 fonts and encodings	AFM metrics, WinAnsi
12.3 / 12.5	Outlines, destinations, link annotations	navigation

The WinAnsi encoding follows annex D of the specification. Font metrics follow the public AFM files distributed with the PDF reference. The implementation is original: no code, prose or comment is taken from another UDF; only the conventional AutoIt documentation layout is shared, as in every official include.

Version history

Version	What changed
1.1	Colour, opacity and dash guards; save lifecycle; UTF-16 metadata, UTC dates, content-derived /ID; CMYK JPEG through APP14; header and footer styling
1.2	DEFLATE compressor written in AutoIt; FlateDecode on every content stream; per-stream fallback for incompressible data
1.3	Hierarchical bookmarks, internal links, opening mode and page layout; the three validation tools in Tools
1.4	English edition of the UDF, examples and tests; this guide

References and licence (continued)

The package

Path	Contents
PDFX.au3	The engine: 42 public functions, no dependency, one #include
Examples	Four complete scripts and the PDFs they produce
Tests	Regression suites with reference outputs, one per subsystem
Tools	pdfcheck.py, outline.py and regress.py, standard library only
Guide	This document and the script that builds it

Licence

MIT licence or public domain, at your choice. Use PDFX in commercial software, modify it, redistribute it with or without this guide. Attribution is welcome, not required.

Author and support

Numeric1 — https://www.autoitscript.com/forum/. Questions, bug reports and example documents belong in the AutoIt forum thread; a minimal script that reproduces the problem is the most useful attachment. Version 1.4.0 is the reference for this guide.

A LAST WORD

The best way to learn the API is to take one of the four examples, change its data, and watch the PDF change with it. The engine is small enough to read end to end when a detail is unclear, and the tests describe the behaviour more precisely than any prose.