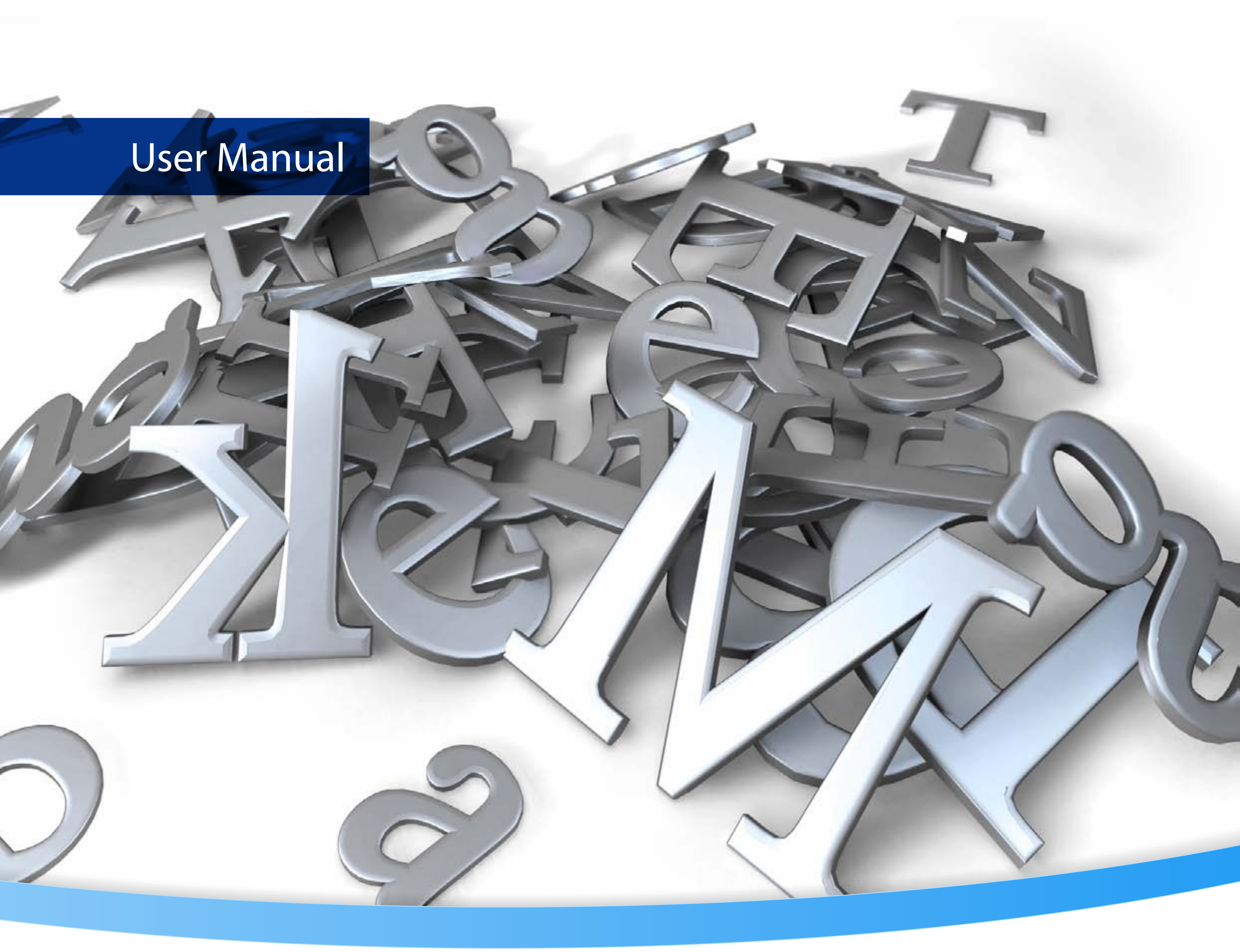


PDF Toolbox SDK

For .NET

Version 4.2.0

pdftools

Contents

1 Introduction

1.1 Description

The PDF Toolbox SDK is a component to create, extract, assemble, and modify PDF documents.

This product is the successor of the 3-Heights® PDF Toolbox API. For the task of migrating existing projects from 3-Heights® to 4-Heights®, PDF Tools offers an additional migration guide and a source code updating script. Please contact us for more information.

