

# ABCpdf .NET Documentation

---

ABCpdf .NET lets you dynamically create Adobe® PDF documents on the fly. Because it doesn't use any print drivers and goes Direct to PDF, it's incredibly fast. WebSupergoo - sticking the web together.



© 2015 WebSupergoo

Every effort has been made to ensure that the information in this manual is accurate. However, WebSupergoo makes no guarantees about the accuracy or quality of this manual or any software it describes.

Apple®, Mac®, Macintosh®, QuickDraw®, and QuickTime® are trademarks of Apple Computer, Inc. Windows®, Windows NT®, Visual Basic® and Visual Studio® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Adobe®, Acrobat®, Adobe PDF®, PostScript® and Photoshop® are trademarks of Adobe Systems Incorporated or its subsidiaries. Macromedia®, Flash and Macromedia Flash are trademarks or registered trademarks of Macromedia, Inc. in the United States and/or other countries. The use of these trademarks does not imply any sponsorship or endorsement by the owners of these trademarks.

## What is ABCpdf .NET?

---



Use ABCpdf .NET to dynamically create Adobe PDF documents on the fly. You won't believe how simple - yet how powerful this tool truly is!

You can use ABCpdf .NET from languages like C# or Visual Basic .NET.

ABCpdf .NET runs on Windows 2000, Windows XP, Windows Server 2003, Windows Vista, and Windows Server 2008.

## Who is this document for?

---



This document is written as a guide to and reference material for ABCpdf .NET. It assumes basic knowledge of ASP.NET and either Visual Basic or C#. It also assumes you know how to set up and configure Microsoft's Internet Information Services - IIS.

## What do I need to use it?

---

You can run ABCpdf .NET on all Windows versions since Windows XP.

You need .NET 4.0. We recommend your host machine has Internet Explorer 9 or later installed.

If you're rendering 3D PDFs, you will need OpenGL version 3.1 compliant graphics drivers.



Finally, you'll need ABCpdf .NET itself.

What happened to ABCpdf ASP? Well, ABCpdf .NET includes a COM layer over the native .NET base - a layer which exactly mimics the interface of ABCpdf ASP. This means you can use one product for your .NET deployment and also for any legacy COM deployment. Think of it as double value - ABCpdf .NET and ABCpdf ASP both in one package. Plus, you get the advantages of the .NET architecture behind a COM front end - advantages like native x64 support and features like SVG import.

## How does it work?

---

ABCpdf .NET gets up close and personal! Because it doesn't use any print drivers and goes Direct to PDF™ it's incredibly fast.



Because it's fully multithreaded you can use it flexibly within any .NET environment from ASP.NET to COM+ to straight Windows applications..

In other products you should look out for external libraries. Threading make no difference at all if further down the line a third-party piece of software pipelines every request.

Because ABCpdf .NET doesn't rely on any other software it can be completely multithreaded without any unpleasant bottlenecks.

# What's Cool?

---

ABCpdf gets up close and personal! Because it goes Direct to PDF™ it's incredibly fast. Because ABCpdf doesn't rely on any other software it can be completely multithreaded without any unpleasant bottlenecks.

ABCpdf is simple yet powerful. It's designed so you can get up to speed and productive within ten minutes. Yet if you want fine low-level control you can have that too with ABCpdf.

Create PDF documents from scratch or read and modify existing PDF documents. Add pages from other documents for seamless joining, insertion and stitching of multiple documents. Stream your documents direct to your client web browser without going via the disk. All up into the GB range and beyond.

Render your PDF documents in a variety of formats. Output your pages in raster formats like JPEG, GIF, TIFF, PSD or JPEG 2000 in any of a variety of color spaces and bit depths. Alternatively choose vector formats like EMF, EPS, XPS, SVG and SWF (Flash) for specialist areas such as high resolution print work. Control advanced rendering settings such as alpha, compression type, multiple pages and different horizontal and vertical resolutions.

Add virtually any graphic into your PDFs. ABCpdf supports JPEG, GIF, TIFF, BMP, PNG, PSD, PSB, EXIF, WMF, EMF, JPEG 2000, PS, EPS, XPS, WPF, SVG and SWF (Flash) amongst others. Additionally you can reference image data from multiple locations

in your document - great for inserting watermarks and other frequently used graphics. ABCpdf is fully PostScript compatible.

Full support for the XML Paper Specification (XPS), Open XML Paper Specification (OXPS) and for Encapsulated PostScript (EPS). Microsoft XPS provides an alternative for page description and document storage but it is not compatible with the de-facto PDF standard. EPS is widely used in printing but often provides compatibility headaches when it comes to interoperability with PDF. ABCpdf supports full interoperability between Adobe PDF and Microsoft XPS and EPS. Convert your PDF documents to XPS or EPS. Convert your XPS or EPS documents to PDF. And our conversion routines are carefully written to preserve the natural structure of your source documents wherever possible. So this means the preservation of color spaces and the preservation or conversion of embedded fonts. Unless of course you want to change the format in which case you can render directly to Grayscale, RGB or CMYK EPS.

Output, validation and conversion to PDF/A standards. Conversion is made to work the way it should - it operates on practically all documents - even documents provided by third parties. A partner in this is transparency flattening which is a technology which allows you to remove transparency from your documents while leaving the vector nature of the format intact.

Full support for the Windows Presentation Foundation (WPF). Microsoft WPF is the new graphics subsystem developed for .NET 3.0. It comes in both the standard flavor seen on platforms like Windows Vista and also in a lightweight web based system - Silverlight. Of course with a new graphics API drawing to screen a new PDF export facility is needed too. As

such ABCpdf comes with full WPF import capability. See our WPFTable example project for details and examples.

Native support for Flash. Yes we wrote our own SWF rendering engine! That means that those lovely smooth graphs you see in Flash stay smooth and resolution independent when you import them into a PDF. And while we were about it we wrote our own Flash export engine so you can convert your PDF documents to native vector Flash.

Not only does ABCpdf support PDF, HTML and Rich Text Format (RTF) natively but it also supports a wide range of other document formats. Formats like Microsoft Word (.doc), Microsoft Excel (.xls), PowerPoint (.ppt), WordPerfect (.wpd), Lotus 1-2-3 (.wk1) and AutoCAD (.dxf). All you need to do is [read](#) them in! \*

ABCpdf allows you precise control over the way that your text is laid out. Paragraph indent, kerning and tracking, word spacing, line spacing, paragraph spacing and horizontal justification are just some of the settings available. ABCpdf supports synthesized bold and italic typeface styles for situations in which you do not want to reference multiple typefaces.

ABCpdf supports Unicode and foreign languages. Reference fonts from the relevant foreign language pack or embed Unicode fonts for guaranteed fidelity of reproduction on any platform. Draw text horizontally or vertically and quickly subset large CJK fonts with minimal use of memory and impact on server load. Work with bidirectional script such as Hebrew and contextual ligatures in languages such as Arabic

Import HTML / CSS pages from local or remote web sites. Placed HTML support means your HTML can

be treated just like any other media - placed wherever you like on the page. Paged HTML support allows you to flow your HTML from one area to another - across pages or columns or both. Select between the FireFox and Internet Explorer HTML rendering engines. A vast range of options gives you full control over the HTML rendering process and DOM.

HTML styled text allows easy creation and layout of multi-styled text; supporting text box chaining to allow you to easily and automatically flow text through from one area to another, around images and other irregular objects. More complex options like text on a curve or drop shadows are simple to implement using our example code.



Apply advanced transformations like rotation, magnification, skew and translation. This means you can draw rotated text, images and graphics all with only a few simple commands.

ABCpdf allows multiple different approaches to layout and document structure. The Table project shows how to use programmatic table based layout. The WPF Table project shows how to take WPF based content and convert it to PDF. The ABCpdf10.Drawing wrapper namespace exists for easy porting of System.Drawing code for PDF output. Or if you're more interested in XML to PDF or tagged PDF output there are the Tagged PDF Example projects.

ABCpdf supports sophisticated color spaces. Mix and match RGB, CMYK, Grayscale, Lab, calibrated color spaces and spot colors. Any kind of graphic from text to lines to blocks of color can be drawn in any of these color spaces. You can even perform complex operations like colorizing grayscale images using spot colors or converting from one color space to another.

ABCpdf supports direct import of ICC based RGB, CMYK and Lab images from TIFF, PSD and PSB. Full TIFF and PSD support for all color spaces, compression models and bit depths up to HDR (High-dynamic-range). This means you can produce high-quality, print-ready PDFs directly from your applications.

ABCpdf supports transparency everywhere. Any kind of graphic from text to lines to blocks of color can be drawn transparently using a simple alpha value to control levels of opacity. You can apply soft masks or chromakey style masks to images for selective transparency. Or you can draw transparent images such as GIF using simple calls.

Create encrypted or signed PDFs for secure storage of PDF documents. Apply user permissions and secure these permissions with encryption keys from 40 to 128 bits in size. Check signed documents for validity. Examine different revisions of documents to find changes between them. Revert back to any previously saved revision for comparison.

ABCpdf supports Fields and Forms. Use placeholder fields in your template documents to position and lay out dynamically created elements or assign field values directly. Stamp fields directly into the PDF to prevent them being modified. Create new annotations, anything from simple text fields through to multiple incrementally updated signatures. With advanced support for exotic types of field structure such as combs.

ABCpdf now supports conversion of PDF content into annotated SVG. This allows you to identify individual elements on a page and map them back to the

operators in the original PDF file. So it lets you perform operations like search and replace on PDF text or identify individual images on a page.

PDF optimization and size reduction provides whole document optimization using a variety of options to reduce font sizes, remove embedded fonts, to resample and recompress images, to flatten wherever possible for the smallest possible output.

PDF analysis has been extended into easy-to-use operations for text and images. Simple on the surface but sophisticated underneath, they allow you to extract common-sense, de-hyphenated and de-ligatured text from PDFs. Then select items of that text within the PDF and perform operations on those selections.

PDF accessibility is now something that can be supported, even for PDFs which were never designed to do so. Standards such as PDF/UA and Section 508 compliance require that PDFs be accessible. Our accessibility operation performs a sophisticated semantic analysis of the document content and adds appropriate tagging information to produce an accessible Tagged PDF output.

Our ABCpdfView sample application now demonstrates this and a whole host of other ABCpdf features. It allows you to open, view and print PDF documents. It lets you edit text and text styles within the PDF. It allows you to insert, delete and re-order pages. It allows you to watermark documents. PDFView comes with full source code.

ABCpdf offers great control over images in PDFs. You can resize them using a variety of sophisticated and high quality resampling algorithms. You can assign new color spaces or convert them from one color

space to another. You can compress them in CCITT, JPEG, Flate or JPEG 2000 format. You can resample them from one bit depth to another. Determine size, resolution and placements using sophisticated analysis operations.

ABCpdf is fully floating point for precise positioning of text, line and other objects at a fractional point level.

ABCpdf allows even more advanced control over any PDF object in your document. If you can't see how to accomplish a task using our simple to use methods then you can always access the raw PDF structure directly. This is true both for the structure of the document itself and also for any content streams held within that structure.

If you think there's something missing please do mail us.

\* Requires helper applications such as WordGlue .NET, Microsoft Office or OpenOffice.org to be installed. WordGlue .NET is available on our site. OpenOffice.org is freely distributed under the [GNU Lesser General Public License \(LGPL\)](#). For full details of the OpenOffice.org project see <http://www.openoffice.org/>.



# What's New?

---



ABCpdf now contains a host of image effects for common operations like [sharpen](#), [auto-levels](#), [contrast](#) and [color adjustments](#); morphs such as [twirl](#) and [wave](#). See the [effects section of the documentation](#) for full details.

Our new [Photoshop read module](#) supports the standard PSD and also the large image PDB file types. It allows the direct import of RGB, Grayscale, CMYK, Lab, Indexed and Duotone images in 1, 8, 16 and 32 bits per component color depth. The PDF format does not support 32 bits per component HDR images so these are scaled down to 16 bits per component. It also supports the import of different layers if you need to extract these individually.

The new [TIFF import module](#) is much more efficient and reliable than the old one. In particular it has been tested extensively with unusual multi-page TIFF formats containing unusual compression schemes. It is a fully comprehensive module which will cope brilliantly with any TIFF you throw at it. It is difficult to describe reliability as a feature but most solutions will not cope with unusual TIFFs and this reliability is invaluable. In terms of more traditional new features, TIFF images may contain rotation flags to indicate orientation. In the past ABCpdf would ignore many of these flags. The effect was that images such as scans, which should have been one way up, may have appeared upside down. ABCpdf now supports

all these flags so your tagged images will automatically appear in an appropriate orientation when they are imported by ABCpdf.

The new [RTF import module](#) is a native Rich Text Format import module which allows RTF to be imported quickly and easily without the need for any helper applications.

PDF reading now supports Portfolios (also known as PDF Collections), controllable using the new [XReadOptions.OpenPortfolios](#) property. This property allows you to select between treating the Portfolio as one document or as a set of documents. The former is what a user sees when they open a Portfolio using Acrobat, so it is what you will want most of the time. The latter is what is appropriate if you want to manipulate the contents of the PDF Portfolio using methods like [Catalog.GetEmbeddedFiles](#).

As well as manipulating PDF Portfolios you can also create them using the [example code](#) that goes with the new [EmbeddedFile](#) and [FileSpecification](#) classes available in the new release.

---

## Rendering

The PDF format has long supported 3D objects as well as standard 2D ones. We believe we are the first in the world outside of Adobe to support rendering of these 3D elements. At present ABCpdf Version 10 supports U3D (Universal 3D) elements but we expect to extend this to include the PRC (Product Representation Compact) 3D format and also to the live manipulation of these 3D models.

PDF supports a variety of color spaces and past versions of ABCpdf have been very good at allowing you to handle RGB, Grayscale and CMYK colors. Spot colors have been implemented using a variant of grayscale but other color spaces such as Lab, Calibrated RGB, Calibrated Grayscale, Multichannel and Pattern have been impossible to represent using the standard ABCpdf objects.

In ABCpdf Version 10 the [XColor](#) object has been extended to allow these types of colors to be represented. An XColor can be any of the more normal color spaces or it can be a generic, unconstrained set of color components defined in the context of the current color space. The new [XColor.ColorSpace](#) enumeration provides detail of the type of color space to which the XColor belongs, the new [XColor.Components](#) collection allows you access to the raw PDF color and the [XColor.Name](#) property provides access to any pattern name.

---

## Colors

This new structure brings benefits in terms of generating new PDFs but also in terms of analyzing and modifying the content of existing PDF documents. In terms of creation, there are new examples showing how to generate PDF documents using the [Calibrated RGB](#), [Calibrated Grayscale](#) and [Lab](#) color spaces. In terms of analysis it is easy to create a color from a set of parameters in a content stream using the [XColor.FromContentStream](#) method.

Text functionality is much improved and contains a number of new and frequently requested features.

ABCpdf now supports proper [kerning](#). It reads the kerning tables from TrueType fonts and automatically adjusts the spacing between letters to make them look just right. You don't need to do anything to take advantage of this - just recompile your code against the new namespace and it will all just happen.

ABCpdf now fully supports bidirectional, contextual ligatures for languages such as Arabic. These can be enabled using the [TextStyle.Direction](#) property.

Another frequently requested feature was the ability to flow text around shapes such as images. Now we make it easy to [create a variable shaped area](#) into which your text can be inserted.

Use HTML styled text to create tables of contents with [leaders](#). Leaders are the dotted lines you see in tables of contents, between the heading on the left and the page number on the right. Previously creating these was complex and error prone. Now it's just one tag.

---

#### Text

There are a variety of new text measurement and positioning options. The [Doc.FitText](#) and [Doc.FitHtml](#) methods allow you to add text scaled to fit a particular area on the page. The [Doc.MeasureText](#) method allows you to measure the length of a string of text without adding it to the document. Or if you want a more fine grained approach you can use the [TextLayer.TextFragments](#) property to find the individual locations and styles of each of the text items that have been added.

The `FontObject` has a new [EmbedFont](#) method to allow a font to be embedded or re-embedded into an existing PDF document. It also has new and useful properties such as [Flags](#), [FontBBox](#), [FontAscender](#), [FontDescender](#), [FontAscent](#), [FontDescent](#),

[FontLineSpacing](#) and [FontLineGap](#). This allows new and useful example code such as [drawing text on a curve](#).

In previous version of ABCpdf, Fields and Annotations were largely static. You could change their value but not much else. In this release they become much more dynamic allowing interactive access to locations and styles.

You can now set the [Field.Rect](#) or [Annotation.Rect](#) to move them around on the page. You can set the [Field.Page](#) or [Annotation.Page](#) to move them between pages. For Annotations you can get and set the [Border](#), [FieldBackgroundColor](#), [FieldBorderColor](#) and [FieldRotation](#). For Fields you can now get and set the [TextAlignment](#), [TextFont](#), [TextSize](#) and [TextColor](#). For more complicated styles you can directly access the [DefaultAppearance](#) which provides control over all other field styles.

---

Forms &  
Fields

Fields and Annotations are intimately linked but in the past ABCpdf has represented them as somewhat separate. In the new release these objects have been rationalized to make them coordinate better. There is a [Field.GetAnnotations](#) method to find all the Annotations referenced by a field. There are new [Annotation.Stamp](#) and [Annotation.Focus](#) methods to mimic those offered by the Field class. There are [Annotation.Flags](#) and [Field.Flags](#) properties to allow common flags such as Hidden and Print to be queried, set or cleared.

New features allow the simple and efficient analysis and deconstruction of PDF documents. Disassemble, modify, reassemble. Used to be complex. Now it's easy.

At a low level, a range of new Atom operations have been introduced for fast and robust handling of PDF content streams. At the core of these changes are the new [OpAtom](#) class and the [ArrayAtom.FromContentStream](#) static constructor. The former represents a PDF operator in a drawing stream and the latter allows a content stream to be deconstructed into an ArrayAtom.

Once the stream has been deconstructed in this way you can use the [OpAtom.Find](#) method for a fast and memory efficient way of selecting various operators out of the array. The parameters for the operators can be established using the [OpAtom.GetParameters](#) method and the operator and parameters can be modified as required.

Finally the content stream can be reassembled using the [Atom.GetData](#) method and then the raw data inserted back into the original PDF.

---

#### Analysis

This sequence allows fast, complex, content stream deconstruction and manipulation using a memory efficient model. For example it can be used to [search a PDF page for particular types of color operators and replace them with different ones](#). PDF color replacement until now has been difficult and error prone. This new functionality makes the process easy.

Along with the new functionality for the manipulation of PDF content streams we also have a new and useful set of low level functionality to allow the

manipulation of text in existing PDFs. The `StringAtom` has new `Decode` and `DecodeDoubleByte` methods to allow text operator parameters to be decoded into the base text encoding. These can then be passed through the `FontObject` `EncodingToChar` and `EncodingToString` properties to allow mapping from the text encoding through to Unicode values.

Going the other way you can use the `FontObject` `CharToEncoding` to map Unicode values to the font encoding and then `StringAtom` `Encode` or `EncodeDoubleByte` to put the text into a format which can be inserted into a content stream.

There are lots of things to make your life a bit easier. Functions like `Doc.AddRect` to better reflect similar functions such as `AddPie` and `AddOval`. A new `FormXObject` and methods to convert from objects such as a `Page` into a `FormXObject` - useful if you need to perform operations such as `scaling an existing page`. An `AddXObject` method to draw a `Pixmap` or `FormXObject` onto the current page.

The `Page` object now implements useful properties like the `MediaBox`, `CropBox`, `BleedBox`, `TrimBox` and `ArtBox` properties. Previously only some of these were available and those were read only. Now they work in a much more intuitive way. There is a new `Thumbnail` property for accessing or setting a thumbnail for the page and there are examples showing how to `insert` or `extract` them. The new `Page.GetBitmap` method allows you to render one or more layers on the page and is ideal for generating `drop shadows`.

There is a new document state property to enable

features like a [graphics state stack](#), with push and pop operators, to be simply and easily implemented.

---

Ease of  
Use

The `Pixmap` object now allows construction from an `XImage` object which means you can add an image to a document without adding it to a specific page. The `Pixmap` class contains `Mask` and `SMask` properties so that you can access or assign new masks or soft masks. There are useful functions like `Flip` and `Rotate` for commonly requested bitmap operation. There is a `SetBitmap` method to go with the existing `GetBitmap` one and there is a new `Save` function to allow the `Pixmap` to be saved in its native color space.

The [AccessibilityOperation example code](#) includes new functionality for enhanced compatibility with common screen readers such as NVDA (NonVisual Desktop Access).

The `TextOperation` now allows the [color space of the text color](#) to be determined. This allows you to detect text drawn using specific spot colors.

Encryption includes automatic support for the new Adobe AES revision / version 5 algorithm. This provides extra security and is now the standard for new releases of Acrobat.

Render quality is enhanced in a number of areas most notably line stroking and anti-aliased clipping.

# Acknowledgements

---



ABCpdf incorporates a variety of freely available and open source software. Many thanks to the authors and contributors.

If you are planning to redistribute ABCpdf you should reproduce these acknowledgments in your documentation.

## Portable Document Format

Adobe Systems Incorporated owns the copyright for the particular data structures and operators and the written specification constituting the interchange format called the Portable Document Format (PDF).

Adobe gives copyright permission for this material to be used for generating, viewing, printing and otherwise manipulating PDF documents. This is subject to certain conditions which are designed to maintain the integrity of the PDF standard.

For definitive details see the Adobe PDF Reference 5th Edition.

## SWF

Adobe Systems Incorporated owns the copyright for the particular data structures and the written specification constituting the interchange format called the SWF File Format.

Adobe gives copyright permission for this material to be used for generating and manipulating SWF files. This is subject to certain conditions which are designed to maintain the integrity of the SWF standard.

For definitive details see the Adobe SWF File Format Specification Version 9.

jpeg

This software is based in part on the work of the Independent JPEG Group. Find out more at <http://www.ijg.org/>.

libtiff

A library for processing and manipulating TIFF images. Find out more at <http://www.libtiff.org/>.

Copyright (c) 1988-1997 Sam Leffler  
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

## zlib

A free, general-purpose, legally unencumbered - that is, not covered by any patents - lossless data-compression library. Find out more at <http://www.gzip.org/zlib/>.

Copyright (C) 1995-2002 Jean-loup Gailly and Mark Adler

# FreeType

Portions of this software are copyright © 1996-2002 The FreeType Project ([www.freetype.org](http://www.freetype.org)). All rights reserved.

## Little cms

Little cms is a fast and efficient color management engine. Find out more at <http://www.littlecms.com/>. Little cms is distributed under the following license.

Copyright (C) 1998-2004 Marti Maria

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## The Legion Of The Bouncy Castle

The Legion of the Bouncy Castle provide a comprehensive cryptography package. Find out more at <http://www.bouncycastle.org/>. The software is distributed under the following license.

The Bouncy Castle License

Copyright (c) 2000-2004 The Legion Of The Bouncy Castle (<http://www.bouncycastle.org>)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## OpenOffice.org

ABCpdf is designed to integrate with OpenOffice.org if it is installed on your system.

OpenOffice.org is a fully-featured open-source office productivity suite available as a free download for major platforms in over 45 supported languages. It is compatible with competing office suites. OpenOffice.org is developed, supported, and promoted by an international community operating from the <http://www.openoffice.org/> website. Beginning with the upcoming 2.0 release, OpenOffice.org will store data in the open XML file format adopted by the international standards body OASIS.

## Fonts

Selected fonts and glyphs created by BenJamin P. Johnson.

# Xerces

This product uses Xerces which is licensed under the Apache License as detailed below. It is not a Derivative Work as it merely links to the interfaces of Xerces.

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

## TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise

of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

## XPS

This product may incorporate intellectual property owned by Microsoft Corporation. The terms and conditions upon which Microsoft is licensing such intellectual property may be found at <http://go.microsoft.com/fwlink/?LinkId=52369>.

## XML schemas

Portions of this software may use XML schemas Copyright (c) 2006 [DCMI](#), the Dublin Core Metadata Initiative. These are licensed under the [Creative Commons 3.0 Attribution](#) license.

## :[diStorm64]:

diStorm64 is an open-source disassembler library for x64, licensed under the BSD License.

:[diStorm64]:  
The ultimate disassembler library.  
Copyright (c) 2003,2004,2005,2006,2007,2008, Gil Dabah  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the diStorm nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Boost

Boost provides free peer-reviewed portable C++ source libraries.

:Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE..

## This software is based in part on JasPer.

JasPer License Version 2.0

Copyright (c) 2001-2006 Michael David Adams  
Copyright (c) 1999-2000 Image Power, Inc.  
Copyright (c) 1999-2000 The University of British Columbia

All rights reserved.

Permission is hereby granted, free of charge, to any person (the "User") obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

1. The above copyright notices and this permission notice (which includes the disclaimer below) shall be included in all copies or substantial portions of the Software.
2. The name of a copyright holder shall not be used to endorse or promote products derived from the Software without specific prior written permission.

THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF THE SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER. THE SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. NO ASSURANCES ARE PROVIDED BY THE COPYRIGHT HOLDERS THAT THE SOFTWARE DOES NOT INFRINGE THE PATENT OR OTHER INTELLECTUAL PROPERTY RIGHTS OF ANY OTHER ENTITY. EACH COPYRIGHT HOLDER DISCLAIMS ANY LIABILITY TO THE USER FOR CLAIMS BROUGHT BY ANY OTHER ENTITY BASED ON INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OR OTHERWISE. AS A CONDITION TO EXERCISING THE RIGHTS GRANTED HEREUNDER, EACH USER HEREBY ASSUMES SOLE RESPONSIBILITY TO SECURE ANY OTHER INTELLECTUAL PROPERTY RIGHTS NEEDED, IF ANY. THE SOFTWARE IS NOT FAULT-TOLERANT AND IS NOT INTENDED FOR USE IN MISSION-CRITICAL SYSTEMS, SUCH AS THOSE USED IN THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION OR COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL SYSTEMS, DIRECT LIFE SUPPORT MACHINES, OR WEAPONS SYSTEMS, IN WHICH THE FAILURE OF THE SOFTWARE OR SYSTEM COULD LEAD DIRECTLY TO DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE ("HIGH RISK ACTIVITIES"). THE COPYRIGHT HOLDERS SPECIFICALLY DISCLAIM ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR HIGH RISK ACTIVITIES.

## This software is based in part on libxml.

Except where otherwise noted in the source code (trio files, hash.c and list.c) covered by a similar licence but with different Copyright notices:

Copyright (C) 1998-2002 Daniel Veillard. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

## Trio Files

\* Copyright (C) 1998 Bjorn Reese and Daniel Stenberg.

\*

\* Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

\*

\* THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

## Hash.c

\* Copyright (C) 2000 Bjorn Reese and Daniel Veillard.

\*

\* Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

\*

\* THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

## List.c

\* Copyright (C) 2000 Gary Pennington and Daniel Veillard.

\*

\* Permission to use, copy, modify, and distribute this software for any  
\* purpose with or without fee is hereby granted, provided that the above  
\* copyright notice and this permission notice appear in all copies.

\*

\* THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED  
\* WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF  
\* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND  
\* CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

# ImageMagick

This product uses ImageMagick (<http://www.imagemagick.org/>), which is available under the following license:

The legally binding and authoritative terms and conditions for use, reproduction, and distribution of ImageMagick follow:

Copyright 1999-2009 ImageMagick Studio LLC, a non-profit organization dedicated to making software imaging solutions freely available.

## 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication intentionally sent to the Licensor by its copyright holder or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License.

APPENDIX: How to apply the ImageMagick License to your work

To apply the ImageMagick License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format.

Copyright [yyyy] [name of copyright owner]  
Licensed under the ImageMagick License (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.imagemagick.org/script/license.php>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

## In addition the following Notice is required to be included by the ImageMagick license:

The ImageMagick logo is copyright Pineapple USA Inc. It is freely distributable, however, modifications (other than resizing) to the logo are not permitted.

----

ImageMagick incorporated a small portion of code from the gsvie package to locate Ghostscript under Windows in `magick/nt_base.c`. The source code is distributed under the following license:

Copyright (C) 2000-2002, Ghostgum Software Pty Ltd. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this file ("Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of this Software, and to permit persons to whom this file is furnished to do so, subject to the following conditions:

This Software is distributed with NO WARRANTY OF ANY KIND. No author or distributor accepts any responsibility for the consequences of using it, or for whether it serves any particular purpose or works at all, unless he or she says so in writing.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

----

The `Base64Decode()` and `Base64Encode()` methods in `magick/utility.c` is based on source code obtained from OpenSSH. The source code is distributed under the following license:

Copyright (c) 2000 Markus Friedl. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following

disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

----

ImageMagick includes patterns in coders/pattern.c which are derived from XFig and is distributed under the following license:

FIG : Facility for Interactive Generation of figures Copyright (c) 1985-1988 by Supoj Sutanthavibul Parts Copyright (c) 1989-2000 by Brian V. Smith Parts Copyright (c) 1991 by Paul King

Any party obtaining a copy of these files is granted, free of charge, a full and unrestricted irrevocable, world-wide, paid up, royalty-free, nonexclusive right and license to deal in this software and documentation files (the "Software"), including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons who receive copies from any such party to do so, with the only requirement being that this copyright notice remain intact.

----

In November 2002, the GraphicsMagick Group created GraphicsMagick from ImageMagick Studio's ImageMagick source. ImageMagick adopted some of their improvements to existing programs and scripts under the following license:

Copyright (C) 2002 GraphicsMagick Group, an organization dedicated to making software imaging solutions freely available.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("GraphicsMagick"), to deal in GraphicsMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of GraphicsMagick, and to permit persons to whom GraphicsMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of GraphicsMagick.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall GraphicsMagick Group be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with GraphicsMagick or the use or other dealings in GraphicsMagick.

Except as contained in this notice, the name of the GraphicsMagick Group shall not be used in advertising or otherwise to promote the sale, use or other dealings in GraphicsMagick without prior written authorization from the GraphicsMagick Group.

----

ImageMagick makes use of third-party "delegate" libraries to support certain optional features. These libraries bear their own copyrights and licenses, which may be more or less restrictive than the ImageMagick license.

## Mozilla XULRunner

This product uses a modified version of Mozilla XULRunner available under the [Mozilla Public License \(http://www.mozilla.org/MPL/\)](http://www.mozilla.org/MPL/).

In compliance with the license, the source code required to produce the modified version of Mozilla Firefox binaries used in this product can be obtained from [bitbucket.org](http://bitbucket.org).

Full license details can be found in the source code for the specific release associated with the product.

# ExplorerCanvas

Modern browsers like Firefox, Safari, Chrome and Opera support the HTML5 canvas tag to allow 2D command-based drawing. ExplorerCanvas brings the same functionality to Internet Explorer. ExplorerCanvas is available under the Apache License detailed below.

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

## TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 4 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind together in the same file) the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received and subsequently incorporated within the Work.

## 2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly sublicense, and distribute the Work and such Derivative Works in Source or Object form.

## 3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

## 4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, in either Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must carry a copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms for any such Derivative Works as a whole, provided Your use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, or distribution of the Work otherwise complies with the conditions stated in this License.

## 5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be for all rights under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, any Contributions you submit may be made under an alternative licensing arrangement, provided that such arrangement is in no way subordinate to the license in force under the terms and conditions of this License, and provided that you state the applicable alternative licensing arrangement in a separate document submitted with the Contribution. Such alternative licensing arrangements shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding your Contributions.

## 6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of third parties as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the Work.

## 7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including but not limited to any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assuming any liability associated with Your exercise of permissions under this License.

## 8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

## 9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of a warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in a comment block for the file format. We also recommend that a best practice comment syntax for the file format be included on the same line prior to the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]
```

Licensed under the Apache License, Version 2.0 (the "License"  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.

# Mhook

## Part of this software uses the Mhook library.

Copyright (c) 2007-2008, Marton Anka Portions Copyright (c) 2007, Matt Conover

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# HarfBuzz

Copyright © 2010,2011,2012 Google, Inc.  
Copyright © 2012 Mozilla Foundation  
Copyright © 2011 Codethink Limited  
Copyright © 2008,2010 Nokia Corporation and/or its subsidiary(-ies)  
Copyright © 2009 Keith Stribley  
Copyright © 2009 Martin Hosken and SIL International  
Copyright © 2007 Chris Wilson  
Copyright © 2006 Behdad Esfahbod  
Copyright © 2005 David Turner  
Copyright © 2004,2007,2008,2009,2010 Red Hat, Inc.  
Copyright © 1998-2004 David Turner and Werner Lemberg

This is part of HarfBuzz, a text shaping library.

Permission is hereby granted, without written agreement and without license or royalty fees, to use, copy, modify, and distribute this software and its documentation for any purpose, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE COPYRIGHT HOLDER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE COPYRIGHT HOLDER SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE COPYRIGHT HOLDER HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

# UCDN

Copyright (C) 2012 Grigori Goronzy <greg@kinoho.net>

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

## The LZW code is covered by the following licenses.

/\*

\* Copyright (c) 1988-1997 Sam Leffler  
\* Copyright (c) 1991-1997 Silicon Graphics, Inc.

\*

\* Permission to use, copy, modify, distribute, and sell this software and  
\* its documentation for any purpose is hereby granted without fee, provided  
\* that (i) the above copyright notices and this permission notice appear in  
\* all copies of the software and related documentation, and (ii) the names of  
\* Sam Leffler and Silicon Graphics may not be used in any advertising or  
\* publicity relating to the software without the specific, prior written  
\* permission of Sam Leffler and Silicon Graphics.

\*

\* THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,  
\* EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY  
\* WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

\*

\* IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR  
\* ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND,  
\* OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,  
\* WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF  
\* LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE  
\* OF THIS SOFTWARE.

\*/

/\*

\* TIFF Library.  
\* Rev 5.0 Lempel-Ziv & Welch Compression Support

\*

\* This code is derived from the compress program whose code is  
\* derived from software contributed to Berkeley by James A. Woods,  
\* derived from original work by Spencer Thomas and Joseph Orost.

\*

\* The original Berkeley copyright notice appears below in its entirety.

\*/

/\*

\* Copyright (c) 1985, 1986 The Regents of the University of California.  
\* All rights reserved.

\*

\* This code is derived from software contributed to Berkeley by  
\* James A. Woods, derived from original work by Spencer Thomas  
\* and Joseph Orost.

\*

\* Redistribution and use in source and binary forms are permitted  
\* provided that the above copyright notice and this paragraph are  
\* duplicated in all such forms and that any documentation,  
\* advertising materials, and other materials related to such  
\* distribution and use acknowledge that the software was developed  
\* by the University of California, Berkeley. The name of the  
\* University may not be used to endorse or promote products derived  
\* from this software without specific prior written permission.

\* THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR  
\* IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED  
\* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

\*/

## Random Numbers

One of the random number generators we use is covered by the following license.

```
/**
 * Copyright (c) 1983 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Berkeley. The name of the
 * University may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */
```

## We make use of jemalloc for some memory management.

Unless otherwise specified, files in the jemalloc source distribution are subject to the following license:

-----  
Copyright (C) 2002-2014 Jason Evans <jasone@canonware.com>.  
All rights reserved.  
Copyright (C) 2007-2012 Mozilla Foundation. All rights reserved.  
Copyright (C) 2009-2014 Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

-----

# Legal Requirements

---

Adobe claims Intellectual Property rights over the Adobe PDF specification and we aim to ensure that those rights are upheld.

When you install ABCpdf you accept the ABCpdf license agreement. Part of the agreement specifies that you must abide by the conditions of use for PDF set down by Adobe and detailed in the Adobe PDF Specification.

For full details you should see the specification. However in summary, Adobe permits you to write software to create, display and manipulate PDF documents.



This permission is conditional on the basis that your software should respect the permissions and permission controls embedded within existing PDF documents.

These permissions do not necessarily make sense within the ABCpdf framework. For example the concept of an encrypted document which bars the user from filling in form fields makes sense if there is a user. However ABCpdf is a component and so is removed from the final user.

You should ensure that when you create software which an end user may operate that your software respects these types of permissions. You are legally bound to do this both by the ABCpdf license agreement and also by the Adobe conditions of use.





## .NET Essentials

---

ABCpdf .NET is made up of the following components.

ABCpdf.dll is the main .NET assembly. When the assembly is loaded, it locates and loads an appropriate core engine establishing a direct high speed link between the two components.

ABCpdf10-32.dll and ABCpdf10-64.dll are the core engines for 32- and 64-bit processes. They incorporate our proprietary Direct to PDF technology and are designed for high performance PDF manipulation in a multithreaded environment.

PrintHook32.dll and PrintHook64.dll are print hooks for 32- and 64-bit processes. These DLLs intercept the Microsoft XPS Document Writer to allow ABCpdf to import an extended range of documents.

ABCpdf supports the import of more image formats through the separate ABCImageMagick COM+ Application, which is not automatically installed and you can manually install with ABCImageMagick.msi.

The .NET tier is placed in the GAC so that you can reference ABCpdf .NET from any of your projects. However, should you require, you can always copy both DLLs to the bin directory of your application.

---

### DLLs

**Copied DLLs.** Visual Studio copies PresentationCore.dll and System.Printing.dll, which are referenced from

ABCpdf, to your application/bin directory because they are platform-specific (32-bit vs 64-bit). This causes problems when your application is published or copied to a machine with a different platform.

To prevent the copying, you can directly reference the file and set Copy Local and Specific Version to False by adding the following lines to your project file (.csproj/.vbproj) under <ItemGroup> with other Reference elements:

```
<Reference Include="PresentationCore">  
  <SpecificVersion>False</SpecificVersion>  
  <Private>False</Private>  
</Reference>  
<Reference Include="System.Printing">  
  <SpecificVersion>False</SpecificVersion>  
  <Private>False</Private>  
</Reference>
```

You need to add a reference to ABCpdf from your Visual Studio Project.

This tells Visual Studio to link the ABCpdf assembly into the build.

---

#### Refs

If you are not using Visual Studio, you will need to consult the documentation for your chosen development environment.

There are four public namespaces in ABCpdf. You can reference these using the following directives.

## [C#]

```
using WebSupergoo.ABCpdf10;  
using WebSupergoo.ABCpdf10.Objects;  
using WebSupergoo.ABCpdf10.Atoms;  
using WebSupergoo.ABCpdf10.Operations;
```

## [Visual Basic]

```
Imports WebSupergoo.ABCpdf10  
Imports WebSupergoo.ABCpdf10.Objects  
Imports WebSupergoo.ABCpdf10.Atoms  
Imports WebSupergoo.ABCpdf10.Operations
```

---

### Names

The ABCpdf10 namespace contains the objects you will use for page layout. Most of the time, it is the only namespace you will need.

The Objects namespace allows you to access and manipulate content you've already added. You may use this namespace for complex operations in which the standard page layout functionality requires some modification.

The Atoms namespace allows you low level access to the raw PDF data structures. You are unlikely to use objects from this namespace unless you are writing very low level code.

The Operations namespace allows you to perform complex operations with multiple parameters and callbacks.

This is some simple example code. All it does is create a simple 'Hello World' PDF in the current working directory.

### [C#]

```
Doc doc = new Doc();  
doc.AddText("Hello World!");  
doc.Save("output.pdf");
```

---

Example

### [Visual Basic]

```
Dim doc As New Doc()  
doc.AddText("Hello World!")  
doc.Save("output.pdf")
```

ASP.NET operates under a restricted set of security permissions. It is quite common for the ASPNET user not to be able to create or write files.

---

Security

So, if you want to save a PDF file from your ASP.NET code, it is quite likely that you will need to adjust the permissions on your destination directory to allow write access for the ASPNET user.



## Simple Example

---

Creating a PDF document is a simple process. First you create an ABCpdf Document object and then you add your content to it. You can add [text](#), [images](#) and other kinds of graphics.

All content addition is done on the current [Page](#) and within the current [Rect](#) on that page. You can change the Page to draw on different pages and the Rect to draw in different areas. The default page is the first page and the default drawing area is the entire page.

---

### Intro

You may find it useful to use the [FrameRect](#) method during development. This frames the current rectangle on the current page so that the area you are about to draw on is outlined.

Every time you add an item of content you will get an Object ID returned. You may wish to save these IDs and use them to query or change object properties at a later date.

First we create an ABCpdf Doc object. Next we set the font size and add some text to it. Finally we save at a specified location and clear our document.

### [C#]