1.2 Functions

1.2.1 Features

Document assembly

- Copy pages from existing PDFs
- Copy annotations, form fields, links, logical structure, destinations, outlines, layers
- Flatten annotations, form fields, signatures
- Optimize resources
- Crop and rotate pages
- Compose content: overlays, underlays, stamps, transformations
- Add encryption: user password, owner password, permissions
- Copy and modify document metadata
- Copy and modify page metadata
- Add embedded files and associated files
- Get and set OpenAction destination
- Merge a PDF and an FDF
- Separate markup annotations into an FDF

Generation

Document level

- Create pages
- Create form fields
  - General text fields and comb text fields
  - Check boxes
  - Radio button groups
  - List boxes
  - Combo boxes
- Create new outline items and insert them at any position in the tree
- Destinations: Named and direct destinations in the same document
- Configure viewer settings

Page content level

- Create new PDF content from scratch
- Apply content to existing pages

Colors

- Device colors: RGB, CMYK, and grayscale
- ICC color profiles
- Transparency: alpha and blend mode

Paths

- Single and multi-segment lines
- Rectangle, circle, Bezier curves, ellipse, arc, pie
- Filling, stroking, clipping, and combinations thereof
- Line width, cap, join, dash array, dash phase, and miter limit
- Inside rule: nonzero winding rule, even/odd rule

Text

- Font size, character spacing, word spacing
- Horizontal scaling, leading, rise
- Enables simple text layouting
- Standard PDF fonts, installed fonts
- Font metrics: italic angle, ascent, descent, cap height, character width
- Unicode characters
- Text stroke: line width, line join, and dashes
- Fill and stroke text, invisible text
- Use text as clipping path

Images

- Bi-level: CCITT G3, G3 2D and G4, Flate, LZW, Packbits, uncompressed
- 4 bit and 8 bit grayscale: Flate, LZW, Packbits, JPEG and JPEG-6 (8 bit only), uncompressed
- RGB: Flate, JPEG and JPEG-6, LZW, Packbits, uncompressed

Transformations

- Translation
- Scaling
- Skewing (horizontal, vertical)
- Rotation

Annotations and links

- Document-internal links
- Web links
- Links to embedded PDFs
- File attachment annotations
- Free text annotation

- Sticky note annotation
- Text stamp annotation
- Custom stamp annotation
- Circle annotation
- Square annotation
- Line annotation
- Poly line annotation
- Polygon annotation
- Ink annotation (pen drawing)
- Highlight annotation
- Underline and squiggly underline annotation
- Strike-through annotation

Modification

Page content

- Selective deletion of content elements (without tagging and layers)
- Geometric transformation of content elements (without tagging and layers)

Annotations

- Web link annotation target URIs
- Markup annotation location, creation/modification date, subject, author, content

Form fields

- Deletion of fields and modification of field values for
  - General text fields and comb text fields
  - Check boxes
  - Radio button groups
  - List boxes
  - Combo boxes

Extraction

Document and page

- Document information entries: title, author, subject, keywords, creator, producer, creation date, modification date
- Document XMP metadata
- Document encryption settings
- Embedded files
- Page bounding boxes: media box, crop box, bleed box, trim box, art box
- Page XMP metadata
- Outline item tree: Tree structure, item title, expanded/collapsed
- Destinations: Named and direct destinations in the same document
- Viewer settings

Content

Page and group content elements, including:

- Bounding box
- Affine transformation

As either of the following:

- Group element
- Image element
  - Width and height in pixel
  - Bits per component
  - Color space
- Image mask element
  - Width and height in pixel
  - Paint for filling the mask
- Path element
  - Alignment box
  - Subpaths and subpath segments
  - Fill parameters including paint and fill rule
  - Stroke parameters including line paint and line style
- Shading element
- Text element
  - Text fragments
    - Bounding box
    - Affine transformation
    - Unicode string
    - Fill parameters, including paint and fill rule
    - Stroke parameters, including line paint and line style
    - Font size, character spacing, word spacing, horizontal scaling, and text rise

Annotations

- Annotations: location
- Markup annotation: type, location, creation/modification date, subject, author, content
- Custom stamp annotations: appearance
- Text markup annotations: markup area
- Link annotations: location, target destination or URI, active link area
- Signature fields: name, location, reason, contact info, date, visibility

AcroForm form fields

- Form field identifiers, export names and user names, including form field hierarchy
- Form field export and display content of:
  - Push buttons
  - Check boxes
  - Radio button groups
  - General text fields and comb text fields
  - List boxes
  - Combo boxes

1.2.2 Formats

Supported PDF formats

- PDF 1.x (PDF 1.0, …, PDF 1.7)
- PDF 2.0
- PDF/A-1, PDF/A-2, PDF/A-3
- FDF

Supported image formats

- BMP
- DIB
- JPEG
- JPEG2000
- JBIG2
- PNG
- GIF
- TIFF

Supported font formats

- Type1
- TrueType
- OpenType
- OpenType (CFF)

1.2.3 Conformance

Standards:

- ISO 32000-1 (PDF 1.7)
- ISO 32000-2 (PDF 2.0)
- ISO 19005-1 (PDF/A-1)
- ISO 19005-2 (PDF/A-2)
- ISO 19005-3 (PDF/A-3)

1.3 Interfaces

The following interfaces are available:

- C
- Java
- .NET Framework
- .NET Core[1]

[1] Limited supported OS versions. Operating systems

1.4 Operating systems

The PDF Toolbox SDK is available for the following operating systems:

- Windows Client 7+ | x86 and x64
- Windows Server 2008, 2008 R2, 2012, 2012 R2, 2016, 2019 | x86 and x64
- Linux:
  - Red Hat, CentOS, Oracle Linux 8+ | x64
  - Fedora 29+ | x64
  - Debian 10+ | x64
  - Other: Linux kernel 2.6+, GCC toolset 4.8+, glibc 2.27+ | x64
- macOS 10.10+ | x64 and arm64

'+' indicates the minimum supported version.