```
Doc theDoc = new Doc();  
theDoc.FontSize = 96;
```

---

Code

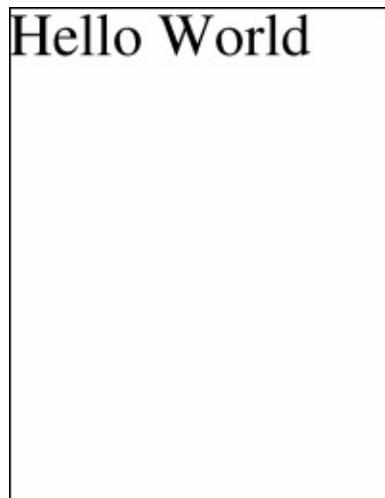
```
theDoc.AddText("Hello World");  
theDoc.Save(Server.MapPath("simple.pdf"));  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.FontSize = 96  
theDoc.AddText("Hello World")  
theDoc.Save(Server.MapPath("simple.pdf"))  
theDoc.Clear()
```

---

Results



simple.pdf



## Coordinate Spaces

---

ABCpdf uses the standard Adobe PDF coordinate space. The origin of this space is at the bottom left of the document. Distances are measured up and to the right in points. Points are a traditional measure for print work - there are 72 points in an inch.

**Please note that the bottom-up PDF coordinate space is different from the top-down coordinate system often used in Windows. It means that everything is based around the bottom left of objects - not the top left.** If you wish to use a different coordinate system you can use the document [Units](#), [TopDown](#) or [Transform](#) to accomplish this.

---

### General

ABCpdf uses the [XPoint](#) object to represent positions in space. It uses the [XRect](#) object to represent areas like the current drawing area or the size of a page.

The [Doc.Rect](#) property is probably the most important property to be aware of. **Virtually everything happens within the Doc.Rect.** If you add text to a document it is added within the Doc.Rect. If you paint or frame a rectangle it is done within the Doc.Rect. If you add an image it is scaled to fit exactly within the Doc.Rect.

The default document size for ABCpdf is 612 by 792. This equates to a physical page size of 8.5 by 11 inches.

The following example draws a rectangle with the bottom left corner positioned 100 points from the left of the page and :

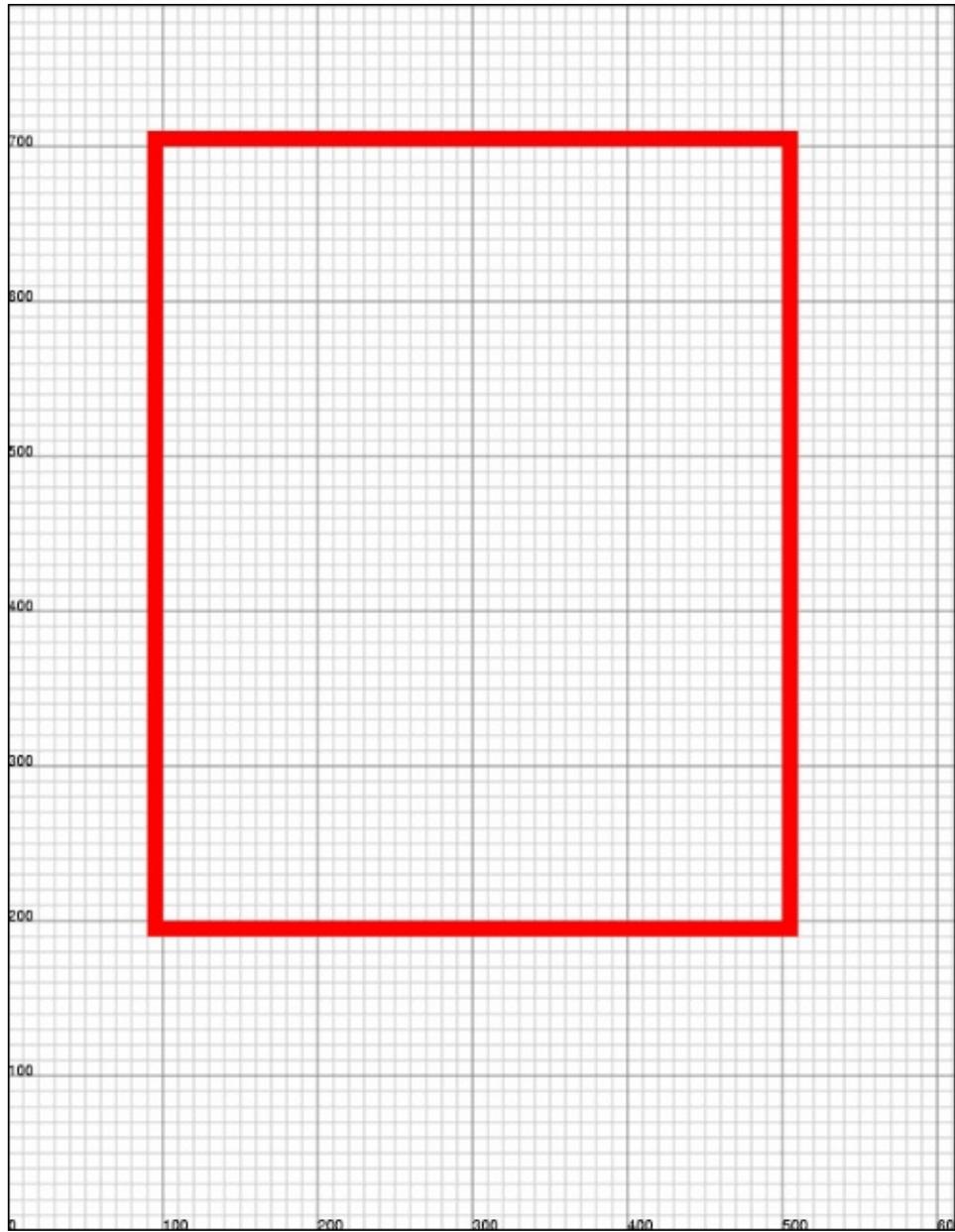
points up from the bottom. The width of the rectangle is 40 points and the height is 500 points. We add a grid to show positioning of the rectangle.

### **[C#]**

```
Doc theDoc = new Doc();
theDoc.AddGrid();
theDoc.Color.String = "255 0 0";
theDoc.Width = 10;
theDoc.Rect.Position(100, 200);
theDoc.Rect.Width = 400;
theDoc.Rect.Height = 500;
theDoc.FrameRect();
theDoc.Save(Server.MapPath("coordinates.pd
```

### **[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
theDoc.AddGrid()
theDoc.Color.String = "255 0 0"
theDoc.Width = 10
theDoc.Rect.Position(100, 200)
theDoc.Rect.Width = 400
theDoc.Rect.Height = 500
theDoc.FrameRect()
theDoc.Save(Server.MapPath("coordinates.pd
```



coordinates.pdf



## Examples

---

There are many code examples in the documentation. These cover common tasks like [adding text](#), [flowing text](#) and [adding images](#).

Every major object method or property has an accompanying code sample. So if you want to know how to use a method like [AddText](#) or [AddImageFile](#) just look at the code sample.

Code samples also cover operations like [rendering HTML pages](#), [paged HTML renders](#), [watermarking](#), [appending PDF documents](#) and [drawing pages from one PDF document into another](#).

For in-situ examples you should look at the example web site that comes installed with ABCpdf.



## Upgrading

---

ABCpdf 10 is a new version completely independent of the old. It incorporates the ABCpdf2, ABCpdf3, ABCpdf4, ABCpdf5, ABCpdf6, ABCpdf7, ABCpdf8 and ABCpdf9 namespaces so that you can upgrade with minimal changes to your code. When you want to take advantage of the new features, simply reference the new name.

Simply replace...

### [C#]

```
using WebSupergoo.ABCpdf9;
```

### [Visual Basic]

```
Imports WebSupergoo.ABCpdf9
```

---

#### Basics

with...

### [C#]

```
using WebSupergoo.ABCpdf10;
```

### [Visual Basic]

```
Imports WebSupergoo.ABCpdf10
```

Unless you have side-by-side versioning (available in Windows XP), you will not be able to have multiple versions of ABCpdf installed in the GAC. In this case, you should install the relevant assemblies in the bin directories of your application.

ABCpdf is fully backward compatible. Although extensive changes have been made to the core engine, we check that these changes produce results that are compatible with previous versions.

There are some minor differences in types between the ABCpdf9 and ABCpdf10 namespaces.

The XColor.ColorSpace property has been changed from an integer to an enum. However since the new enum values are the same as the old integer values you can just cast between the two. See the [XColor.ColorSpace](#) property for details and examples.

There are some minor stylistic differences between best practice in the old and new namespaces. Any detected issues will result in a compile time warning and a suggestion for a substitute line of code. We would recommend that you make these changes but you are safe to ignore the warnings if you do not have time to do so.

There are some minor differences in behavior between the ABCpdf9 and ABCpdf10 namespaces. These differences are improvements but if you have been relying on the old behavior you may wish to disable them.

Kerning is enabled for higher quality text output. However it does mean that the positioning of letters in text inserted using methods like AddText and AddHtml, may be slightly different. If you have been relying on exactly the same positioning you will want to use the following line of code after creating any Doc object.

```
doc.TextStyle.Kerning =  
KerningType.None;
```

Images are now always inserted at the size that is specified. In earlier versions of ABCpdf, if the width or height of the destination rectangle was zero, the rectangle was expanded to insert them at 72 DPI. This was not logical or intuitive behavior but if you have been relying on it you will want to use the following line of code after creating any Doc object.

```
doc.SetInfo(0, "AutosizeImages", 1);
```

---

#### Changes

TIFF orientation tags are supported. This means that images with these tags will automatically be presented the right way up. However if you have been processing these tags yourself you may wish to disable this behavior. Simply use the following line of code after creating any Doc object.

```
doc.SetInfo(0,  
"AutorotateTIFFImages", 0);
```

PDF Portfolios are now, by default, opened the way a user would see them when opening them in Acrobat. If you have written custom code to extract individual PDFs from a portfolio you will need to set the [XReadOptions.OpenPortfolios](#) property to false before reading them in. You will probably just need to use the following line of code after creating any Doc object.

```
doc.ReadOptions.OpenPortfolios =  
false;
```

Color components in the XColor object used not to

accept values outside the legal limits. In line with Microsoft best practice, this has been changed so that you can store values outside the legal range. The advantage of this is that, say, you might want to store values between 0 and 65535 and then later scale them down to 0 to 255 before using them for PDF drawing. If your code results in illegal color values at the point of drawing, then this change may result in a change in behavior. In the past illegal values would be likely to result in randomly colored objects. In the ABCpdf10 namespace illegal values are likely to result in black, white or invisible objects. If you see this type of symptom you should be looking for a bug in the way you assign color values in your code. Alternatively you can just change your code to use the behavior in the old namespace. To do this for grayscale and RGB values you need to modulus your values with 256 at the point of assignment. For CMYK values you need to pin them from 0 to 100 at the point of assignment.

The `TextStyle.String` property has been updated to reflect various state properties which had been left out in earlier versions. If you are using the `TextStyle.String` to store state, this should mean that your code will now store all `TextStyle` state rather than just a part of it.

# Object Paths

---

Adobe Portable Document Format (PDF) files are made of a number of objects. Objects may describe a page, a resource, a sequence of drawing operations an image or many other components as required by the document.



Every object has a unique Object ID numbered from one upwards. The ID zero refers to the Empty Object which is an object required internally within ABCpdf. ID -1 refers to the document trailer. The root of the document hierarchy can be accessed using the [Doc.Root](#) property or the [ObjectSoup.Catalog](#).

You can use the [GetInfo](#) and [SetInfo](#) methods to directly manipulate any PDF object in your document. However, this is not advisable unless you are reasonably familiar with the Adobe PDF Specification.

Under normal situations, ABCpdf ensures that your documents are internally consistent. Using the [SetInfo](#) method with [Dictionaries](#), [Values](#) or [Paths](#) allows great flexibility in modifying documents but also allows you to create invalid or corrupt documents.

## Dictionaries

If your object is a dictionary, you can specify a particular dictionary entry for replacement or insertion (dictionary entries always begin with a slash '/' character). So if you wanted to change the type of an annotation, you might use the following code:

```
theDoc.SetInfo(theID, "/Subtype", "(Stamp)")
```

## Values

Alternatively, you can use the 'Value' selector to specify a replacement for the entire object. However, if you do this, you must ensure that the type of your new object is the same as the type of your old one - you cannot replace a number with a string. For example.

```
theDoc.SetInfo(theID, "Value", "<</Font /Helvetica  
/Size 10>>")
```

## Paths

Specifications can be chained together to form complete paths. Dictionary entries are specified by preceding the entry name with a slash. Array items are specified using square brackets containing the index of the item you wish to reference (starting at zero).

For example, the code below would return the first item in the MediaBox array.

```
theDoc.GetInfo(theID, "/MediaBox[0]")
```

And the code below would return the count entry of the parent of the object.

```
theDoc.GetInfo(theID, "/Parent/Count")
```

## References

Sometimes, you may wish to find a reference to a particular object. Sometimes, you may wish to skip through the reference and jump straight into the object itself.

You can do this using an asterisk to de-reference an object within a path. If the object is a reference, it will be de-referenced; if it is not, then the operator will be ignored.

For example, the code below might be used to return the content stream of the first page of a document.

```
theDoc.GetInfo(theDoc.Root, "/Pages*/Kids*[0]*/Contents")
```

## SetInfo

You can use [SetInfo](#) to insert values specified by paths. You can specify the type of object to be inserted by appending an identifier to the path.

Object Type	Description
:Bool	A Boolean value.
:Name	A name value.
:Num	A numeric value
:Text	A string value.
:Ref	An indirect reference to an object.
:Rect	A rectangle. Internally, rectangles are held as arrays of numbers, but this provides a convenient shortcut.
:Del	This is a special entry. It does not insert an object. Instead, it ensures that the object specified by the path is deleted.
	<p>The asterisk is a delimiter. Each of 'Hex', 'NoBreak', and 'Byte' is optional. For example, you can specify <code>"/MyEntry:Text[Hex]"</code>.</p> <p>The 'Hex' specifier indicates that the text should</p>

:Text[Hex*NoBreak]	be hex-encoded. The 'NoBreak' specifier indicates that there should be no line breaks. Also available is the 'Byte' specifier, which indicates that characters should be interpreted as a string of bytes rather than a Unicode string. These settings are used infrequently, and in general, you will not need them.
:String	<p>This only works for string atoms.</p> <p>If the string atom contains a value it returns that value.</p> <p>If there is no value, or the atom is of the wrong type, it returns the special string.</p> <p>"string_is_null\0ABCpdf"</p>

Suppose you wanted to insert an annotation into the page annotations array. The following code will find the page entry named "/Annots" (or it will create it if it doesn't exist). It will then ensure that this entry references an array, and it will insert a reference Atom at the beginning (item zero) of the array.

```
theDoc.SetInfo(theDoc.Page, "/Annots[0]:Ref",
theID)
```

Alternatively, if you want to insert your annotation at the end of the array, just leave out the array index:

```
theDoc.SetInfo(theDoc.Page, "/Annots[]:Ref", theID)
```

You can also locate items in an array from the end. Use -1 for the last item, -2 for the second last item, and so on.

```
theDoc.GetInfo(theDoc.Page, "/Annots[-1]:Ref")
```

Insertions can be complex. The next example gets the entry called "/Font", which contains a dictionary. This dictionary includes an element called "/Names", which contains an array. The call inserts the Name object "/Helvetica" at the start of this array.

```
theDoc.SetInfo(theID, "/Font/Names[0]:Name",  
"Helvetica")
```

## GetInfo

You can use [GetInfo](#) to query values specified by paths. The format of the return value is exactly the same as would be output to your PDF file. You can specify an alternative format by appending an identifier to the string.

Format Name	Description
:ID	<p>The Object ID associated with an object reference.</p> <p>Normally, object references are returned in an extended format (e.g. <code>23 0 R</code>). However, if you are only interested in the Object ID, then you use this format specifier to get only the Object ID (e.g. <code>23</code>).</p>
:Obj	<p>The object value.</p> <p>This is used to ensure that all indirect references are resolved before the value of the object is returned. This ensures you always get an object value rather than an object reference.</p>
:Text	<p>The text of a name or string.</p> <p>Names and strings may be encoded in a number of ways before output to PDF. The text format specifier ensures that the unencoded value is returned.</p>
	<p>The value of a number.</p>

:Num	If the object referred to is not a number, then no value is returned.
:Rect	The rect string for a rectangle object. Rects are typically represented as an array. By specifying the rect format, you will get a string value you can place directly into the XRect object.
:Count	The number of items in an array. The count specifier is a special directive which returns the number of items in an array rather than an item from that array.
:KeyCount	The number of items in a dictionary. The keycount specifier is a special directive which returns the number of items in a dictionary rather than an item from that dictionary.
:Keys	The keys for a dictionary. The keys specifier is a special directive which returns a comma delimited list of the names of the entries in the dictionary.

For example, the code below could return the rect of the page CropBox.

```
theDoc.GetInfo(theID, "/CropBox:Rect")
```



# XML to PDF

---

When ABCpdf was designed a conscious design decision was made not to make it XML based. While XML templates might be an obvious route to take there are problems with this approach.

The key factor is that the creation of a PDF document is an interactive one. Often you don't know how much text you're going to be adding until you're given the text. You may need to flow the text from one column to another, you may need to create a new page to hold any extra text or you may choose to reduce the font size so that all your text fits. These are the kinds of decision that are very difficult to describe in terms of XML.



Although it is easy to create an XML based PDF generator using a product like ABCpdf it is not possible to work the other way round. So you can produce template based documents using ABCpdf. If you were to want to implement interactive code using an XML based solution this would be impossible.

For an example of how to convert XML content to PDF using ABCpdf see the Tagged PDF Example project under the ABCpdf menu item. As well as demonstrating how to convert XML to PDF it also demonstrates how to add semantic tags to the document at the same time.



# Manual Installation

---

Under most circumstances, you will want to install ABCpdf .NET (ABCpdf10-32.dll) and ABCpdf10-64.dll. Occasionally, you may wish to install ABCpdf manually. To do this

File Name	Notes
ABCpdf.dll	<p>The Assembly used by .NET.</p> <p>The installer places this Assembly in the %ProgramFiles%\WebSupergoo\ABCpdf folder. However, if you are installing manually, you should place it in the following location:</p> <p>%ProgramFiles%\WebSupergoo\ABCpdf</p>
ABCpdf10-32.dll	<p>The ABCpdf core engine.</p> <p>This DLL contains the core engine. It improves performance PDF manipulation in a multi-processor environment. It is installed in the following location:</p> <p>%SystemRoot%\System32\</p> <p>However, if you are installing manually, you should place it in the following location:</p>
ABCpdf10-64.dll	<p>Same as ABCpdf10-32.dll but for 64-bit operating systems.</p> <p>Please note that the 32 bit version of this DLL is required for 32-bit systems.</p>
	<p>This is an optional component required for ABCpdf to function on 64-bit systems.</p>

3DGlue10-32.dll	This DLL is normally located in the same systems.
3DGlue10-64.dll	<p>This is an optional component required for ABCpdf.</p> <p>Same as 3DGlue10-32.dll but for 64-bit operating systems.</p> <p>Please note that the 32-bit version of this DLL is also required for 64-bit systems.</p>
PrintHook32.dll	<p>This is an optional component required for ABCpdf.</p> <p>It intercepts the Microsoft XPS Document Writer print jobs.</p> <p>This DLL is normally located in the same systems.</p>
PrintHook64.dll	<p>This is an optional component required for ABCpdf.</p> <p>Same as PrintHook32.dll but for 64-bit operating systems.</p> <p>Please note that the 32-bit version of this DLL is also required for 64-bit systems.</p>
ABCGeckoWP.exe, XULRunner38_0 (directory)	<p>This is an optional component required for ABCpdf.</p> <p>This EXE should be placed in the bin directory of the XULRunner38_0 folder, which is located in the %SystemRoot%\XULRunner38_0 folder.</p> <p>This EXE is the interface ABCpdf uses to communicate with the XULRunner38_0 folder, which is located in the %SystemRoot%\XULRunner38_0 folder.</p> <p>ABCGecko cannot function without the XULRunner38_0 folder, even if ABCGeckoWP.exe is found only in the %SystemRoot%\XULRunner38_0 folder (because of the way the XULRunner38_0 folder is named).</p>



TaskGardener.exe	<p>This is an optional component required for</p> <p>The <a href="#">Gecko HTML engine</a> runs in separate process. You can specify the user account to directly start a process by using the <code>Doc.HtmlOptions.ProcessOptions.UserName</code> property for ABCpdf.</p> <p>Run it with <code>-install</code> to install it as a Windows service.</p>
ABCImageMagick.msi	<p>This is an optional component.</p> <p>ABCpdf supports the import of more image formats if ImageMagick is automatically installed and you can manually install it if needed.</p>

If you are deploying manually to a restricted permission environment, then you may need to check the registry for certain structures. The `NT\CurrentVersion\Devices` should contain printer entries. The Microsoft requirement. If there are no entries in this key, you can copy them from `NT\CurrentVersion\Devices`. Absence of entries in this registry key

### **Background.** Why XULRunner?

Well, XUL (pronounced zool) is a kind of enhanced XHTML document type acronym and also a reference to the scene in Ghostbusters in

"There is no Dana, only Zuul!"

Since XUL incorporates everything needed for an application

"There is no data, only XUL."

Other references to the Ghostbusters movie are scattered throughout

Most calls to ABCpdf will result in a trial license being installed if i query the value of the [XSettings.LicenseDescription](#) property. Wri

### [C#]

```
using WebSupergoo.ABCpdf10;
MessageBox.Show("New License: " + XSettings.Lic
```

### [Visual Basic]

```
Imports WebSupergoo.ABCpdf10
MessageBox.Show("New License: " + XSettings.Lic
```

You can use a full license key as provided to you when you purch enter a license key, call [XSettings.InstallLicense](#) or at application Redistribution license, you may prefer to call [XSettings.InstallRed](#) unloads.

### [C#]

```
using WebSupergoo.ABCpdf10;
// here we use a trial license key as copied fr
if
(XSettings.InstallLicense("cd9b5c07db69df2bf57c
    MessageBox.Show("License Installed Successful
else
    MessageBox.Show("License Installation Failed"
```

### [Visual Basic]

```
Imports WebSupergoo.ABCpdf10
' here we use a trial license key as copied fro
If
XSettings.InstallLicense("cd9b5c07db69df2bf57c0
Then
    MessageBox.Show("License Installed Successful
Else
    MessageBox.Show("License Installation Failed"
End If
```

**Shared Hosts.** Most installations of ABCpdf .NET on shared server and your host may have locked down permissions in w you or we can do about it.

This is why we recommend deployment on a dedicated serve allow is only one of the very significant advantages they affor

If you are intending to install on a shared server, the essentia problems.



## Fonts and Languages

---

The base fonts are guaranteed to be available on all systems.

Using the base fonts in your PDF results in a small document which is guaranteed to render in the same way on all systems. However the base fonts do not support complex languages such as Chinese and Japanese and they do not support some characters used in Eastern Europe.

The following are the names of the base fonts.

- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Symbol
- ZapfDingbats

---

Base

Use the AddFont method, with the "Latin" language, to reference these fonts.

Embedded TrueType or OpenTypeUnicode fonts are generally the most efficient and reliable way of adding complex language support to your PDF documents. However you need to ensure that you have permission to embed and redistribute your chosen font.

ABCpdf includes options to try and protect you from inadvertently embedding copyrighted fonts. However you should not rely on this protection. If you wish to embed fonts It is vitally important that you think about the way these fonts will be used and that you understand the operations permitted by the copyright holder. Ultimately you need to take responsibility for any fonts you embed in your PDF documents.

When embedding large fonts you should generally choose to subset them. Not only does it produce smaller PDF documents but It is also generally faster to embed a subset of a large font than to embed the entire font.

Use the [Doc.EmbedFont](#) method to embed fonts.

---

**Embed**

The table below shows the valid combinations of language and horizontal and vertical writing directions.

Language	Horizontal	Vertical	Notes
"Latin"	●		The default and most efficient encoding for Western languages.
"Unicode"	●	●	The setting you will most often want to use when embedding

			foreign character sets.
"Korean"			
"Japanese"			
"ChineseS"			Simple Chinese
"ChineseT"			Traditional Chinese

Referenced TrueType or OpenType fonts produce a smaller PDF document. However for complex languages you will need a recent version of Adobe Acrobat and you will need to install the relevant language pack. Without these you will not be able to view your documents and your viewer may report errors.

Use the [Doc.AddFont](#) method to reference fonts.

The table below shows the valid combinations of language and horizontal and vertical writing directions.

Language	Horizontal	Vertical	Notes
"Latin"	●		The default and most efficient encoding for Western languages.
"Unicode"			This selector requires that the font be embedded - it cannot be used for referencing

			fonts.
"Korean"	●	●	Requires the relevant Adobe Acrobat Language Pack
"Japanese"	●	●	Requires the relevant Adobe Acrobat Language Pack
"ChineseS"	●	●	Simple Chinese requires the relevant Adobe Acrobat Language Pack
"ChineseT"	●	●	Traditional Chinese requires the relevant Adobe Acrobat Language Pack

Sometimes, typically when dealing with the Mac platform, you may need to add or embed Type 1 fonts. You can do this using [Doc.EmbedFont](#) or [Doc.AddFont](#) in exactly the same way as you would for any other font.

Type 1 fonts contain a limited set of characters. As such the only language which is available is the "Latin" language.

Unlike TrueType fonts you can reuse Type 1 fonts, previously been embedded in a PDF and read using the [Doc.Read](#) method.

---

## Type 1

To access a font which has been previously embedded in the document search through the objects looking for objects of type 'font'. Assign the ID of the object to the `Doc.Font` property. Note that only standard (WinAnsiEncoding) Type 1 fonts are available in this way.

If you need to obtain the name of the font embedded in a PDF use the following code to get the PostScript name of the font.

```
theDoc.GetInfo(theID,  
"/BaseFont:Name")
```

ABCpdf will allow you to use any valid TrueType or Type 1 (PostScript) font.

If you are having problems accessing a particular font try disabling the ABCpdf font protection functionality. However note that you should not embed and redistribute fonts unless you have permission to do so.

ABCpdf maintains a font cache. This means that for ABCpdf to pick up on a newly installed font you will need to restart any processes that are using ABCpdf.

Alternatively you can pass the path to your font file to the `AddFont` or `EmbedFont` method. This will automatically load the font file. Do not move the font file after doing this - ABCpdf relies on fonts staying in place.

---

## Issues

Sometimes permissions are placed on individual font

files which may restrict access from restricted permission accounts such as ASP or ASP.NET.

Occasionally TrueType fonts are corrupt or non-standard. This can cause problems for ABCpdf (which will refuse to recognize them) or Acrobat (which will refuse to use a font embedded in the PDF). However this type of problem is relatively infrequent and tends to be restricted to unusual fonts such as bar-codes.

If you hit a problem you think is related to a corrupt or nonstandard font please mail us the font and we'll see what we can suggest.



## Fields and Forms

---

PDF allows the creation of interactive forms. Sometimes you will hear these types of PDF referred to as electronic forms, eForms or AcroForms.

To a client an eForm simply looks like a normal PDF containing text boxes, buttons and other interactive elements of the type you typically see on the web.

---

### Basics

Indeed eForms support elements almost identical to HTML forms. They also support mechanisms very similar or identical to HTML forms for content submission to a web server.

For more details see the [Adobe Web Site](#).

Most people coming to eForms will be coming from an HTML background. There are subtle differences between the PDF form model and the HTML form model that can cause confusion.

In HTML a form is part of a web page. Each field in the HTML form has a visible appearance on the page (with the exception of hidden fields). So the concept of a field and what it looks like on the page is pretty much identical.

eForms separate the idea of a field from its visible representation.

Each page of the document owns a set of visible

annotations. There are lots of different types of annotation but the ones that are used in eForm fields are called widgets. A widget contains properties that allow a PDF reader to display it appropriately on the page.

---

## Fields

Each document owns a tree structure of fields which spans the entire document. Each field has a name and also a full name (like a file path) which tells you how to get to that field from the top level of the tree. The leaves of the tree contain the visible widgets that you see on the pages. Note that PDF does not require that full names are unique.

So there are two independent sets of entities each with its own features. The bits you see and which people normally call fields are in fact widgets.

Widgets don't have a name or identity as such - it's just that they may be associated with a field which has an identity. Similarly a field doesn't exist at a location or on a page it is just that it is associated with a widget that exists at a particular location on a particular page.

ABCpdf hides these distinctions from you. When you ask for a field by name it will return you the widget associated with that field. Given the widget you can find out where it is physically located in terms of page numbers and page rectangles. Similarly if you have a widget you can ask it for a field name. Behind the scenes ABCpdf finds the field associated with the widget and returns you the field name.

So why are eForms useful? There are a range of purposes to which eForms can be put. However

probably the most common purpose is to provide placeholders in template PDFs.

---

Why?

Suppose you have a template document you want to use within ABCpdf. Using Acrobat you can insert form fields into the PDF at locations you want content to be inserted. Using ABCpdf you can automatically detect the locations of those fields and enter content at those locations.

If later you want to shift your content a little it's a simple matter to open the PDF in Acrobat and move the fields around.

See the [eForm Example](#) for details.

You can use Acrobat to edit forms using the Advanced Editing tools available under the Tools menu.

However if you choose items from the Forms menu or toolbar then you will probably find that you end up editing your form in Adobe LifeCycle Designer rather than Acrobat.

Adobe Designer is an application which comes with Acrobat Pro. It uses PDF as an output medium. However the way that Designer operates means that forms created by Designer are fundamentally different from forms created by Acrobat.

For example an Acrobat created form typically contains a background and then a set of fields. The fields operate separately from the background.

Adobe Designer created forms do not make this distinction. They use a separate data store to specify

the fields. The PDF content is merely the visible rendition of this field specification. The underlying field specification is made up of chunks of XML embedded in the PDF. This XML format is known as XFA - Adobe XML Forms Architecture.

XFA is referenced in, but is not part of, the ISO standard for PDF.

---

#### Creation

Because Designer documents are PDF documents you can add content to them using standard ABCpdf methods of adding PDF content. However if you then open them then in Designer the content will most likely be deleted because Designer will recreate the PDF appearance using the separate field specification.

Equally because the PDF output is merely the visible rendition of a separate field specification the fields and background may be tied to each other. So you might use ABCpdf to delete a field and find that the border has been left behind.

In extreme cases the LifeCycle documents may not even contain any useful PDF content. Instead the PDF becomes a placeholder which contains an XFA document. The document is only really a thin PDF wrapper around XFA data. You will find that many PDF viewers (including Acrobat Reader X and earlier) cannot display this type of PDF properly.

If you have input to the document creation process, using LifeCycle to save the file as "Adobe Static PDF Form" may help.

If you want to modify forms you will generally find it easier to work with Acrobat created forms than Designer created ones.





## Image Handling

---

ABCpdf embeds images into the PDF at their original resolution. Images may be displayed at different sizes but the underlying image is exactly the same. If your image looks grainy when you print you'll need to find a higher resolution (bigger) image.

---

### Basics

Suppose you have an image 300 pixels wide by 600 pixels high. If you wanted this image to print well at 300 dpi then you would need to add it into a rectangle no larger than 1 inch by 2 inches. If you wanted to view it at a screen resolution of 75 dpi you could place it into a rectangle up to 4 inches by 8 inches.

You can present images to ABCpdf in one of two ways.

You can operate in pass-through mode and add an image directly into your [Doc](#) object. The [Doc.AddImageFile](#), [Doc.AddImageData](#) operate in pass-through mode.

Alternatively you can operate in indirect mode and draw your data into an [Image](#) object before adding the Image object to your document. The [Doc.AddImageObject](#) method defaults to indirect mode.

Indirect mode has a number of advantages over

pass-through mode.

Because each image is fully decoded when it is presented to the Image object, image corruption can be caught at this stage. Corrupt images are not uncommon and if you don't detect the corruption before the data is inserted you may end up with a corrupt PDF.

---

#### Modes

Images are color corrected which means that color profiles do not need to be embedded and file size may be reduced.

Pass-through mode has advantages for some types of images.

Because the image may not have to be decompressed and re-compressed, using pass-through mode can be much faster than indirect mode. This is particularly true for scanned TIFF images.

Because the original image compression is maintained there is no possibility of expansion due to re-compression.

Images are inserted in their native color space together with any color profiles - this guarantees fidelity of color reproduction.

Colors can be specified in a number of ways - most commonly in terms of RGB or CMYK values. These values are literal - RGB values relate directly to the brightness of the red, green and blue phosphors on a display - CMYK values relate directly to the amounts of Cyan, Magenta,

Yellow and Black inks applied to a piece of paper.

These literal values do not always equate to the same perceived color. Monitors vary and an image displayed on one may look quite different to the same image displayed on another.

Similarly a CMYK value applied with one printer to one type of paper may look quite different to a CMYK value applied with a different printer to a different type of paper.

---

## Colors

The [International Color Consortium](#) (ICC) provides specifications to allow an independent definition of color. An ICC color profile is designed to complement your raw color data. So if you have a CMYK image you also specify an color profile which tells you more about the intended destination of the image. This allows you to adjust the colors for your intended output medium.

In order to maintain flexibility and color fidelity it is important to preserve both the raw image data and the ICC profile associated with it. This can only be done if you provide images in pass-through rather than direct mode.

ABCpdf will accept RGB TIFF, CMYK TIFF, LAB TIFF, Grayscale JPEG, RGB JPEG and CMYK JPEG direct. This means whatever colors you pass into ABCpdf will go direct into the PDF without any modification. Additionally any ICC profile will be preserved and inserted appropriately so that the colors can be adjusted accurately for any output.

So what types of compression does the [Image](#) object use?

The Image object generally uses Flate compression. This is the same compression type as is used for PNG images. It is a lossless method which ensures that the quality of your original image is maintained.

---

**Comp**

However if your source image is black and white then ABCpdf will use CCITT G4 fax compression as this typically results in reduced file sizes for this type of image.



## HTML Styled Text

---

ABCpdf allows a range of HTML support for use when inserting multi-styled text. This can make it much easier to design documents and it can reduce the quantity of code required.

---

### Basics

The HTML support offered here does not cover the entire HTML specification. It covers a limited range of the HTML specification as is required for styled text. It also extends the HTML specification to allow you to precisely control elements of style not covered by HTML.

ABCpdf lets you use Unicode text. So you can use HTML Styled Text with any language from English to Korean to Japanese.

Normal character entities are standard HTML and hence use the Latin 1 character set. For example '&#153;' equates to the trademark sign.

For convenience you can also specify hex and octal character entities. For example '&#65;', '&#x0041;', and '&#o0101;' all equate to lower case 'a'. Hex and octal character entities are assumed to be expressed in the Unicode character set.

---

### Chars

For most situations this will make no difference. However you should note that, above character

127, there are differences between the Latin 1 and the Unicode character set. For example the trademark symbol which is character 153 in Latin 1 is 8482 in Unicode.

If you require fine control over hyphenation you can make use of the soft hyphen character – '&shy;'. This character is invisible and indicates a point at which a chunk of text may reasonably be broken.

You can add HTML to your documents using the [AddHtml](#) method. ABCpdf supports the following HTML tags and Attributes.

- <Head>
- <Body>
- <BR>
- <P>
- <H1> to <H6>
- <List>
- <UL>
- <OL>
- <LI>
- <A>
- <B>
- <I>
- <U>
- <Strike>
- <Sup>
- <Sub>
- <Font>
- <StyleRun>
- <BlockQuote>
- <Pre>

---

Tags

- <Leader>

## <Head>

This tag is used to delimit the head of an HTML document. All content in the head is ignored.

This tag does not accept any attributes.

## <Body>

This tag is used to mark the body of an HTML document. The body is a type of stylerun and so it accepts the same attributes as a [stylerun](#) tag. It also accepts the following additional attributes.

Attribute	Notes
link	<p>The color for links (anchors) in subsequent content.</p> <p>Colors are generally specified as RGB in hexadecimal notation (e.g. color="#FF0000") or as one of the sixteen standard color names (e.g. color=red").</p> <p>You can specify grayscale colors by supplying only one component (e.g. color="#80") and CMYK colors by supplying four (e.g. color="#10203040"). CMYK component ranges between 0 and 100 inclusively so the hexadecimal representation is between 00 and 64.</p> <p>You can specify high precision colors by passing an array of floating point numbers prepended by an at sign. Each number represents a component intensity in the range zero to one (e.g. color="@ 0.244 0.122 0.342"). The number of components indicates whether the color space is grayscale, RGB or CMYK (e.g. color="@ 0.123 0.246 0.999 0.025" would be CMYK). An alpha value can be indicated by prepending an 'a' to one of the components.</p> <p>The default link color is RGB blue.</p>

`<BR>`

This tag is used to force a line break.

This tag does not accept any attributes.

## <P>

This tag is used to mark up paragraphs. Paragraphs are types of `styleruns` and so they accept the same attributes as `stylerun` tags. They also accept the following additional attributes.

Attribute	Notes
<code>align</code>	<p>The text alignment for the paragraph. If no value is specified it is assumed that the text should be left aligned.</p> <p>You can specify left aligned text (e.g. <code>align=left</code>), right aligned text (e.g. <code>align=right</code>), centered text (e.g. <code>align=center</code>) or justified text (e.g. <code>align=justify</code>).</p> <p>Changing this attribute is identical to changing the <code>XTextStyle.HPos</code> or <code>XTextStyle.Justification</code> properties.</p>
<code>break</code>	<p>The line breaking style for the paragraph. This attribute takes a comma delimited list of hints used to control how lines are broken when chaining from one text area to another.</p> <p>You can specify that a paragraph should be kept with the next one by assigning the <code>keepwithnext</code> hint (e.g. <code>break=keepwithnext</code>). You can specify that there should be a break before a paragraph by specifying the <code>breakbefore</code> hint (e.g. <code>break=breakbefore</code>).</p> <p>For details on chaining see the <code>AddHtml</code> function.</p>
<code>spacebefore</code>	<p>Normally paragraphs have vertical space before and after the body text.</p>

	<p>By setting this attribute to zero you can remove vertical space before the paragraph.</p>
spaceafter	<p>Normally paragraphs have vertical space before and after the body text.</p> <p>By setting this attribute to zero you can remove vertical space after the paragraph.</p>

## <H1> to <H6>

This tag is used to mark header sections. Headers are types of paragraphs and so they accept the same attributes as [paragraph](#) tags.

## <List>

This tag is used to indicate a list of items. Each list item consists of a marker and some text. Markers may be bullet points, numbers or letters.

Lists are types of paragraphs and so they accept the same attributes as [paragraph](#) tags. They also accept the following additional attributes.

Attribute	Notes
itemindent	<p>The indent of the item text from the left of the marker. This value is measured in the current units.</p> <p>By altering this property you can change the distance between the marker and the text.</p> <p>The default is dynamically determined based on the type of marker and the size of the text.</p>
markerindent	<p>The indent of the left of the marker from the current left of the surrounding text. This value is measured in the current units.</p> <p>By altering this property you can alter the indent distance for the markers in the list.</p> <p>The default is dynamically determined based on the type of marker and the size of the text.</p>
start	<p>Specifies the starting number for the first item in the list.</p> <p>This is only used when ordered markers are specified.</p> <p>The default is one.</p>

type

Specifies the type of marker to use. You can use either ordered markers or unordered markers.