2 Installation

2.1 NuGet package

NuGet is a package manager that facilitates the integration of libraries for the software development in .NET. The NuGet package for the PDF Toolbox SDK contains all the libraries needed, managed and native, for all supported operating systems.

Installation

You should get the NuGet package `PdfTools.PdfToolbox.4.2.0.nupkg` directly from https://www.nuget.org/. In Microsoft Visual Studio, this is the default location for downloading packages.

Development The `PdfTools.PdfToolbox.4.2.0.nupkg` package contains .NET libraries with version .NET Standard 2.0, and native libraries for Windows, macOS and Linux.

The required native libraries are loaded automatically. All project platforms are supported, including “AnyCPU”.

In order to use the software, you should download a license key from your account on https://www.pdf-tools.com/ and use it as described in License keys.

> **Note:** This NuGet package is only supported on a subset of the operating systems supported by .NET Core. See also Operating systems.

2.2 Color profiles

If no color profiles are available, default profiles for both RGB and CMYK are generated on the fly by the PDF Toolbox SDK.

2.2.1 Default color profiles

If no particular color profiles are set, default profiles are used. For device RGB colors, a color profile named `"sRGB Color Space Profile.icm"` and for device CMYK, a profile named `"USWebCoatedSWOP.icc"` are searched for in the following directories:

Windows

1. `%SystemRoot%\System32\spool\drivers\color`
2. directory `Icc`, which must be a direct sub-directory of where the `PdfToolboxAPI.dll` resides.

Linux and macOS

1. `$PDF_ICC_PATH` if the environment variable is defined
2. the current working directory

2.2.2 Get other color profiles

Most systems have pre-installed color profiles available. For example, on Windows at `%SystemRoot%\system32\spool\drivers\color\`. Color profiles can also be downloaded from the links provided in the directory `bin\Icc\` or from the following websites:

- `http://www.pdf-tools.com/public/downloads/resources/colorprofiles.zip`
- `http://www.color.org/srgbprofiles.html`
- `https://www.adobe.com/support/downloads/iccprofiles/iccprofiles_win.html`

2.3 Fonts

When text is created by the PDF Toolbox SDK, all fonts from the Font directories can be used.

2.3.1 Font installation

On Windows, when a font is installed, it is by default installed only for a particular user. It is important to either install fonts for all users, or make sure the PDF Toolbox SDK is run under that user and the user profile is loaded.

2.3.2 Font cache

A cache of all fonts in all Font directories is created. If fonts are added or removed from the font directories, the cache is updated automatically.

To achieve optimal performance, make sure that the cache directory is writable for the PDF Toolbox SDK. Otherwise, the font cache cannot be updated and the font directories have to be scanned on each program startup.

The font cache is created in the subdirectory `<CacheDirectory>/Installed Fonts` of the Cache directory.

2.4 Special directories

2.4.1 Directory for temporary files

This directory for temporary files is used for data specific to one instance of a program. The data is not shared between different invocations and deleted after termination of the program.

The directory is determined as follows. The product checks for the existence of environment variables in the following order and uses the first path found:

Windows

1. The path specified by the `%TMP%` environment variable.
2. The path specified by the `%TEMP%` environment variable.
3. The path specified by the `%USERPROFILE%` environment variable.
4. The Windows directory.

Linux and macOS

1. The path specified by the `$PDFTMPDIR` environment variable.
2. The path specified by the `$TMP` environment variable.
3. The `/tmp` directory.

2.4.2 Cache directory

The cache directory is used for data that is persisted and shared between different invocations of a program. The actual caches are created in subdirectories. The content of this directory can safely be deleted to clean all caches.

This directory should be writable by the application. Otherwise, caches cannot be created or updated and performance degrades significantly.

Windows

- If the user has a profile:
  `%LOCAL_APPDATA%\PDF Tools AG\Caches`
- If the user has no profile:
  `<TempDirectory>\PDF Tools AG\Caches`

Linux and macOS

- If the user has a home directory:
  `~/.pdf-tools/Caches`
- If the user has no home directory:
  `<TempDirectory>/pdf-tools/Caches`

where `<TempDirectory>` refers to the Directory for temporary files.

2.4.3 Font directories

The location of the font directories depends on the operating system. Font directories are traversed recursively in the order as specified below.

If two fonts with the same name are found, the latter one takes precedence, i.e. user fonts always take precedence over system fonts.