Ordered markers increment for each item in the list. You can use numbers (type=1), lower case roman numerals (type=i), upper case roman numerals (type=I), lower case letters (type='a'), upper case letters (type=A) or none (type=none).

Unordered markers are the same for each item in the list. You can specify bullet points (type=disk), hollow bullets (type=circle) or squares (type=square).

The default type is 'disk'.

## <UL>

The UL tag is used to indicate an unordered list. Unordered lists are types of lists and so they accept the same attributes as [list](#) tags. The default marker is the bullet point but the marker will change as lists are nested within each other.

## <OL>

The OL tag is used to indicate an ordered list. Ordered lists are types of lists and so they accept the same attributes as [list](#) tags. The default marker type is numeric.

## <LI>

This tag is used to indicate an item within a list. It accepts the following attributes.

Attribute	Notes
value	Specifies the number for this list item. Subsequent items are numbered incrementally from this new value.
type	Specifies the type of marker to use. You can use the same types as you find in the type attribute of the <a href="#">list</a> tag.

<A>

The anchor tag is used to mark subsequent content as a hyperlink. Anchors are types of styleruns and so they accept the same attributes as [stylerun](#) tags. They also accept the following additional attributes.

Attribute	Notes
href	The URL of the destination.

Hyperlinks normally appear as blue underlined text. You can override this style using the [body](#) link attribute or by using [stylerun](#) attributes in the body of your anchor tag.

**<B>**

This tag is used to apply a bold text style to subsequent content.

ABCpdf will attempt to reference an appropriate bold font. If it cannot locate one it will generate a synthetic bold style using the [XTextStyle.Bold](#) property.

This tag does not accept any attributes.



This tag is used to apply an italic text style to subsequent content.

ABCpdf will attempt to reference an appropriate italic font. If it cannot locate one it will generate a synthetic italic style using the [XTextStyle.Italic](#) property.

This tag does not accept any attributes.

<U>

This tag is used to apply an underline text style to subsequent content. The effect is identical to changing the `XTextStyle.Underline` property.

This tag does not accept any attributes.

## <Strike>

This tag is used to apply an strike-through text style to subsequent content. The effect is identical to changing the [XTextStyle.Strike](#) property.

This tag does not accept any attributes.

`<Sup>`

This tag is used to indicate text to be rendered as superscript.

This tag does not accept any attributes.

`<Sub>`

This tag is used to indicate text to be rendered as subscript.

This tag does not accept any attributes.

## <Font>

The font tag is used to change the current font style. Fonts are types of styleruns and so they accept the same attributes as [stylerun](#) tags. They also accept the following additional attributes.

Attribute	Notes
size	<p>The font size for subsequent content.</p> <p>You can set absolute font size by specifying an integer ranging from one to seven (e.g. size=6). Or you can specify a font size relative to the current base font size (e.g. size="+1").</p> <p>For greater control over the size of text you should use the <a href="#">fontsize</a> attribute.</p>
color	<p>The color for subsequent content.</p> <p>Colors are generally specified as RGB in hexadecimal notation (e.g. color="#FF0000") or as one of the sixteen standard color names (e.g. color=red").</p> <p>You can specify grayscale colors by supplying only one component (e.g. color="#80") and CMYK colors by supplying four (e.g. color="#10203040"). CMYK component ranges between 0 and 100 inclusively so the hexadecimal representation is between 00 and 64.</p> <p>You can specify an alpha value for your color by appending a slash and a hex value to the end of your color string (e.g. color="#10203040/C0").</p> <p>You can specify that no color should be applied by using the special 'none' keyword (e.g. color="none"). This can be</p>

	<p>useful if you wish to specify your own color operators in your own low level code.</p> <p>You can specify high precision colors by passing an array of floating point numbers prepended by an at sign. Each number represents a component intensity in the range zero to one (e.g. color="@ 0.244 0.122 0.342"). The number of components indicates whether the color space is grayscale, RGB or CMYK (e.g. color="@ 0.123 0.246 0.999 0.025" would be CMYK). An alpha value can be indicated by prepending an 'a' to one of the components.</p> <p>This attribute sets both the stroke and fill colors.</p>
color-stroke	<p>The stroke color for subsequent content.</p> <p>This attribute allows you to specify the stroke color independently from the fill color.</p>
color-fill	<p>The fill color for subsequent content.</p> <p>This attribute allows you to specify the fill color independently from the stroke color.</p>
csid	<p>The color space for subsequent content.</p> <p>This attribute takes an Object ID obtained from a previous call to <a href="#">AddColorSpaceFile</a> or <a href="#">AddColorSpaceSpot</a>.</p> <p>This attribute sets both the stroke and fill color space.</p>
csid-stroke	<p>The stroke color space for subsequent content.</p> <p>This attribute allows you to specify the stroke color space independently from the fill color space.</p>

csid-fill	<p>The fill color space for subsequent content.</p> <p>This attribute allows you to specify the fill color space independently from the stroke color space.</p>
face	<p>The font typeface for subsequent content.</p> <p>This value takes a comma delimited list of typeface names, listed in order of preference. Fonts are referenced rather than embedded.</p> <p>For greater control over the way that fonts are added to your document you should use the <a href="#">pid</a> attribute.</p>
embed	<p>Whether to embed or reference fonts added via the face attribute.</p> <p>For details see the <a href="#">EmbedFont</a> method. The default is false.</p>
language	<p>What language to use when embedding fonts added via the face attribute.</p> <p>For details see the <a href="#">EmbedFont</a> method. The default is Latin.</p>
protection	<p>Whether to apply font protection when embedding fonts added via the face attribute.</p> <p>For details see the <a href="#">EmbedFont</a> method. The default is true.</p>
font-family	<p>The font family for subsequent content.</p> <p>This value operates in the same way as the face attribute</p>

	<p>detailed above.</p>
font-style	<p>The font style.</p> <p>This attribute can take the values oblique, italic or normal. Normal is the default.</p> <p>NB. ABCpdf does not currently make a distinction between oblique and italic styles.</p>
font-weight	<p>The font weight.</p> <p>This attribute can take the a value between 100 (lightest) and 900 (heaviest).</p> <p>It can also take the following pre-defined values – normal, bold, bolder and lighter.</p> <p>The default is normal – 400.</p> <p>NB. ABCpdf cannot currently synthesize font weights of less than 400.</p>
rendering-mode	<p>The text rendering mode. Possible values are:</p> <ul style="list-style-type: none"> <li>0 Fill text (default)</li> <li>1 Stroke text.</li> <li>2 Fill, then stroke text.</li> <li>3 Neither fill nor stroke text (invisible).</li> <li>4 Fill text and add to path for clipping.</li> <li>5 Stroke text and add to path for clipping.</li> <li>6 Fill, then stroke text and add to path for clipping.</li> <li>7 Add text to path for clipping.</li> </ul> <p>NB. The outline style is a more commonly used alternative to the rendering-mode.</p>



## <StyleRun>

The stylerun tag is used to change the current style. It accepts the following attributes.

Attribute	Notes
pid	The font typeface for subsequent content. This attribute take an Object ID obtained from a previous call to <a href="#">AddFont</a> or <a href="#">EmbedFont</a> .
fontsize	The font size for subsequent content. Changing this attribute is identical to changing the <a href="#">XTextStyle.Size</a> property.
charspacing	The character spacing for subsequent content. Changing this attribute is identical to changing the <a href="#">XTextStyle.CharSpacing</a> property.
wordspacing	The word spacing for subsequent content. Changing this attribute is identical to changing the <a href="#">XTextStyle.WordSpacing</a> property.
justification	The justification for subsequent content. Changing this attribute is identical to changing the <a href="#">XTextStyle.Justification</a> property.

hpos	<p>The horizontal positioning for subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.HPos</a> property.</p>
bold	<p>Whether to apply a synthetic bold style to subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Bold</a> property.</p>
italic	<p>Whether to apply a synthetic italic style subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Italic</a> property.</p>
underline	<p>Whether to underline subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Underline</a> property.</p>
strike	<p>Whether to apply a strike-through effect to subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Strike</a> property.</p>
strike2	<p>Whether to apply a double strike-through effect to subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Strike2</a> property.</p>

outline	<p>Whether to outline subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Outline</a> property.</p>
linespacing	<p>The line spacing for subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.LineSpacing</a> property.</p>
paraspacing	<p>The paragraph spacing for subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.ParaSpacing</a> property.</p>
leftmargin	<p>The left margin for subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.LeftMargin</a> property.</p>
leftmargins	<p>Variable left margins for subsequent content.</p> <p>The leftmargin property applies one margin to all text in the range. The leftmargins property allows you to apply variable margins depending on how far down the page the text is being positioned. This can be used to flow text around objects such as pictures.</p> <p>This tag takes a an array of floating point numbers. The array must be a multiple of three long and each triplet defines a margin for a particular vertical range within the <a href="#">Doc.Rect</a>. The values are [yMin yMax Margin] where the y values are measured downwards from the top of the <a href="#">Doc.Rect</a>.</p>

	For typical code see the <a href="#">Text Flow Round Image</a> example.
rightmargin	<p>The right margin for subsequent content.</p> <p>This property is similar to the leftmargin attribute but allows you to specify a margin on the right hand side of the text block.</p>
rightmargins	<p>Variable right margins for subsequent content.</p> <p>This property is similar to the leftmargins attribute but allows you to specify margins on the right hand side of the text block.</p>
indent	<p>The indent for subsequent content.</p> <p>Changing this attribute is identical to changing the <a href="#">XTextStyle.Indent</a> property.</p>
fixedwidth	<p>A fixed width for the style run.</p> <p>Each style run has a width. The width is normally determined by the size of the characters in the text.</p> <p>Under some situations it can be useful to assign a fixed width to the entire style run. This can be used for aligning text and for bullet pointed lists.</p>
textrise	<p>The text rise for subsequent content.</p> <p>Positive values shift the text upwards. Negative values shift it downwards. The textrise distance is measured in the current units.</p>

annots	<p>Annotations associated with subsequent content.</p> <p>You can use a link annotation to insert a hyperlink (e.g. <code>annots='link:http://www.google.com/'</code>).</p> <p>You can use a goto annotation to insert a link to another page in the document (e.g. <code>annots='goto:3'</code>). The number indicates the page number. Note that the destination page must exist at the point at which the text is inserted.</p> <p>You can use a text annotation to insert a textual note (e.g. <code>annots='text:A note to be inserted'</code>).</p> <p>You can use highlight, squiggly, underline, and strikeouts annotations for text markup (e.g. <code>annots='highlight:some contents'</code>).</p> <p>While other types of annotations have textual contents (that are specified after colon), link and goto annotations do not. However, you can still mark link and goto annotations with (non-displayed) textual contents using "contents:" so that you can later identify them easily with <a href="#">Doc.GetInfo</a> or <a href="#">Annotation.Contents</a> (e.g. <code>annots='link:http://www.google.com/;contents:alternative description'</code>).</p>
	<p>The default reading direction.</p> <p>Bi-directional text such as Hebrew or Arabic is laid out in the context of the default reading direction.</p> <p>You can specify left to right paragraph direction (e.g. <code>dir=ltr</code>), right to left paragraph direction (e.g. <code>dir=rtl</code>), or use the default of none (e.g. <code>dir=none</code>).</p> <p>When none is specified the original ABCpdf left to right layout is preserved for compatibility with previous</p>

dir	<p>versions.</p> <p>Arabic requires special text shaping for contextual ligatures and combining characters. This is because each character has different forms depending on placement within a string. Each character can be independent, initial, medial or final. This text shaping is only performed if a reading direction is set. So for Arabic support you should always specify a default reading direction.</p>
canbreakafter	<p>Characters at which lines may be broken.</p> <p>Normally ABCpdf will only break lines after certain characters like spaces. You can indicate additional characters after which a break is acceptable using this parameter.</p> <p>For example to allow a break after hyphens or underscores you might use <code>canbreakafter='-_'</code>.</p>
breakengine	<p>The default line breaking engine.</p> <p>The line breaking engine determines at which points lines can be broken.</p> <p>You can specify the Uniscribe line breaking engine (e.g. <code>breakengine=uniscribe</code>), the Unicode line breaking engine (e.g. <code>breakengine=unicode</code>), or use the default of auto (e.g. <code>breakengine=auto</code>).</p> <p>When auto is specified the Uniscribe engine is used for installations on Windows XP and 2003 and the Unicode engine is used for installations on Windows NT and 2000.</p>

wrap	<p>Whether lines should wrap or not. The default is true.</p> <p>If wrapping is turned off (ie by setting this value to false) then lines of text extending outside the drawing area will be truncated.</p>
transform	<p>The transformation for text.</p> <p>The format of this value is the same as that of <a href="#">XTransform.String</a>. The default value is the identity. This attribute takes precedence over the rotate attribute. The numeric values can optionally be preceded by 'fixed' (e.g. transform='fixed 2 0 0 2 0 0'), which fixes the transformation origin.</p> <p>The direction to which the rightward direction (the default primary advancement direction) is mapped is the primary advancement direction. The direction in which a new line is offset (i.e. the direction of the offset caused by starting a new line) is the secondary advancement direction.</p> <p>When the origin is not fixed and the StyleRun is broken into pieces (because of nested tags or line breaks, for example), the transformation is applied to each piece individually. When the primary advancement direction is not horizontal, the effect is observably different as each piece starts at the original baseline.</p> <p>When the origin is fixed, each piece (without its own transformation) appears as if it starts at where the previous piece ends because the transformation origin is fixed at the beginning of the StyleRun.</p> <p>The secondary advancement direction is always perpendicular to the primary advancement direction regardless of skews in the transformation. (This is meaningful only in the context of multiple lines within a</p>

	<p>StyleRun with a fixed transformation.) In which of the two opposite directions the secondary advancement is depends on the direction to which the downward direction (the default secondary advancement direction) is mapped. This allows artificial styles that simulate oblique/italic without affecting the secondary advancement.</p> <p>Decorations (underline and strike-through) are placed at the correct locations without distortion so they remain rectangular.</p>
rotate	<p>The rotation for text.</p> <p>This value is measured in degrees anticlockwise. The default is zero. This attribute is ignored if the transform attribute is present. The numeric value can be preceded by 'fixed' (e.g. rotate='fixed 30'), which fixes the transformation origin as explained in the transform attribute.</p>
ascender	<p>The default size of the ascender.</p> <p>The PDF specification defines text as drawn from a point anchored at the baseline of a letter. So to place a chunk of text inside a box the text needs to be shifted downwards by the distance between the baseline and the top of the letter. This distance is typically known as the ascender height.</p> <p>ABCpdf uses its own methods to determine the ascender height for fonts. However using this attribute you can override these methods and specify your own values. This can be useful for situations in which you are drawing text anchored at the baseline rather than the top of the glyphs.</p>

The ascender value is measured in 1000ths of the font height. A typical value might be 800.

## <BlockQuote>

This tag is used to indicate quotations. Block quotes are indented on the left and right relative to the surrounding text. The tag accepts the following attributes.

Attribute	Notes
leftindent	The left indent for the block of text. Distances are measured in the current units. The default is 36 points.
rightindent	The right indent for the block of text. Distances are measured in the current units. The default is 36 points.

`<Pre>`

This tag is used to indicate preformatted text. Spaces and line breaks are preserved.

## <Leader>

This tag is used to create leaders for structures like a table of contents. A leader is a repeated character - most normally a period - which runs across the page from the heading on the left to the page number on the right.

The item to be repeated is placed between the start and end tag of the leaders element. For example code see the [XTextStyle.Kerning](#) property.

The tag accepts the following attributes.

Attribute	Notes
align	<p>Specifies whether the leaders should be horizontally aligned with each other.</p> <p>If the tag value is set to true then the individual items in the leader (typically period characters) will be aligned so that leaders on one line are horizontally aligned with the leaders on the next.</p> <p>If the tag value is set to false then the alignment of the leaders on a particular line will be determined by the width of the text which precedes the leaders..</p> <p>The default align is 'true'.</p>
	<p>Specifies the alignment for the leaders.</p> <p>The align and hpos tags are mutually incompatible. As such, specifying an hpos will automatically set the align to false.</p> <p>A block of leaders may not exactly fill the gap</p>

hpos

between the item of text on the left and the item of text on the right. This setting controls how the block is aligned within this gap. It works in the same way as the [XTextStyle.HPos](#) property. So to center your block of leaders between the two items of text you would set the value to 0.5.

By default the hpos is blank.



## Other Coordinate Spaces

---

ABCpdf uses the standard [PDF coordinate space](#).

The origin of this space is at the bottom left of the document. Distances are measured up and to the right in points.

Windows often uses a top-down coordinate space. The difference between the bottom-up nature of PDF and the top-down nature of Windows can result in confusion. Using the [AddGrid](#) method in your code can be very helpful in resolving this type of issue.

---

### Basics

Abstracting the coordinate space can make layout easier but removes you from the underlying page layout embedded into your PDF. For this reason we would suggest you stick to the standard PDF coordinate space where practical.

If you wish to use a different coordinate system you can use the document [Units](#) and [TopDown](#) properties or a [Transform](#) to accomplish this. There are subtle differences between these approaches.

The document [Units](#) and [TopDown](#) properties

allow you to use units of measurement like mm or cm. They also allow you to invert the coordinate system so that distances are measured down from the top rather than up from the bottom.

These properties enable an abstraction designed to make layout easier. However the underlying units of measurement and page layout remain the same.

When you call a page layout operation like [AddText](#) or [FrameRect](#) your abstracted coordinates are translated into PDF coordinates before the object is inserted into the document.

---

#### Units

For most layout tasks you can ignore the translation. However if you are using low-level access to the PDF structure or if you are using [Transforms](#) then you need to be aware of it.

Suppose you call [FrameRect](#) and then extract the raw content stream from the returned object. You must be aware that the base coordinates included in the stream are measured in the PDF coordinate space and not in your current coordinate system.

Document Transforms operate on the native PDF coordinate space. So when you specify a rotation around the origin - the origin specified is at the bottom left of the document - not the top.

**Transforms** are an inherent part of the PDF specification. They are not an abstraction imposed by ABCpdf but a set of drawing instructions inserted into the PDF document to modify the PDF drawing space. The fact that there is no abstraction can be very useful.

For example suppose you impose a scale transform so you can specify your page layout in mm rather than points. You then call **FrameRect** and get the raw content stream from the returned object. The coordinates included in the stream are measured in mm rather than points.

---

**Transform**

However transforms can be difficult to understand and they are extremely literal.

Suppose you impose a transform to flip, scale and translate your coordinate space so that distances are measured in mm from the top left rather than points from the bottom left.

Many graphical objects - lines and rectangles - will appear correctly. However the transform is literal and the flip aspect means that any images and text you add will appear upside down.

# Registry Keys

---

Some aspects of ABCpdf can be controlled on a global level using registry keys.

In general, you will not need to use these, and you should take them with a grain of salt. You should decide to insert them.

Registry settings are held at:

HKEY\_LOCAL\_MACHINE\SOFTWARE\WebSupergoo\ABCpdf

Except those listed as "registry only", they can also be specified in the command line. Please see [XSettings.SetConfigSection](#) for the syntax.