Windows

1. `%SystemRoot%\Fonts`
2. User fonts listed in the registry key `\HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Fonts`. This includes user-specific fonts from `C:\Users\<user>\AppData\Local\Microsoft\Windows\Fonts` and app specific fonts from `C:\Program Files\WindowsApps`
3. directory `Fonts`, which must be a direct subdirectory of where `PdfToolboxAPI.dll` resides.

macOS

1. `/System/Library/Fonts`
2. `/Library/Fonts`

Linux

1. `/usr/share/fonts`
2. `/usr/local/share/fonts`
3. `~/.fonts`
4. `$PDFFONTDIR` or `/usr/lib/X11/fonts/Type1`

2.5 License keys

This key must be set programmatically by using the `Sdk.Initialize` method prior to making any calls to the library. You can download your license key from your account on https://www.pdf-tools.com/.

For licensing questions, contact pdfsales@pdf-tools.com.

3 User guide

3.1 General concepts

3.1.1 Document model

The document model of the PDF Toolbox SDK consists of two different types of objects:

Structure objects define the structure of the document, such as `Document`, `Page` or `Content`.

Graphics resources can be used to draw content with a `ContentGenerator`. Examples are `Image`, `Font` or `ColorSpace`.

All objects in the document model are bound to a specific document. They can only be used in the context of the document for which they were created using their `Create` or `Copy` method.

The objects of the document model are all stateless. Where a stateful interface is useful, it is provided by an external generator or extractor, which is not considered part of the document model.

3.1.2 Copying instead of modification

The PDF Toolbox SDK does not allow in-place modification of documents. Instead, the content is copied into a new document, while performing the necessary changes.

To copy objects from a source document into a target document, the object's static `Copy` method is called with the **target** document as first argument.

This concept allows the processing of very large files without consuming much memory: The content of the input document is only read on demand and any modifications can be directly stored in the output file.

3.1.3 Differentiation between object creation and use

To provide a uniform interface, many operations are divided into two steps:

1. Create (or copy) the object
2. Use the object

This separation allows to provide multiple variants for both steps, without having a "combinatorical explosion" of methods.

Step 1: Create

The object is created in the target document or copied from the source document to the target document.

After creating, the object is associated with the document, but not yet used. This means that copying or creating an object may change the size of the target file. However, logically, the PDF is still unchanged.

Examples are the following methods:

- `Page.Create`
- `Font.Create`
- `Page.Copy`
- `PageList.Copy`
- `ColorSpace.Copy`

- `Metadata.Copy`
- `ContentElement.Copy`

Step 2: Use

The associated object can then be used in the target document.

This second step is often more lightweight than the first step, since all the necessary copying is already done.

Examples are the following methods of a `ContentGenerator`:

- `PaintImage`
- `PaintGroup`
- `AppendContentElement`

or the `PageList.Add` method.

3.1.4 Generator objects

Some objects in a PDF consist of a list or stream of operations that operate on an internal state:

- Content streams
- Text objects
- Path objects

Since all data objects in the PDF Toolbox SDK are stateless, a (simplified) stateful interface is provided by the generator interfaces:

- `Content` objects can be modified with a `ContentGenerator`.
- `Path` objects can be modified with a `PathGenerator`.
- `Text` objects can be modified with a `TextGenerator`.

Generator objects must always be closed explicitly, before the generated object can be used.

3.1.5 Garbage collection and closing objects

Every interface object is considered being a resource that needs to be closed after use. Most objects are closed automatically, at the latest when the owning document is closed, in C# and Java, possibly earlier by the garbage collector.

In addition to `Document` objects, the "generator" objects `ContentGenerator`, `PathGenerator`, and `TextGenerator` must be closed, lest the generated objects are incomplete.

3.2 Thread safety

The PDF Toolbox SDK is generally thread-safe with one exception:

A document may only be accessed in one thread concurrently, including all sub-objects.

Almost all objects are directly or indirectly associated with a document.

Note: Methods that copy from a source to a target document have to access both documents. The thread safety rules not only apply to the target document, but also to the source document.
This means that concurrently copying from the same source document is not allowed.

3.2.1 Garbage collection and finalizer

Object finalization is thread-safe with one exception:

The finalizer of the `Document` is not thread-safe regarding access to its subobjects.

Subobjects do not retain their associated document object. If all references to an open document go out of scope, the document finalizer is eventually run and the document is closed.

Explicitly accessing (even closing) any subobject while the document finalizer is running is **not** safe!

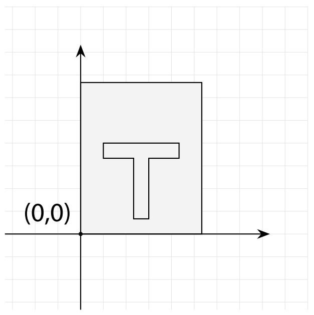
3.3 The PDF graphics model

3.3.1 Coordinate system

PDF coordinates are measured from bottom to top, unlike many other coordinate systems used in IT.

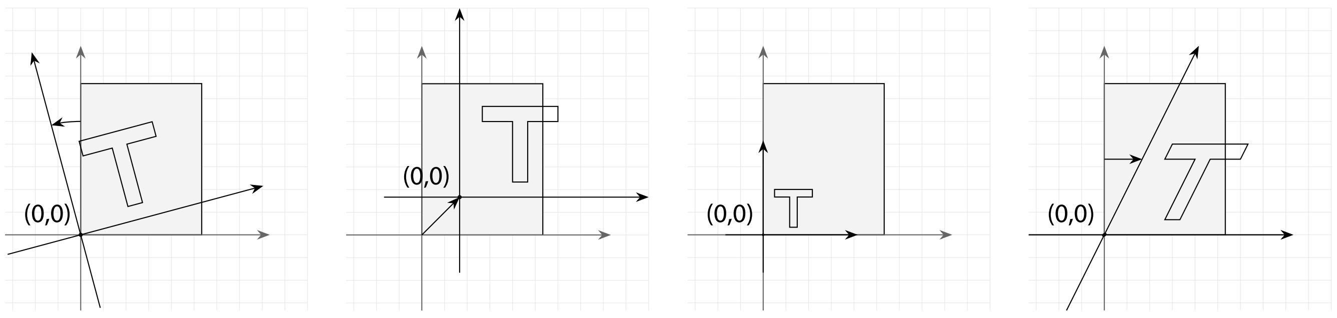
For the sake of simplicity, all coordinates used in the PDF Toolbox SDK are normalized such that the point (0,0) denotes the lower left corner of the visible page (crop box).

The internal **Rotate** attribute of a PDF page is not exposed at the API. Instead, all coordinates are assumed to refer to the already rotated page.



3.3.2 Affine transformations

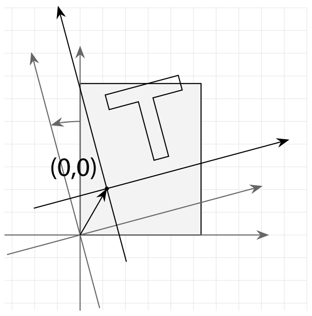
Affine transformations can be used to rotate, move, scale, or otherwise skew any page content.



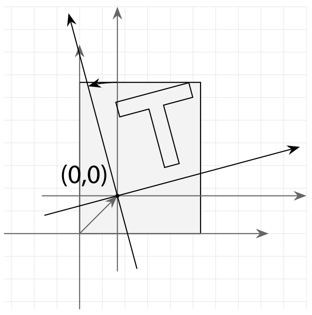
Rotate Move Scale Skew

Transformations always affect the coordinate system as a whole. All **following** graphics operations are executed in the transformed coordinate system, including additional transformations.

This means that the ordering of how transformations are applied is important.

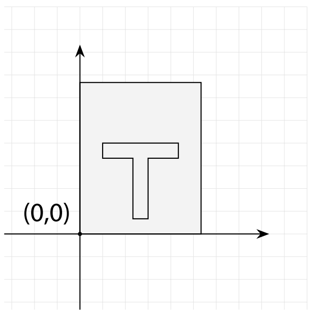


Rotate, then move

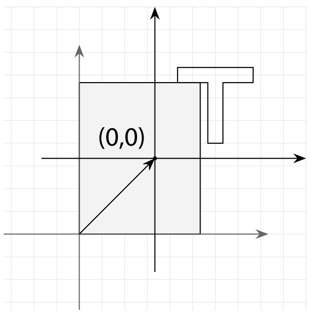


Move, then rotate

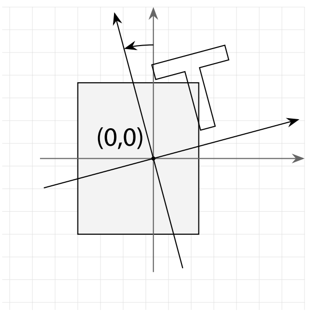
Example: Rotate around a certain point



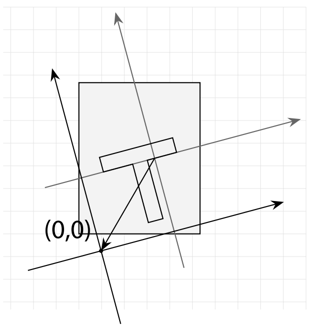
Start ..



.. Move to center point ..



.. Rotate ..



.. Move back to origin

3.4 Annotations and form fields

Annotations are elements that are not part of a page's content, but are applied on top of a page. In contrast to ordinary page content, many annotation types are meant to behave interactively in a PDF viewer.

3.4.1 Form fields

AcroForm form fields in a PDF consist of data structures that represent variable values, potentially to be modified by a user in a PDF viewing application.

Form fields are structured in a tree topology in which the `Document` acts as the tree root. The immediate child form field nodes thereof can be accessed via the `Document.FormFields` property.