Name	Type	Default	Overridable
TempDirectory	STRING	""	
V3Compatible	DWORD	0 [false]	

V4Compatible	DWORD	0 [false]	
V5Compatible	DWORD	0 [false]	
MemoryManager	DWORD	1	
DisableCCITT	DWORD	0 [false]	

InsertJPEGsDirect	DWORD	1 [true]	<input checked="" type="checkbox"/>
InsertTIFFsDirect	DWORD	1 [true]	<input checked="" type="checkbox"/>

LogEvents	DWORD	1 [true]	
LogErrors	DWORD	0 [false]	
AutoRecover	DWORD	0 [false]	
LogOperations	DWORD	0 [false]	
MaxOperationTime	DWORD	1,000	
LogDirectory	STRING	"%SystemRoot%\Temp\"	
LogMaxFileSize	DWORD	0xFFFF	
LogAsserts	DWORD	1 [true]	
LogAssertsMax	DWORD	5	

ClearoutSize	DWORD	1,000	
LoadType1Fonts	DWORD	1 [true]	
WriteVersionNumber	DWORD	1 [true]	
SetInfoProps	DWORD	1 [true]	

CheckSaveRestore	DWORD	1 [true]	<input checked="" type="checkbox"/>



Preflight

DWORD

1



Refactor

DWORD

1



PagesMaxKids

DWORD

20



FontInheritsStyles	DWORD	0 [false]	<input type="checkbox"/>

SVGDrawInvisible	DWORD	0 [false]	<input type="checkbox"/>
AutorotateTIFFImages	DWORD	1 [true]	<input checked="" type="checkbox"/>

AutosizeImages	DWORD	0 [false]	
KerningType	DWORD	1	

EncryptCompat

DWORD

0



## Annotation and Field

Name	Type	Default	Overridable	Description
AnnotBorder	DWORD	0	 SetInfo only	<p>The default width of the borders on Annotations applied to PDF documents.</p> <p>This property is only accessible using the Doc.SetInfo method. The default cannot be overridden using a registry key.</p>
NeverClipAppearances	DWORD	1 [true]		<p>This parameter allows control over the way that text fields appear when they are generated.</p> <p>Some documents contain text fields which are not large enough for the text that they contain. There are two choices here. You can preserve the requested text and let the text be clipped. Alternatively, you can reduce the size so that it</p>

			<p>the height of the field.</p> <p>The default behavior is to reduce the text size. However, in some cases you may want to override this behavior by a different setting.</p>
			<p>In general, AB will try to avoid updating field appearances. This is because different applications have different ways of generating field appearances, so any update is likely to cause visual changes.</p> <p>These changes are likely to be subtle - involving pixel-level movements and perhaps changes in text wrapping. However, because there is no specification relating to how appearances</p>

ForceUpdateAppearances DWORD 0 [false]

should be generated the kind of subtle are inevitable regenerate appearance streams.

Acrobat will a regenerate file appearance streams for all documents in the NeedsAppear flag is set. Th comes at the of the kind of shifts detailed above. However because this done interacti this doesn't te matter very m

Some PDF producers cre PDF document containing inc appearance streams. Bec: Acrobat regenerates t field appeara streams it will display these correctly. ABC will assume th because they

				<p>there they are correct and w</p> <p>In this case yo may wish to n the behavior of Acrobat force update of the appearance streams. This allows you to specify that th should be the</p>
SignatureSizeMultiplier	DWORD	2	•	<p>This can affect size of buffers ABCpdf will allocate for th /Contents field signatures.</p> <p>This is a low l setting. If ABC encounters er related to buff size not big er to hold signed messages, yo manually incre this multiplier.</p>

## HTML/EMF Rendering

Name	Type	Default	Overridable	Description
RenderDelay	DWORD	0	<input checked="" type="checkbox"/>	<p>HTML render stages: an in a render stag</p> <p>For backward can insert a c stages. This milliseconds.</p>
AutoDeleteHTML	DWORD	1 [true]	<input type="checkbox"/>	<p>Certain HTML deleted by Al</p> <p>You are unlik value of this</p>
OneStageRender	DWORD	1 [true]	<input checked="" type="checkbox"/>	<p>This property compatibility</p>
LargeDocumentSize	DWORD	2,000	<input checked="" type="checkbox"/>	<p>The size at w constructed f</p> <p>For larger do is a much mo rendering the</p> <p>For smaller c build the tree advantages s better.</p> <p>This property ABCpdf shift:</p>

				<p>binary search number of objects. The number of objects is extremely dependent on the page, but fragments per page are atypical.</p>
DisableFilters	DWORD	1 [true]	<p>● SetInfo only</p>	<p>Whether to apply filters.</p> <p>HTML filters PDF conversion raster rather than vector reason, we try to keep the image as small as possible.</p> <p>This property is set in Doc.SetInfo and can be overridden using the following methods:</p>
JPEGThreshold	DWORD	256	<p>● SetInfo only</p>	<p>The threshold for compressed images.</p> <p>When an uncached image is used in a document, the image is converted to use JPEG lossless compression if the size of the image is less than the threshold.</p> <p>ABCpdf converts images in the image to use JPEG lossless compression if the size of the image is less than this threshold. The image is eligible for JPEG lossless compression if the size of the image is less than the threshold for color images.</p>

				This property Doc.SetInfo is overridden us
UnloadInterval	DWORD	0	● SetInfo only	<p>Whether we when the scr</p> <p>Appropriate v never unload little while aft change).</p> <p>The number of calls that A change has c MSHTML for some point d unload the D</p> <p>This property Doc.SetInfo is overridden us</p>
MakeURLsUnique	DWORD	1 [true]	●	<p>Whether to n page cache i with a 7-rand and a 7-rand added to the that contain c not end with</p> <p>Sometimes c ABCpdf. Mak provide an ef to be refresh</p>

BgImageVertProximity	DWORD	480	●	<p>ABCpdf treat images differently generally across image across acceptable to this way.</p> <p>For this reason whether image background</p> <p>If the same in times within a ABCpdf will a image.</p> <p>This property distance (in p image in which images.</p>
BgImageMinRepeat	DWORD	5	●	<p>ABCpdf treat images differently generally across image across acceptable to this way.</p> <p>For this reason whether image background</p> <p>If the same in times within a ABCpdf will a image.</p> <p>This property</p>

				repeats above that the image
NarrowBgImageMaxWidth	DWORD	80	●	<p>ABCpdf treats images differently, generally across image across acceptable to this way.</p> <p>For this reason, whether image background is</p> <p>If an image is assume it is a</p> <p>This property width for this</p>
NarrowBgImageMinHeight	DWORD	480	●	<p>ABCpdf treats images differently, generally across image across acceptable to this way.</p> <p>For this reason, whether image background is</p> <p>If an image is assume it is a</p> <p>This property height for this</p>

CheckBgImages	DWORD	0 [false]	●	<p>Sometimes, when an HTML page has a background image, it may not load properly during page loading.</p> <p>This option allows you to specify whether or not the background image should be loaded. For example, if you are using ABCpdf to generate PDFs, you may want to set this option to false so that the background images are not loaded.</p>
HTMLBottomExtraMargin	DWORD	30	●	<p>Change this property to specify the extra margin that is added to the bottom of the page when rendering. This is useful when rendering is necessary to make the page taller. This extra margin is added to the bottom of the page so it does not get cut off.</p> <p>However, if the page is positioned relative to the top of the page, the object will not be cut off. Set this property to a value in order to specify the extra margin.</p>
HostWebBrowser	DWORD	1 [true]	●	<p>The initial value of the <a href="#">XHtmlOptions</a> property.</p>
URLPolicies	STRING	"0x1608		<p>This setting is used to specify the default URL for the application. It is your machine's default security key.</p> <p>The string is a hexadecimal string (such as "0x1608") and policies are defined by the string.</p>

		0x03"		<p>URLPOLICY delimited list. type of action on the Microsoft value for this tags.</p> <p>Treat with caution settings are t</p>
TrustedSites	STRING	""		<p>Cross domain permitted when This means that one domain will be unable widths and h</p> <p>Ideally to resolve should move domain or re frameset to e this type of c situations wh can specify s Any page UF trusted site U trusted status</p> <p>URL lists are example:</p> <p>http://www.google.com http://www.microsoft.com http://www.yahoo.com</p> <p>Treat with caution settings are t</p>

CharWidthRatioMin	DWORD	700	●	<p>When import using the MS specifies the character-wic</p> <p>Characters w form ratios b and CharWic assume the l substitution r widths form r placed as sp placed in an</p>
CharWidthRatioMax	DWORD	1,400	●	<p>When import using the MS specifies the character-wic</p> <p>Characters w form ratios b and the spec assume the l substitution r widths form r placed as sp placed in an</p>
				<p>When HTML movies are a ABCpdf will r movie.</p> <p>The preview don't have FI when printing</p>

FlashPreviewTime	DWORD	0	●	<p>you will see i viewer other</p> <p><u>This value is previous vers same meanin for SWF vect</u></p> <p>When the mo specify a tim movie will be</p> <p>Note that ma can alter the For example, current frame of content ha content you g to ten second content a use movie for ten</p>
FlashPreviewWaitTime	DWORD	2,000	●	<p>When HTML movies are a ABCpdf will r movie.</p> <p>The preview don't have FI when printing you will see i viewer other</p> <p>This property milliseconds) play before tl</p> <p>Note that ma</p>

				<p>can alter the For example, current frame of content ha content you g to ten second content a use movie for ten</p>
FlashPreviewDPI	DWORD	96	<ul style="list-style-type: none"> <li>●</li> </ul>	<p>When HTML movies are a ABCpdf will r movie.</p> <p>The preview don't have FL when printing you will see i viewer other</p> <p><u>This value is previous vers used for SWI</u></p> <p>This property dots per inch made.</p>
				<p>Changing the process is ru versions of M If the screen of the output right hand sic</p> <p>It is unusual because cha</p>

CheckDisplaySettings	DWORD	0		<p>not a normal access applic settings in ar means that s remotely can event in a lon web applicati</p> <p>This registry ABCpdf does detected. The</p> <ul style="list-style-type: none"> <li>• 0 - Ignor</li> <li>• 1 - Raise settings as a trig package</li> <li>• 2 - Atten previous only if th</li> </ul>
HtmlProcessPoolSize	DWORD	10		<p>When using t rendering, su engine or MS specifying ou maintains a p value specific such process</p> <p>The default v threaded env server. If you multiple threa rendering, yc save system</p>
				This setting c

MSHtmlBootstrap	DWORD	0	<ul style="list-style-type: none"> <li>•</li> </ul>	<p>ABCpdf uses</p> <ul style="list-style-type: none"> <li>• 0 - This registry) IE9 or al same as</li> <li>• 1 - MSH process, load IE8 achieve requires present</li> <li>• 2 - ABCj required without v be used User Pro registry.</li> <li>• 3 - ABCj configur hive if ec not in ef requires enabled</li> <li>• 4 - ABCj DLLs in- achieve requires present applicati MSHTM control) DLLs.</li> <li>• 5 - ABCj bootstra</li> </ul>

<p>AddLinksToIDs</p>	<p>DWORD</p>	<p>1 [true]</p>	<p><input checked="" type="checkbox"/></p>	<p>This is a sett up HTML ren AddLinks is s</p> <p>Fragment lin refer to anch So to use the to keep track can add a sig which contain</p> <p>If you are not method you c false. Indeed ABCpdf will s IDs in anchor overhead as elements.</p>
<p>AddImagePolicy</p>	<p>DWORD</p>	<p>0 [default]</p>	<p><input checked="" type="checkbox"/></p>	<p>This is a sett images in HT</p> <p>The default b directly as lo been specific quality and s maintained, t are inserted i transparency minimized.</p> <p>By setting thi force images HTML render decompress be recompre</p>

MSHtmlOverrideKeyPath	STRING	""	<p>The registry location is HKEY_CURRENT_USER\Software\Microsoft\MSHTML\Options. The value returned through IDocHostUIFactory::GetOptions() is the path to the registry key in HKEY_CURRENT_USER\Software\Microsoft\MSHTML\Options. The override value is "Software\Microsoft\MSHTML\Options". While MSHTML uses both HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE, the path specified by the override is HKEY_CURRENT_USER.</p> <p>If this setting is not set, the registry will write in the path "Software\Microsoft\MSHTML\Options" when bootstrapping.</p>
XULRunnerDirectory	STRING	""	<p>The path to the XULRunner engine.</p>
GeckoPrefFile	STRING	""	<p>The path to the preferences file for the <a href="#">XHtmlOptions</a>.</p>
UseCSSClassToAddTags	DWORD	1 [true]	<p>When the Gecko setting specified by the preference 'abcpdf-tag-visible' (XHtmlOptions::abcpdf-tag-visible) is deprecated, the value 'visible: true' is used.</p>

StopSMILAnimation	DWORD	1 [true]	●	When the Ge setting speci stopped. Wh the first fram
RelCharShiftTolerance	DWORD	60	●	This value is When the Ge specifies the shift between the glyphs to text. Because widths and p may be differ Glyphs in a c spaced in the
AbsCharShiftTolerance	DWORD	500	●	This value is is 1/72 of an is used, this absolute hori glyphs to cor continuous fl hinting, char reported by C those in the f flow are ever

## Rendering

Name	Type	Default	Overridable	Description
RenderNoBitmap	DWORD	0 [false]	 SetInfo only	<p>This parameter allows low-level control over the way that documents are rendered to vector formats like EMF and EPS.</p> <p>During the generation of an EMF or EPS file, a bitmap is generated due to transparency detection will be suppressed. This can cause some unexpected results as neither EMF or EPS format supports transparency at non-default blending modes to the extent that PDF does.</p> <p>This property is only accessible using the Doc.SetInfo method. The default cannot be overridden using a registry key.</p>

RenderNoDisplayList

DWORD

0  
[false]

●  
SetInfo only

This parameter allows low-level control over the way that documents are rendered to bitmap outputs.

During the generation of a high-resolution scene-antialias bitmap, the renderer stores PDF rendering objects in a display list. The bitmap is broken into sections and the display list is rendered to each section.

For some very complex PDF files, this display list can become rather large and consume a lot of memory. In these cases, it is better to allocate a complete image and render the objects directly to the image than to try and store individual PDF objects.

This property is only accessible

				using the Doc.SetInfo method. The de cannot be overridden using registry key.
RenderTextAsText	DWORD	1 [true]	● SetInfo only	<p>This parameter allows low-level control over the way that text is rendered into v formats like EP</p> <p>If the value is tr the text is place directly into the output file when possible. If it is false, the text is converted to vectors, which a then output to t file.</p> <p>The former met results in small files. The latter results in larger font-independent output files. It is important to not that not everyth that looks like t is actually text c can be extracte text.</p>

				<p>This property is only accessible using the Doc.SetInfo method. The default cannot be overridden using registry key.</p>
RenderEmfType	DWORD	1	<p>SetInfo only</p>	<p>This parameter allows low-level control over the way that documents are rendered to EMF.</p> <p>This setting controls the type of EMF output. Possible values are:</p> <ul style="list-style-type: none"> <li>• 0 - EmfOnly</li> <li>• 1 - EmfPlusOrEmfOnly</li> <li>• 2 - EmfPlusDual</li> </ul> <p>You may want to change the setting to EmfPlusDual which means that both EMF and EMF+ entries are included in the output. This produces greater compatibility but at the cost of a</p>

			<p>doubling in file :</p> <p>This property is only accessible using the <code>Doc.SetInfo</code> method. The default value cannot be overridden using registry key.</p>
			<p>This registry key allows you to control the installation of embedded fonts when rendering EMF.</p> <p>By default, when you use <code>Rendering.GetFont</code> to render to EMF fonts will be installed in memory and will remain available to the ABCpdf process until <code>Doc.Clear</code> is called. This means that the EMF can be spooled to a printer and the fonts will appear correctly.</p> <p>By default, when you use</p>

RenderInstallFonts

DWORD

0

Rendering. Save render to EMF, embedded fonts not installed and text will be rendered as polygons for fonts that are not available on the system. This is because when EMF file is viewed outside the ABC process, memory resident fonts are not available. Therefore any text using these fonts would be subject to font substitution by Windows.

Set this value to 0xFFFFFFFF to disable the installation of embedded fonts in memory, even when calling Rendering.GetFont. This results in text rendered with polygons for embedded fonts. Hence, the rendering is slow and the output is bigger.

			<p>Set this value to always install fonts, even when calling Rendering.Save. This results in faster rendering and smaller outputs. However, viewing emf file outside the ABC process may not display embedded fonts correctly.</p>
RenderBypassDeviceIcc	DWORD	0	<p>Some PDFs contain color profiles which do not handle bit objects correctly.</p> <p>This option allows you to bypass the profile transformation in input color space. If the input is CMYK ICC and the output is device CMYK, or if the input is Gray ICC based and the output is device CMYK.</p>
			<p>When an EMF is created the standard settings are taken from</p>

EmfDeviceName	STRING	""	SetInfo only	<p>default screen.</p> <p>Under some circumstances it can be desirable to allow these settings to be taken from different screens, indeed a device such as a printer.</p> <p>This property allows you to specify the name of the device to be used for the default EMF settings.</p> <p>This property is only accessible using the Doc.SetInfo method. The device cannot be overridden using registry key.</p>
RenderNoAAClipping	DWORD	1		<p>Disabled by default. Certain non-rectangular clip paths currently require that we keep these as alpha masks. For large files or large images to be generated, this result in increased</p>

			memory usage slower rendering speeds. Set this '0' to enable the bit clipping masks. Otherwise the clipping masks be represented bit masks (no antialiasing).
RenderCurveFlatness	STRING	"1"	When rendering stroked curves, curve is broken a series of line segments. The flatness value is default value for this tessellation, default is '1' which means that when the curve has deviated by 1 pixel it is broken into line segment. Note that even though registry representation is type 'STRING', will be converted into a DOUBLE numerical value.

XPS/HTML Export				
Name	Type	Default	Overridable	Description
				<p>This hint flag controls ABCpdf policy regarding the use of <a href="#">Windows Media Photo (WMP/JPEG XR)</a> images in converted XPS documents. If this flag is set, ABCpdf would prefer to use WMP over PNG in some situations.</p> <p>WMP is a versatile format that can handle lossless and lossy compression as well as transparency. It can often reduce the file size of produced XPS documents.</p>

XPSExportWMPImages DWORD 1 [true]



However, some compact .NET runtime environments, such as Silverlight, do not support decoding WMP images. If your XPS documents need to be used in those environments, you may want to turn this flag off.

Images in color-spaces such as CMYK must always be represented as WMP in XPS documents. As such, if you wish to ensure that your documents never use WMP, they must be converted to RGB or Grayscale

				before export.
--	--	--	--	----------------

## XpsAny Import

Name	Type	Default	Overridable	Description
PrintHookLog	DWORD or STRING	0	registry only	<p>Set this value to 1 to enable the logging of the PrintHook library using %SystemRoot%\Temp or set it to a path to a folder to enable the logging using that folder.</p> <p>The PrintHook library is used to print files to XPS, which are then converted to PDF in ImportAny operations. We may ask you to enable the logging and send us the log files for debugging purposes if you request us to solve related problems.</p> <p>Log files are named spy_&lt;PID&gt;.txt.</p> <p>On 64-bit machines, you will need to set this value in the 32-bit registry section (Wow6432Node) as well for intercepting 32-bit processes.</p>
				Set this value to 1 for

PrintHookShow	DWORD	0	<p>viewing applications that are launched by <code>Operations.ImportAny()</code>, typically Office applications. Set it to 0 or remove it to hide them.</p> <p>We use Office applications to print their documents to XPS files, which we then convert to PDF. Sometimes these applications get stuck on pop-ups of various types, so it may be useful to see what they are doing.</p> <p>Please note that there are limitations with this feature—it may not be possible to view applications that run under a different user account. This is normally the case when running Office Applications in a web server process.</p>
---------------	-------	---	---

## MSSOffice Import

Name	Type	Default	Overridable	Description
MSSOfficeShow	DWORD	0 [false]		Setting this to true will cause MSSOffice application windows to be displayed when using the MSSOffice read module. This can be useful for troubleshooting purposes.
PreferDCOMLaunch	DWORD	0 [false]		<p>This is a setting for the MSSOffice read module.</p> <p>Set to true to prefer using Windows' DCOMLaunch service to create MSSOffice application processes. This is mainly needed for using the MSSOffice read module in Classic ASP. Please refer to our support page for more instructions to prepare the</p>

				system for the MSOffice read module in Classic ASP.
--	--	--	--	--

## WordGlue Import

Name	Type	Default	Overridable	Description
WordGlueAssemblyPath	STRING	""		<p>If WordGlue .NET is installed, then ABCpdf will use it for importing DOC and DOCX content.</p> <p>ABCpdf will look in standard locations for WordGlue. However, if you have WordGlue installed in an unusual location, you may wish to let ABCpdf know where to look using this setting.</p>





# Troubleshooting

---

We don't expect you to have any problems with our software but if you do we offer a full level of support as standard. If you need an extra level of security we offer a [Platinum Level Support](#) program offering a priority channel through to our engineers.