Each child form field node in the tree can itself have more children. Such form field is called a "sub form". A child form field node that has no children is simply a "form field", of which the following concrete sub-types exist:

- General text field
- Comb text field
- Check box field
- Push button field
- Radio button field
- Combo box field
- List box field

In the PDF Toolbox SDK, the above types are modeled by classes that inherit from a base class `FieldNode` in the following way:

```
FieldNode
 ├─ Field
 │   ├─ CheckBox
 │   ├─ ChoiceField
 │   │   ├─ ComboBox
 │   │   └─ ListBox
 │   ├─ PushButton
 │   ├─ RadioButtonGroup
 │   └─ TextField
 │       ├─ CombTextField
 │       └─ GeneralTextField
 └─ SubForm
```

`FieldNode`s are contained in a `FieldNodeMap` in the form of key-value pairs, for which the keys act as form field node **identifiers**. The identifier must not contain any full stops (“.”).

The **fully qualified identifier** of a form field node is defined as the concatenation of all its ancestor `SubForm`’s identifiers and its own identifier, separated by full stops (“.”), e.g. `MySubForm.MyField`. The fully qualified identifier of each form field node is unique within a document.

While the form field tree models the form’s data, the visual manifestations of form fields are managed by `Widget`s contained in `Page.Widgets`, of which each form field has at least one.

Creating form fields

In an output document (a document created with `Document.Create`), form fields can be created from scratch by means of the following methods:

- `CheckBox.Create`
- `ComboBox.Create`
- `CombTextField.Create`
- `GeneralTextField.Create`
- `ListBox.Create`
- `RadioButtonGroup.Create`
- `SubForm.Create`

A `PushButton` cannot be created.

After creating a `ChoiceField`, i.e. a `ComboBox` or a `ListBox`, `ChoiceItem`s should be created and added with the `ChoiceField.AddNewItem` method.

After creating a `RadioButtonGroup`, new `RadioButton`s should be created and added with the `RadioButtonGroup.AddNewButton` method.

Each created form field must be added either to the document’s `Document.FormFields`, or to the `SubForm.Children` of a `SubForm`.

You should set all form field properties prior to creating any `Widget`s. Specifically, changing form field properties that affect the form field’s visual appearance fails with an `UnsupportedOperation` error if the form field has widgets.

For each form field, at least one `Widget` should be created using the `RadioButton.AddNewWidget` method for radio button groups, or the form field’s `Field.AddNewWidget` method for all other field types.

Finally, each created widget must be added to one of the `Page.Widgets` of any of the document’s `Page`s.

A page can either be created from scratch with the `Page.Create` method or it can be copied with `Page.Copy` from an input document (a document created with `Document.Open`). In the latter case, `PageCopyOptions.FormFields` must not be set to `FormFieldCopyStrategy.Copy` and `PageCopyOptions.UnsignedSignatures` not to `CopyStrategy.Copy`.

The combination of creating form fields and copying form fields or unsigned signatures via `Page.Copy` or `PageList.Copy` (with `PageCopyOption` argument in which either the `PageCopyOptions.FormFields` property is set to `FormFieldCopyStrategy.Copy` or the `PageCopyOptions.UnsignedSignatures` property is set to `CopyStrategy.Copy`) is not supported. Specifically:

- Once `Page.Copy` or `PageList.Copy` has been called with `PageCopyOptions.FormFields` set to `FormFieldCopyStrategy.Copy` or `PageCopyOptions.UnsignedSignatures` set to `CopyStrategy.Copy`, any subsequent call to any of the form field creation methods fails with an `IllegalState` error.
- Once any of the form field creation methods has been called, any subsequent call to `Page.Copy` or `PageList.Copy` with `PageCopyOptions.FormFields` set to `FormFieldCopyStrategy.Copy` or `PageCopyOptions.UnsignedSignatures` set to `CopyStrategy.Copy` fails with an `IllegalArgument` error.

Filling form fields

Filling a form means that the values (field content) of form fields are modified. Depending on the field type, this implies the following:

- `TextField`: modify text using `TextField.Text`.
- `CheckBox`: modify the state using `CheckBox.Checked`.
- `RadioButtonGroup`: modify the choice using `RadioButtonGroup.ChosenButton`.
- `ComboBox`: modify choice using `ComboBox.ChosenItem` or `ComboBox.EditableItemName`.
- `ListBox`: modify choice using `ListBox.ChosenItems`.

To use the PDF Toolbox SDK for filling out the values of form fields in a PDF, the following procedure must be followed:

1. An input document is opened with `Document.Open` and an output document is created with `Document.Create`.
2. Before copying pages, the form fields must be copied from the input document to the output document as follows:
   a. Access the form field node map of the input and the output document via `Document.FormFields`.
   b. Copy each form field node found in the input to the output document using the `FieldNode.Copy` method. Copying `SubForm`s automatically copies their children. (Note that the copied form fields have no widgets yet.)
   c. The value of a copied form field can be modified here or later in Step 3.
   d. Add each copied form field node to the output document's field node map `Document.FormFields`, preferably using the same key as used in the input document's field node map.
3. The output document's form field nodes can be accessed, e.g. using `FieldNodeMap.Lookup` to modify form field values.
4. Copy all pages with the `PageList.Copy` method. Hereby, the `PageCopyOptions.FormFields` property in the `PageCopyOptions` argument must be set to `FormFieldCopyStrategy.CopyAndUpdateWidgets` and the `PageCopyOptions.UnsignedSignatures` must be set to `CopyStrategy.Remove` or `CopyStrategy.Flatten`. In this step, the `Widgets` of input form fields are copied to the output form fields and automatically updated to reflect the new form field values. (As soon as a form field has Widgets, its value can no longer be modified.)

The combination of filling form fields and copying form fields via `Page.Copy` or `PageList.Copy` (with `PageCopyOptions` argument in which `CopyOptions.FormFields` is set to `FormFieldCopyStrategy.Copy`) is not supported. Specifically:

- Once `Page.Copy` or `PageList.Copy` has been called with `PageCopyOptions.FormFields` set to `FormFieldCopyStrategy.Copy`, any subsequent call to `FieldNode.Copy` fails with an `IllegalState` error.
- Once `FieldNode.Copy` has been called, any subsequent call to `Page.Copy` or `PageList.Copy` with `PageCopyOptions.FormFields` set to `FormFieldCopyStrategy.Copy` or `PageCopyOptions.UnsignedSignatures` set to `CopyStrategy.Copy` fails with an `IllegalArgument` error.

4 .NET interface

4.1 IDisposable objects

Objects that must be closed explicitly implement the `IDisposable` interface. Instead of calling `Dispose()` directly, it is recommended that you use the "`using`" statement:

```
using (Document document = ...)
{
  ...
} // document.Dispose() is called implicitly here
```

See also Garbage collection and closing objects.

4.2 Error handling

Errors are reported using exceptions.

The following logic errors are mapped to the corresponding native exception classes:

`IllegalArgument` maps to `System.ArgumentException`

`IllegalState` maps to `System.InvalidOperationException`

`UnsupportedOperation` maps to `System.NotSupportedException`

Additionally, the following infrastructure error is mapped:

`IO` maps to `System.IO.IOException`

The remaining errors are modeled using exception classes that inherit from the class `PdfToolboxException`.

4.3 Streams

The native stream interface `System.IO.Stream` is used.

4.4 Lists

Lists implement the native list interface `System.Collections.Generic.IList<T>`.

4.5 Enumerables

Enumerables implement the native interface `System.Collection.Generic.IEnumerable<T>`.

4.6 Maps

Maps implement the native dictionary interface `System.Collections.Generic.IDictionary<K, V>`.

5 Version history

5.1 Changes in Version 4

Changes in Version 4.2

Pdf.Annotations.TextStamp

- **Deprecated** static method `createRaw`.
- **New** static method `create`.

Pdf.Content.TextFragment

- **New** method `remove`.

Pdf

- **Changed** the following classes to be abstract:
  - `Pdf.Content.ColorSpace`
  - `Pdf.Content.ContentElement`
  - `Pdf.Forms.ChoiceField`
  - `Pdf.Forms.Field`
  - `Pdf.Forms.FieldNode`
  - `Pdf.Forms.TextField`
  - `Pdf.Navigation.Destination`
  - `Pdf.Navigation.DirectDestination`
  - `Pdf.Navigation.Link`

Changes in Version 4.0

Sdk

- **Changed** behaviour of static property `ProducerFullName`. The producer full name is returned instead of the product name.

Pdf.Content

- **New** enumeration `FontWeight`.

Pdf.Content.Font

- **New** property (get) `Weight`.

Pdf.Content.Glyph

- **New** property (get) `Width`.

Pdf.Document

- **Changed** behaviour of static methods `Create` and `CreateWithFdf`. A `ConformanceException` is generated if the conformance level is lower than 1.7 and Unicode passwords are specified.

Pdf.Permission

- **New** enumeration items `All` and `None`.

5.2 Changes in Version 3

Changes in Version 3.10

Pdf.Content

- **New** class `Subpath`.
- **New** struct `PathSegment`.
- **New** enumeration `PathSegmentType`.

Pdf.Content.Path

- **New** extension of interface: Now extracted `Path`s are an iterable for contained `Subpath`s.

  New static method `CreateWithFdf` to store markup annotations in a separate FDF document.

Changes in Version 3.8

Pdf.Content.ContentGenerator

- **Removed** errors `IO` and `Corrupt` from `ContentGenerator` constructor.

Changes in Version 3.6

Pdf.Annotations.AnnotationList

- **Changed** methods `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.Annotations.MarkupInfoList

- **Changed** methods `Add`, `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.Content.Text

- **Changed** method `Add`: Since this method is not supported it is now implemented explicitly.

Pdf.Content.TextFragment

- **Changed** methods `Add`, `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.Forms.FieldNodeMap

- **Changed** methods `Clear` and `Remove`: Since these methods are not supported they are now implemented explicitly.