---

## Intro

Your first port of call should be to check out the [FAQ on our website](#). If it doesn't cover your particular query you can mail us via the email addresses listed there. We prefer queries via email as technical issues are often complicated and require some thought on our side as well as yours. However if you need to telephone then numbers are listed on our web site.

Most bug reports relate to misunderstandings so if you're reporting a bug please do first check the documentation carefully. If you feel that an issue hasn't been explained well please do tell us so we can correct it.

Most configuration problems relate to security settings. If you are operating under a restricted security account please try running your code in a standard application while logged on as Administrator. It will only take you a few minutes to do this and it will instantly tell you if you have a security problem or not.

---

## Common

Because ABCpdf typically renders URLs from the

server-side the server itself may impose limits. For example if your server has a video card with a limited range of colors then certain graphics operations may be limited to that number of colors.

If you find anything that doesn't seem right please say. We don't know unless you tell us.

Remember to tell us what Operating System you're on and what version of ABCpdf you're using. You can get the exact version of ABCpdf from the PDFSettings control panel. If you're using an older version of ABCpdf we will be asking you if the problem occurs using the current download so you may wish to check our downloads page.

If the problem involves URL rendering please tell us exactly what version of Internet Explorer you have installed on your server. You can get the exact version of Internet Explorer from the About Internet Explorer menu item. If you're using an older version of Internet Explorer we will be asking you if the problem occurs using the current version.

Please give us a precise description of what's gone wrong. We like to see examples so if you have reasonably sized input or output documents which illustrate your point please do feel welcome to mail them to us.

If the problem relates to a particular resource which we may not have access to - maybe a font or an image - please do send these to us so we can find out what may be unusual about them. If you think we might like to see your code please show it to us (but please don't send your entire project).

If something's working partially please tell us what parts are working and what parts are not. If you have any theories or suspicions about what's going on please do tell us. Also tell us about any unusual software or configuration that might be relevant.

The fastest and most reliable way to resolve an issue is to provide a standalone code sample which demonstrates a problem. Please provide input and output resources and tell us what you think is wrong with your output.

A code sample should be small and should illustrate one particular problem. Please do not send us your entire project and please do not send us code we can't run because it's tied to your particular setup.

---

**Ideally**

Please put a little bit of effort into defining exactly what the problem is. In the long run this will save you time.

Once we have a standalone code example we can tell you exactly why your code isn't doing what you think it should. If it's a problem with our software we'll fix it or suggest a workaround. Even if the problem is outside our software we are often able to suggest workarounds.



## Corrupt Documents

---

Occasionally people may supply you with corrupt or non-standard PDF documents.

When you write code to draw on these supplied documents the drawing may appear in the wrong place or the wrong size or even not at all.

The most common cause of this type of issue is that the document MediaBox is non-standard. For example designers using Adobe Illustrator can easily manipulate the page MediaBox to produce - but unusual - documents.

---

Intro

You should start by using [AddGrid](#) as this should give you an idea of the way in which your drawing is failing.

Most PDFs define page size using a MediaBox anchored at the origin. For example "0 0 612 792".

However it is not required that you do this and some PDF documents define a MediaBox that is correct but unusual. For example "-612 -792 0 0".

You can check if this is the case by reporting the [MediaBox](#) property after the [Read](#) method has been called.

---

MediaBox

If you find this kind of non-standard MediaBox you are probably drawing your content off the edge of

the page.

You will need to adjust your drawing so that it occurs within the bounds of the document MediaBox.

The MediaBox defines the area of a page - the actual media size. However there are other measures as well.

The CropBox defines "the visible region of default user space. When the page is displayed or printed, its contents are to be clipped (cropped) to this rectangle and then imposed on the output medium".

When you display your PDF in Acrobat what you see is the area of the page defined by the CropBox rather than the MediaBox.

---

### CropBox

Normally CropBoxes and MediaBoxes are identical or practically identical. However occasionally you may find that they're not.

This may cause your output to appear in a different location from the one you are expecting.

Indeed your PDF may also impose a TrimBox or a BleedBox. If you're curious about these there are full details in the [Adobe PDF Specification](#).



## HTML / CSS Rendering

---

ABCpdf fully supports HTML and CSS.

You can render individual pages of HTML using the [AddImageUrl](#) method.

---

### Intro

You can page HTML over multiple PDF pages using the [AddImageUrl](#) method in combination with the [AddImageToChain](#) method.

ABCpdf allows you to treat HTML like any other media so you can even page your HTML across multiple columns of multiple pages of your PDF.

HTML was designed to specify the meaning of document content and leave the precise rendering and layout up to the browser. PDF was designed to specify the appearance of a document and ignore the meaning of the document content. HTML and PDF are fundamentally different.

HTML is being changed to allow greater control over the appearance of a document and PDF is being changed to allow the meaning of a document to be better represented. However, the fact that the two specifications are based on diametrically opposed concepts does mean that it can be difficult to convert between the two.

---

### Method

ABCpdf can use the MSHTML engine (used in Microsoft Internet Explorer) or the Gecko engine

(used in Mozilla Firefox) to parse and preprocess the HTML for insertion into your PDF. This provides an extremely accurate rendition of the HTML. Due to the differences in behavior and capabilities of the underlying rendering engine, you should expect differences in the rendered output when switching HTML engine. Please refer to [Engine](#), [ForMSHtml](#), and [ForGecko](#) for further elaborations on the engines' distinct characteristics.

ABCpdf holds a cache of recently requested URLs and it's only after five minutes or so that these pages expire from the cache.

This results in a considerable degree of optimization for many common operations. However, if you wish to bypass the cache, you can do so by setting the `DisableCache` parameter to true when you call `AddImageUrl` or `AddImageHtml`.

Occasionally, you may find that your page is being cached elsewhere. There are all kinds of places this can happen. For example, Windows sometimes caches individual page resources. Proxy servers may cache entire pages.

The standard reason that content gets cached is that pages are sending HTTP header information which indicates that it is acceptable to cache this content. If you are using the Internet Explorer HTML engine, sometimes it will insist to cache certain Web pages. In that case, your first step should be to use a tool like IEWatch to view the content expiration headers. Indeed, you may find that simply adjusting the content expiration settings found in the IIS Management Console will resolve the issue.

---

Cache

If you want to be totally sure that your URLs are rendered afresh each time, you need to vary the URL. For example:

`http://www.microsoft.com/?dummy=1`

`http://www.microsoft.com/?dummy=2`

`http://www.microsoft.com/?dummy=3`

These will all render the same page (`www.microsoft.com`) but because the URL is varying, you can be sure that they will be rendered afresh each time.

Obvious things will impact the speed of HTML conversion. So if you want to optimize the process look at retrieval times for your http requests, the size of your HTML and any related resources, the complexity of HTML, the speed of your computer. Tweaking these can make a big difference.

However there are also some small and simple things you may be able to do without getting into the complexity of system wide optimization.

The MSHTML rendering engine is, by default, set up for accuracy and quality. In order to ensure that the output is always good we have to enable every setting that might ever affect the output quality. This is the case even for situations in which you are not using those features.

So if your HTML does not contain features which require these settings then you can disable them. Doing so can result in significant speed improvements.

---

## Speed

The setting which typically makes the biggest difference is HostWebBrowser but DoMarkup and AdjustLayout are also worth looking at. The actual speed increases depend very much on the input HTML, but in our tests, disabling these features for simple HTML, increased the speed of processing by about 30% for HostWebBrowser, another 10% for the DoMarkup property and another 7% for the AdjustLayout property.

Another property which needs examination is the UseScript one. By default this is set to false but many people enable it in their ABCpdf code. As long as your JavaScript is good and sensible then there is no problem. However JavaScript is often coded poorly and as such it may have an unpredictable effect on speed. Consider disabling this feature if you do not actively need it.

Setting the BrowserWidth to a predefined value means that ABCpdf does not have to compute one. This can result in an increase of speed or perhaps 10% or so.

You can render any page you can supply a URL for.

When you render a page the page has to be reloaded by ABCpdf. This is because you - as a client - are looking at the page from your current machine. ABCpdf lives on the server and so it exists in a different session.

So, you cannot generally rely on cookies, session state or form submission in your page. The page must be reliant only on the URL you supply.

---

## Caveats

If you have to rely on session state, you could use cookie-less sessions (which will give you a URL for your session) or you could save the session information under a specific unique ID then pass the ID via the URL and pick up the information via your server-side code.

Problems which appear to be related to SSL or HTTPS connections are often authentication issues simply solved by providing a user name and password. See the [LogonName](#) property for details.

Screen resolution is typically 96 DPI. So, when you view an HTML page on your monitor, Windows will display it at 96 DPI.

The disparity between the screen resolution and the PDF 72 DPI resolution means that HTML appears larger in print documents than it does on screen.

---

## Size

You will need to apply a scale of  $72/96$  (0.75) to compensate for this if you want both to appear the same size.

For example, if you are rendering a web page supplying a value of 800 for the Width parameter, you will need to set the width of your Rect to 600 if you want both to appear the same size.

PDF documents are predominantly vector based. As such, they do not really have a DPI because they are resolution independent. The only portions of PDFs which are raster based are images.

Most elements of HTML - text, lines - are vector based. So, they are resolution independent.

---

DPI

The resolution at which images in your web pages are rendered is complicated. Suppose you have a 300 square image referenced by an image tag. If the width of your Doc.Rect is the same as the width you pass to AddImageUrl, this will be rendered at 72 DPI. However, by changing the ratios between these two values, the image will be scaled and hence the resolution will be changed.

And... if your 300 square is in an img tag with a width and height of 150, the default resolution will be doubled.

ABCpdf uses a sophisticated set of heuristics to determine where to break pages. For greater control over page breaking, you can use the page-break-before, page-break-after and page-break-inside CSS styles.

You must ensure that the element to which you apply your page breaking style is visible. For example:

```
<div style="page-break-before:always">&nbsp;</div>
```

... will break but ...

```
<div style="page-break-before:always"></div>
```

... will not.

---

## Breaks

### **Useful Tip.** Debugging page break styles.

Sometimes, your page breaks don't work in the way you think they should. Because these kinds of tags are invisible, it's very difficult for you to know whether you've applied them correctly or not. One simple solution is to debug your HTML using a visible style.

For example, when you apply your "page-break-inside: avoid" style, apply a right border style at the same time. That way, you can see exactly where your elements are. If the borders don't appear in the right places, then you know there's something wrong with your HTML.

The page break styles in the Gecko engine are not always applied as intuitively as they are in MSHTML. The root of this is the CSS specification that which says that break styles must be applicable to block-level elements within the "normal flow of the root element". It allows for these styles to be applied to other elements but does not mandate it.

The upshot of this, within the Gecko engine, is that page break styles cannot be applied within tables, to elements such as table rows. If you are unsure about whether something is likely to work just try Print Preview from within Firefox 38.0 as a simple sanity check.

You may wish to take a snapshot of the current URL.

In many circumstances, you should be able to derive a URL for the current page using the value of the `SERVER_NAME`, `URL` and `QUERY_STRING` Server Variables. You should be able to derive a URL for the previous page using the `HTTP_REFERER` (sic) Server Variable.

Alternatively, you can obtain the HTML of the current page using the `HttpResponse.Filter` property or by overriding the `Render` method of the page.

---

**Snapshot**

You can then present this HTML to ABCpdf using `AddImageHtml`. If your HTML references resources using relative references, you may wish to insert a `<BASE>` tag into the HTML before presentation to ABCpdf.

When you perform this kind of operation, be careful not to recursively call ABCpdf. If you do this, you will get into a hall-of-mirrors type situation and the software will not be able to return you a sensible image.



## SVG to PDF Import

---

When you use Doc.Read to import SVG content, ABCpdf uses its own native SVG import functionality. This is very fast and controllable and produces a very direct rendition of the SVG structures in terms of the PDF output. For SVG Tiny compliant input it is the best option.

---

### Intro

The alternative is to use AddImageUrl/Html with the Gecko engine. Because this uses the Firefox SVG rendering engine is extremely full featured. For complex SVG content including features outside the SVG Tiny specification it is the best option.

ABCpdf supports a subset of SVG based around the SVG Tiny specification.

It does not support all the features of SVG Tiny because some features do not easily map through to equivalents in the PDF format. However it supports some features outside SVG Tiny to cover common usage in real world (non SVG Tiny) documents.

---

### SVG Tiny

So what are the key differences between ABCpdf SVG and SVG Tiny?

PDF is essentially a static medium.

ABCpdf supports the import of static content.

---

Content

However it does not import dynamic SVG content such as animations, videos and scripts.

ABCpdf supports external fonts referenced in SVG.

However it is also possible to embed fonts in an SVG file using a set of tags defining the path for each glyph. ABCpdf does not support this type of embedded font.

Some text styles such as light weights and font variants are not easily represented in PDF. For this reason ABCpdf may approximate them if they are used.

---

Text

ABCpdf does not support the display-align property of the textarea element. However this is an element which is very rarely used (it is part of the new SVG 1.2 draft specification) and is not widely supported.

SVG Basic allows text in individual tspan blocks to be individually positioned attributes using x, y, dx and dy attributes. Because this is not part of SVG Tiny ABCpdf does not support it. However it is worth noting that this is perhaps the most commonly used text construct outside the specification.

Links are only supported for text elements within anchor elements.

---

**Links** Internal document links are not currently supported.

In SVG it is possible to specify opacity values for the stop-colors of gradients.

Unfortunately this is not possible in PDF.

---

**Gradients**

As such ABCpdf substitutes transparent stop-colors with brighter solid stop-colors.

Although it is not a part of SVG Tiny, ABCpdf supports a simplified CSS system.

This is useful because a large number of real world SVG documents use simple CSS selectors.

---

**CSS**

ABCpdf supports simple type selectors in internal style sheets. ABCpdf supports external style sheets as long as they are on the local file system.

However ABCpdf skips "import" and "media" CSS directives when parsing the style sheet itself.



## PDF to SVG Export

---

### Intro

When you use `Doc.Rendering.Save` to convert your PDF to SVG, ABCpdf produces a faithful rendition of the document. However for a variety of reasons the faithful rendition may not appear exactly as you expect. Here we explain why this is and how you can adapt the output to your needs.

PDF allows a complex set of options for sophisticated control over text display. While SVG contains many options for text layout, it does not support as wide a range as PDF.

PDF viewers are generally pretty good at supporting the text layout options supported by the PDF specification. Unfortunately the same cannot be said of SVG. Many SVG viewers ignore or misrepresent the more sophisticated text layout attributes supported by SVG.

The PDF format supports embedded fonts for high fidelity text reproduction. However the types of embedded fonts it supports are not the same as those supported by SVG and indeed subsetting may remove aspects that are crucial in the SVG representation.

### Text

For these reasons, if you require precise control over the way that your text appears when exporting to SVG, you should vectorize the page in question to convert any text into paths.

You can do this to the current page using the following code.

**[C#]**

```
((Page)doc.ObjectSoup[doc.Page]).VectorizeText()
```

## [Visual Basic]

```
DirectCast(doc.ObjectSoup(doc.Page),  
Page).VectorizeText()
```

Rendering to SVG produces placeholder tags for bitmap images in the SVG.

If you require that the images be exported, you should do so using code of the following form.

## [C#]

```
static void SaveAsSvg(Doc pdf, string file)  
{  
    pdf.Rendering.Save(file);  
    string svg = File.ReadAllText(file);  
    HashSet<string> hrefs = new  
HashSet<string>();  
    string pattern = "<image  
xlink:href\\s*=\\s*(?:[\"'\"](?<1>[^\\"'\"]*)  
[\"'\"]|(?<1>\\S+))";  
    MatchCollection matches =  
Regex.Matches(svg, pattern);  
    foreach (Match match in matches)  
        hrefs.Add(match.Groups[1].Value);  
    string folder =  
Path.GetDirectoryName(file);  
    foreach (string href in hrefs) {  
        string image = Path.Combine(folder,  
href);  
        if (!File.Exists(image)) {  
            // href is of form "imageXX.png"  
            where XX is the PixMap ID  
            int id = int.Parse(href.Substring(5,  
href.Length - 9));
```

```

        PixMap pm = pdf.ObjectSoup[id] as
        PixMap;
        using (Bitmap bm = pm.GetBitmap())
            bm.Save(image);
    }
}
}

```

### [Visual Basic]

```

Private Shared Sub SaveAsSvg(pdf As Doc,
file As String)
    pdf.Rendering.Save(file)
    Dim svg As String =
File.ReadAllText(file)
    Dim hrefs As New HashSet(Of String)()
    Dim pattern As String = "<image
xlink:href\s*=\s*(?:[\"' ](?<1>[^\"]*)
[\"' ]|(?<1>\S+))"
    Dim matches As MatchCollection =
Regex.Matches(svg, pattern)
    For Each match As Match In matches
        hrefs.Add(match.Groups(1).Value)
    Next
    Dim folder As String =
Path.GetDirectoryName(file)
    For Each href As String In hrefs
        Dim image As String =
Path.Combine(folder, href)
        If Not File.Exists(image) Then
            ' href is of form "imageXX.png" where
XX is the PixMap ID
            Dim id As Integer =
Integer.Parse(href.Substring(5, href.Length
- 9))
            Dim pm As PixMap =
TryCast(pdf.ObjectSoup(id), PixMap)

```

```
        Using bm As Bitmap = pm.GetBitmap()  
            bm.Save(image)  
        End Using  
    End If  
Next  
End Sub
```

Note that in the above code the images are all exported as PNG. This is a good general purpose lossless export format. However for continuous tone images such as photographs you may wish to export as JPG as this will produce a smaller file size.

It is worth noting that one of the internal compression formats within PDF streams is JPEG. So in cases where the [Stream.Compressions](#) has length one and the [Stream.Compression](#) is `CompressionType.Jpeg`, it is often possible to use [Stream.GetData](#) save the raw data directly to a JPEG file.

However you need to be aware that there are differences. So while often possible, it is not always possible. In most situations for most images in the RGB color space it will work. However for other color spaces such as CMYK it will not and even in the case of RGB the output may be missing secondary data such as any related ICC color profile.



## Common Tasks

---

There are many code examples in the documentation. These cover common tasks like [adding text](#), [flowing text](#) and [adding images](#).

Every major object method or property has an accompanying code sample. So if you want to know how to use a method like [AddText](#) or [AddImageFile](#) just look at the code sample.

Code samples also cover operations like [rendering HTML pages](#), [paged HTML renders](#), [watermarking](#), [appending PDF documents](#) and [drawing pages from one PDF document into another](#).

For in-situ examples you should look at the example web site that comes installed with ABCpdf.

## Text Flow Example

---



This example shows how to flow text from one area to another. The techniques shown here are used to flow text between pages but they could equally well be applied to flowing text between areas - such as columns - on the same page.

First we'll set up some convenient variables. One will determine the gap between our columns and the other will contain the text we want to display.

### [C#]

```
int theID = 0;
string theText = "Gallia est omnis
divisa in partes tres, quarum unam
incolunt Belgae, aliam Aquitani,
tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes...";
// truncated for clarity
```

Setup

### [Visual Basic]

```
Dim theID As Integer = 0
Dim theText As String = "Gallia est
omnis divisa in partes tres, quarum
unam incolunt Belgae, aliam Aquitani,
tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes..."
' truncated for clarity
```

Next we create an ABCpdf Doc object and give it our basic settings.

We enlarge the line width, increase the font size, enable justification and inset the drawing rectangle from the edges of the document.

### [C#]

Doc Obj

```
Doc theDoc = new Doc();
theDoc.Width = 4;
theDoc.FontSize = 32;
theDoc.TextStyle.Justification = 1;
theDoc.Rect.Inset(20, 20);
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Width = 4
theDoc.FontSize = 32
theDoc.TextStyle.Justification = 1
theDoc.Rect.Inset(20, 20)
```

We add our base text to the current page of the document. We then enter a loop, adding pages and chaining HTML boxes together until we run out of text to display.