Pdf.Forms.RadioButtonList

- **Changed** methods `Add`, `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.Forms.SignatureFieldList

- **Changed** methods `Add`, `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.Forms.WidgetList

- **Changed** methods `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.Navigation.LinkList

- **Changed** methods `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.FileReferenceList

- **Changed** methods `Clear`, `Remove` and `RemoveAt`: Since these methods are not supported they are now implemented explicitly.

Pdf.PageList

- **Changed** methods Clear, Remove and RemoveAt: Since these methods are not supported they are now implemented explicitly.

Changes in Version 3.4

Pdf.Content

- **New** enumeration WritingMode.
- **New** class Glyph.

Pdf.Content.TextFragment

- **New** extension of interface: Now this is an iterable for contained Glyphs.
- **New** property (get) Font

Changes in Version 3.3

Pdf.Navigation.WebLink

- **Changed** error behavior of property setter for Uri: An IllegalArgument error is generated if the given value is empty.
- **Changed** error behavior of methods Create and CreateFromQuadrilaterals: An IllegalArgument error is generated if the given uri argument is empty.

Pdf.PageCopyOptions

- **Deprecated** property OcgConflictResolution.

Pdf.Forms.SignatureField

- **New** derived classes DocumentSignature, DocMdpSignature, DocUrSignature and DocumentTimeStamp to extract properties of digital signatures.
- **Replaced** property IsSigned with derived class SignedSignatureField.
- **Moved** properties Name and Date to derived class SignedSignatureField, because they are only available in signed signature fields.
- **Moved** properties Location, Reason and ContactInfo to derived class Signature, because they are only available in certain types of signed signature fields.

Pdf.Forms

- **New** type MdpPermissions for the Permissions of DocMdpSignature.

Changes in Version 3.1

Pdf.Content

- **New** item `All` in enumeration `UngroupingSelection`.

5.3 Changes in Version 2

- **New** support for creating ranges of page lists.
- **New** support for copying page lists.
- **Changed** type of argument in `FileReference.Create` from `DateTimeOffset` to `DateTimeOffset?`.
- **Changed** type of argument in `TextGenerator.Create` from `Geometry.Real.Point?` to `Geometry.Real.Point`.
- **New** struct `Quadrilateral` and class `QuadrilateralList`.

StringMap

- **Re-implemented** as explicit interface methods of `ICollection<KeyValuePair<string, string>>`:
  - `Add(KeyValuePair<string, string> item)`
  - `Contains(KeyValuePair<string, string> item)`
  - `CopyTo(KeyValuePair<string, string>[] array, int arrayIndex)`
  - `Remove(KeyValuePair<string, string> item)`

Geometry.Real.AffineTransform

- **New** method `TranformRectangle`.
- **New** method `TransformQuadrilateral`.

Pdf.Content.Font

- **New** property (get) `Leading`.

Pdf.Content.TextFragment

- **New** property (get) `FontSize`.
- **New** property (get) `CharacterSpacing`.
- **New** property (get) `WordSpacing`.
- **New** property (get) `HorizontalScaling`.
- **New** property (get) `Rise`.

Pdf.Navigation.Link

- **New** property (get) `ActiveArea`.

Pdf.Navigation.InternalLink

- **New** static method `Create`.
- **New** static method `CreateFromQuadrilaterals`.

Pdf.Navigation.WebLink

- **Changed** error behavior of static method `Create`: Argument `string uri` is not allowed to be null.
- **New** static method `CreateFromQuadrilaterals`.

Pdf.Navigation.EmbeddedPdfLink

- **New** static method `CreateFromQuadrilaterals`.

Pdf.Annotations.TextMarkup

- **New** property (get) `MarkupArea`.

Pdf.Annotations.Highlight

- **New** static method `CreateFromQuadrilaterals`.

Pdf.Annotations.Underline

- **New** static method `CreateFromQuadrilaterals`.

Pdf.Annotations.StrikeThrough

- **New** static method `CreateFromQuadrilaterals`.

Pdf.Annotations.Squiggly

- **New** static method `CreateFromQuadrilaterals`.

Pdf.Forms.FieldNodeMap

- **Re-implemented** as explicit interface methods of `ICollection<KeyValuePair<string, FieldNode>>` methods:
  - `Add(KeyValuePair<string, FieldNode> item)`
  - `Contains(KeyValuePair<string, FieldNode> item)`
  - `CopyTo(KeyValuePair<string, FieldNode>[] array, int arrayIndex)`
  - `Remove(KeyValuePair<string, FieldNode> item)`

6 Licensing, copyright, and contact

Pdftools is a world leader in PDF (Portable Document Format) software, delivering reliable PDF products to international customers in all market segments.

Pdftools provides server-based software products designed specifically for developers, integrators, consultants, customizing specialists and IT-departments. Thousands of companies worldwide use our products directly and hundreds of thousands of users benefit from the technology indirectly via a global network of OEM partners. The tools can be easily embedded into application programs and are available for a multitude of operating system platforms.

Contact

PDF Tools AG
Brown-Boveri-Strasse 5
8050 Zürich
Switzerland
`http://www.pdf-tools.com`
`pdfsales@pdf-tools.com`