### [C#]

```
theDoc.FrameRect();
theID = theDoc.AddHtml(theText);
while (theDoc.Chainable(theID)) {
    theDoc.Page = theDoc.AddPage();
    theDoc.FrameRect();
    theID = theDoc.AddHtml("", theID);
}
```

---

Adding

```
}
```

### [Visual Basic]

```
theDoc.FrameRect()  
theID = theDoc.AddHtml(theText)  
While theDoc.Chainable(theID)  
    theDoc.Page = theDoc.AddPage()  
    theDoc.FrameRect()  
    theID = theDoc.AddHtml("", theID)  
End While
```

After adding all our text we save the document to a specific location and clear our document.

### [C#]

```
theDoc.Save(Server.MapPath("textflow.pdf"))  
theDoc.Clear();
```

---

Save

### [Visual Basic]

```
theDoc.Save(Server.MapPath("textflow.pdf"))  
theDoc.Clear()
```

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, institutis, legibus inter se differunt. Gallos ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit. Horum omnium fortissimi sunt Belgae, propterea quod a cultu atque humanitate provinciae longissime absunt, minimeque ad eos mercatores saepe commeant atque ea quae ad effeminandos animos pertinent important, proximique sunt Germanis, qui trans Rhenum incolunt, quibuscum continenter bellum gerunt. Qua de causa Helvetii quoque reliquos Gallos virtute praecedunt, quod fere cotidianis proeliis cum Germanis contendunt, cum aut suis finibus eos prohibent aut ipsi in eorum finibus bellum gerunt. Eorum una, pars, quam Gallos obtinere dictum est, initium capit a flumine Rhodano, continetur Garumna flumine, Oceano, finibus

textflow.pdf [Page 1]

Belgarum, attingit etiam ab Sequanis et Helvetiis flumen Rhenum, vergit ad septentriones. Belgae ab extremis Galliae finibus oriuntur, pertinent ad inferiorem partem fluminis Rheni, spectant in septentrionem et orientem solem. Aquitania a Garumna flumine ad Pyrenaeos montes et eam partem Oceani quae est ad Hispaniam pertinet; spectat inter occasum solis et septentriones. Apud Helvetios longe nobilissimus fuit et ditissimus Orgetorix. Is M. Messala, M. Pisone consulibus regni cupiditate inductus coniurationem nobilitatis fecit et civitati persuasit ut de finibus suis cum omnibus copiis exirent: perfacile esse, cum virtute omnibus praestarent, totius Galliae imperio potiri. Id hoc facilius iis persuasit, quod undique loci natura Helvetii continentur: una ex parte flumine Rheno latissimo atque altissimo, qui agrum Helvetium a Germanis dividit; altera ex parte monte Iura altissimo, qui est inter Sequanos et Helvetios; tertia lacu Lemanno

---

Results

textflow.pdf [Page 2]

et flumine Rhodano, qui provinciam nostram ab Helvetiis dividit. His rebus fiebat ut et minus late vagarentur et minus facile finitimis bellum inferre possent; quae ex parte homines bellandi cupidi magno dolore adficiiebantur. Pro multitudine autem hominum et pro gloria belli atque fortitudinis angustos se fines habere arbitrabantur, qui in longitudinem milia passuum CCXL, in latitudinem CLXXX patebant.



Next we create an ABCpdf Doc object and give it our basic settings.

We enlarge the line width, increase the font size, enable justification and inset the drawing rectangle from the edges of the document.

### [C#]

Doc Obj

```
Doc doc = new Doc();
doc.Width = 4;
doc.FontSize = 32;
doc.TextStyle.Justification = 1;
doc.Rect.Inset(20, 20);
```

### [Visual Basic]

```
Dim doc As Doc = New Doc()
doc.Width = 4
doc.FontSize = 32
doc.TextStyle.Justification = 1
doc.Rect.Inset(20, 20)
```

We save the rect since we're going to need it later. Then we add an image to the left hand side of the page. This is the image we are going to flow around.

### [C#]

```
string saveRect = doc.Rect.String;
using (XImage xi =
XImage.FromFile("pic.jpg", null))
{
    doc.Rect.Resize(xi.Width / 2,
```

---

Image

```
xi.Height / 2,  
XRect.Corner.TopLeft);  
    doc.AddImage(xi);  
}
```

### [Visual Basic]

```
Dim saveRect As String = doc.Rect.  
[String]  
Using xi As XImage =  
XImage.FromFile("pic.jpg",  
Nothing)  
    doc.Rect.Resize(xi.Width / 2,  
xi.Height / 2,  
XRect.Corner.TopLeft)  
    doc.AddImage(xi)  
End Using
```

The crucial part occurs here which is where we set up the variable left margins to ensure that our text is shifted away from the image. If we had images on the right we would want to use variable right margins too. But here it is not necessary.

### [C#]

```
double padX = doc.FontSize;  
double padY = doc.FontSize / 3;  
string format = "<stylerun  
justification=\"1.0\"  
leftmargins=\"0 {0} {1}\">";  
string style =  
string.Format(format,  
doc.Rect.Height + padY,
```

---

Text

```
doc.Rect.Width + padX);
```

### [Visual Basic]

```
Dim padX As Double = doc.FontSize  
Dim padY As Double = doc.FontSize  
/ 3  
Dim format As String = "<stylerun  
justification=""1.0""  
leftmargins=""0 {0} {1}"">"  
Dim style As String =  
String.Format(format,  
doc.Rect.Height + padY,  
doc.Rect.Width + padX)
```

After adding and framing our text we save the document to a specified location and clear our document.

### [C#]

```
doc.Rect.String = saveRect;  
doc.FrameRect();  
int id = doc.AddHtml(style + text +  
"</stylerun>");  
doc.Save("textflowroundimage.pdf");
```

---

Save

### [Visual Basic]

```
doc.Rect.[String] = saveRect  
doc.FrameRect()  
Dim id As Integer =  
doc.AddHtml((style & text) + "  
</stylerun>")  
doc.Save("textflowroundimage.pdf")
```

---

## Results



Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, institutis, legibus inter se differunt. Gallos ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit. Horum omnium fortissimi sunt Belgae, propterea quod a cultu atque humanitate provinciae longissime absunt, minimeque ad eos mercatores saepe commeant atque ea quae ad effeminandos animos pertinent important, proximique sunt Germanis, qui trans Rhenum incolunt, quibuscum continenter bellum gerunt. Qua de causa Helvetii quoque reliquos Gallos virtute praecedunt, quod fere cotidianis proeliis cum Germanis contendunt, cum aut

textflowroundimage.pdf



## Multistyle Example

---

This example shows how to create multistyled text.

We want to display all our proper names in bold so we enclose them in bold tags.

### [C#]

```
string theText = "<b>Gallia</b> est  
omnis divisa in partes tres, quarum  
unam incolunt <b>Belgae</b>, aliam  
<b>Aquitani</b>, tertiam qui ipsorum  
lingua <b>Celtae</b>, nostra  
<b>Galli</b> appellantur.";
```

---

### Setup

### [Visual Basic]

```
Dim theText As String = "  
<b>Gallia</b> est omnis divisa in  
partes tres, quarum unam incolunt  
<b>Belgae</b>, aliam <b>Aquitani</b>,  
tertiam qui ipsorum lingua  
<b>Celtae</b>, nostra <b>Galli</b>  
appellantur."
```

Next we create an ABCpdf Doc object and give it some basic settings. Although we could pass our HTML styled text directly to the AddHtml function, we can take more control over the way that fonts are added to the PDF if we specify font IDs.

### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 72;
theDoc.Rect.Inset(10, 10);
theDoc.FrameRect();
int theFont1 =
theDoc.EmbedFont("Verdana",
LanguageType.Latin, false, true);
int theFont2 =
theDoc.EmbedFont("Verdana Bold",
LanguageType.Latin, false, true);
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 72
theDoc.Rect.Inset(10, 10)
theDoc.FrameRect()
Dim theFont1 As Integer =
theDoc.EmbedFont("Verdana",
LanguageType.Latin, False, True)
Dim theFont2 As Integer =
theDoc.EmbedFont("Verdana Bold",
LanguageType.Latin, False, True)
```

We replace the bold tags with font tags that directly reference our chosen fonts and then add the HTML styled text to the current rectangle.

### [C#]

```
theText = "<font pid=" +
theFont1.ToString() + ">" + theText +
"</font>";
```

---

Adding

```
theText = theText.Replace("<b>", "  
<font pid=\"" + theFont2.ToString() +  
">");  
theText = theText.Replace("</b>", "  
</font>");  
theDoc.AddHtml(theText);
```

### [Visual Basic]

```
theText = "<font pid=\"" +  
theFont1.ToString() + ">" + theText +  
</font>"  
theText = theText.Replace("<b>", "  
<font pid=\"" + theFont2.ToString() +  
">")  
theText = theText.Replace("</b>", "  
</font>")  
theDoc.AddHtml(theText)
```

Finally we save and clear the document.

### [C#]

```
theDoc.Save(Server.MapPath("styles.pdf"));  
theDoc.Clear();
```

---

Save

### [Visual Basic]

```
theDoc.Save(Server.MapPath("styles.pdf"))  
theDoc.Clear()
```

---

Results

**Gallia** est  
omnis divisa in  
partes tres,  
quarum unam  
incolunt  
**Belgae**, aliam  
**Aquitani**,  
tertiam qui  
ipsorum lingua  
**Celtae**, nostra

styles.pdf

# Image Example

---



This example shows how to create a simple PDF displaying an image.

First we create an ABCpdf Image object and we assign our image

## [C#]

```
XImage theImg = new XImage();  
theImg.SetFile(Server.MapPath("../mypics/pic.jpg"))
```

---

Image

## [Visual Basic]

```
Dim theImg As XImage = New XImage()  
theImg.SetFile(Server.MapPath("../mypics/pic.jpg"))
```

Next we create an ABCpdf Doc object.

When we add our image it will be scaled to fit the current rect so is important that we adjust the rect to reflect the dimensions of our image. Here we assume a one to one ratio between pixels and points which will give us a 72 dpi result when printed.

## [C#]

```
Doc theDoc = new Doc();  
theDoc.Rect.Left = 100;  
theDoc.Rect.Bottom = 100;  
theDoc.Rect.Width = theImg.Width;  
theDoc.Rect.Height = theImg.Height;  
theDoc.AddImageObject(theImg, false);
```

---

Doc

```
theDoc.Save(Server.MapPath("image.pdf"));  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Rect.Left = 100  
theDoc.Rect.Bottom = 100  
theDoc.Rect.Width = theImg.Width  
theDoc.Rect.Height = theImg.Height  
theDoc.AddImageObject(theImg, false)  
theDoc.Save(Server.MapPath("image.pdf"))  
theDoc.Clear()
```

---

Results



image.pdf



## Deletion Example

---

This example shows how to delete pages from a PDF document.

First we create an ABCpdf Doc object and read our source document. We store the number of pages we're going to delete - we're going to delete all but one.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.pdf"));
int theCount = theDoc.PageCount - 1;
```

Setup

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample.pdf"))
Dim theCount As Integer = theDoc.PageCount - 1
```

We go round a loop deleting the second page each time.

### [C#]

```
for (int i = 0; i < theCount; i++) {
    theDoc.PageNumber = 2;
    theDoc.Delete(theDoc.Page);
}
```

Delete

### [Visual Basic]

```
For i As Integer = 1 to theCount
```

```
theDoc.PageNumber = 2
theDoc.Delete(theDoc.Page)
Next
```

We add some text to the PDF so that we know how many pages we've deleted. Finally we save the PDF.

### [C#]

```
theDoc.FontSize = 500;
theDoc.Color.String = "255 0 0";
theDoc.TextStyle.HPos = 0.5;
theDoc.TextStyle.VPos = 0.3;
theDoc.AddText(theCount.ToString());
theDoc.Save(Server.MapPath("deletion.pdf"));
theDoc.Clear();
```

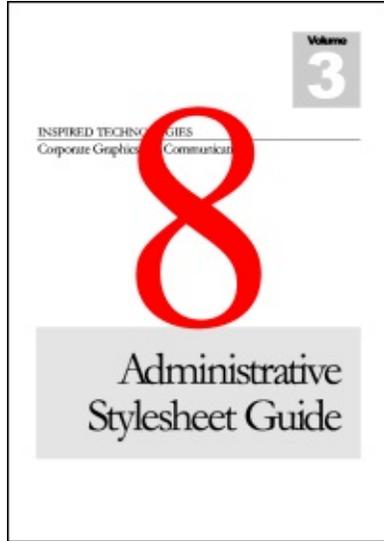
---

Save

### [Visual Basic]

```
theDoc.FontSize = 500
theDoc.Color.String = "255 0 0"
theDoc.TextStyle.HPos = 0.5
theDoc.TextStyle.VPos = 0.3
theDoc.AddText(theCount.ToString())
theDoc.Save(Server.MapPath("deletion.pdf"))
theDoc.Clear()
```

Results



deletion.pdf

# Headers and Footers Example

---



This example shows one method of adding headers and footers.

First we create an ABCpdf Doc object and define the content we're going to be adding.

## [C#]

```
Doc theDoc = new Doc();
int theID, theCount;
string theText = "Gallia est omnis divisa
in partes tres, quarum unam incolunt
Belgae, aliam Aquitani, tertiam qui ipsorum
lingua Celtae, nostra Galli appellantur. Hi
omnes..."; // truncated for clarity
```

---

Setup

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theID, theCount As Integer
Dim theText As String = "Gallia est omnis
divisa in partes tres, quarum unam incolunt
Belgae, aliam Aquitani, tertiam qui ipsorum
lingua Celtae, nostra Galli appellantur. Hi
omnes..." ' truncated for clarity
```

We set up the style for our content.

We need to specify the Rect defining the area in which

content will be inserted, a color (red) and a font size.

We add the text using the same method as employed in the [Text Flow Example](#). We draw a frame round our content area each time so we can see how the text has been positioned.

### [C#]

```
theDoc.Rect.String = "100 200 500 600";
theDoc.Color.String = "255 0 0";
theDoc.FontSize = 24;
theID = theDoc.AddHtml(theText);
theDoc.FrameRect();
while (theDoc.Chainable(theID)) {
    theDoc.Page = theDoc.AddPage();
    theID = theDoc.AddHtml("", theID);
    theDoc.FrameRect();
}
theCount = theDoc.PageCount;
```

---

Content

### [Visual Basic]

```
theDoc.Rect.String = "100 200 500 600"
theDoc.Color.String = "255 0 0"
theDoc.FontSize = 24
theID = theDoc.AddHtml(theText)
theDoc.FrameRect()
While theDoc.Chainable(theID)
    theDoc.Page = theDoc.AddPage()
    theID = theDoc.AddHtml("", theID)
    theDoc.FrameRect()
End While
theCount = theDoc.PageCount
```

We set up the style for our header.

We need to specify the Rect defining the area in which the header will be inserted, a color (green) and a font size. We use the HPos and VPos parameters to center the text both horizontally and vertically.

We then iterate through the pages in the document adding headers as we go. We frame our headers so we can see the header area.

### [C#]

```
theDoc.Rect.String = "100 650 500 750";
theDoc.TextStyle.HPos = 0.5;
theDoc.TextStyle.VPos = 0.5;
theDoc.Color.String = "0 255 0";
theDoc.FontSize = 36;
for (int i = 1; i <= theCount; i++) {
    theDoc.PageNumber = i;
    theDoc.AddText("De Bello Gallico");
    theDoc.FrameRect();
}
```

---

Header

### [Visual Basic]

```
theDoc.Rect.String = "100 650 500 750"
theDoc.TextStyle.HPos = 0.5
theDoc.TextStyle.VPos = 0.5
theDoc.Color.String = "0 255 0"
theDoc.FontSize = 36
For i As Integer = 1 To theCount
    theDoc.PageNumber = i
    theDoc.AddText("De Bello Gallico")
    theDoc.FrameRect()
Next
```

We set up the style for our footer.

We need to specify the Rect defining the area in which the footer will be inserted, a color (blue) and a font size. We use the HPos and VPos parameters to center the text vertically and align it to the right.

We then iterate through the pages in the document adding footers as we go. We frame our footers so we can see the footer area.

### [C#]

```
theDoc.Rect.String = "100 50 500 150";
theDoc.TextStyle.HPos = 1.0;
theDoc.TextStyle.VPos = 0.5;
theDoc.Color.String = "0 0 255";
theDoc.FontSize = 36;
for (int i = 1; i <= theCount; i++) {
    theDoc.PageNumber = i;
    theDoc.AddText("Page " + i.ToString() + "
of " + theCount.ToString());
    theDoc.FrameRect();
}
```

---

Footer

### [Visual Basic]

```
theDoc.Rect.String = "100 50 500 150"
theDoc.TextStyle.HPos = 1.0
theDoc.TextStyle.VPos = 0.5
theDoc.Color.String = "0 0 255"
theDoc.FontSize = 36
For i As Integer = 1 To theCount
    theDoc.PageNumber = i
    theDoc.AddText("Page " + i.ToString() + "
of " + theCount.ToString())
    theDoc.FrameRect()
Next
```

Finally we save the PDF.

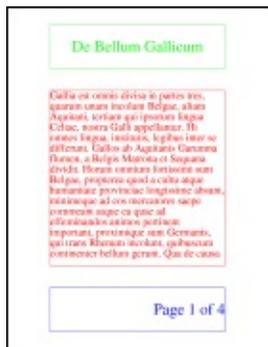
[C#]

```
theDoc.Save(Server.MapPath("headerfooter.pdf"))  
theDoc.Clear();
```

Save

[Visual Basic]

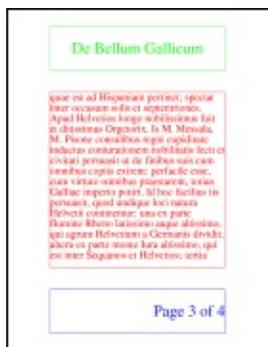
```
theDoc.Save(Server.MapPath("headerfooter.pdf"))  
theDoc.Clear();
```



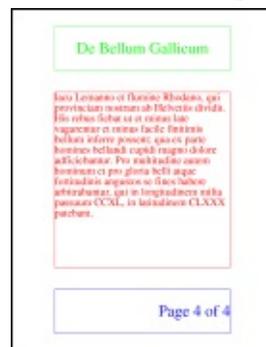
headerfooter.pdf [Page 1]



headerfooter.pdf [Page 2]



headerfooter.pdf [Page 3]



headerfooter.pdf [Page 4]

Results



## Landscape Example

---



This example shows how to create a PDF document rotated by 90 degrees for a landscape rather than portrait view.

We start by creating a PDF document. We use two transforms to apply a generic 90 degree rotation around the center of the document and rotate the drawing rectangle by the same amount. After applying our rotation we add some text to the page.

### [C#]

```
Doc theDoc = new Doc();
// apply a rotation transform
double w = theDoc.MediaBox.Width;
double h = theDoc.MediaBox.Height;
double l = theDoc.MediaBox.Left;
double b = theDoc.MediaBox.Bottom;
theDoc.Transform.Rotate(90, l, b);
theDoc.Transform.Translate(w, 0);

// rotate our rectangle
theDoc.Rect.Width = h;
theDoc.Rect.Height = w;

// add some text
theDoc.Rect.Inset(50, 50);
theDoc.FontSize = 96;
theDoc.AddText("Landscape Orientation");
```

Setup

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
' apply a rotation transform
Dim w As Double = theDoc.MediaBox.Width
Dim h As Double = theDoc.MediaBox.Height
Dim l As Double = theDoc.MediaBox.Left
Dim b As Double = theDoc.MediaBox.Bottom
theDoc.Transform.Rotate(90, l, b)
theDoc.Transform.Translate(w, 0)

' rotate our rectangle
theDoc.Rect.Width = h
theDoc.Rect.Height = w

' add some text
theDoc.Rect.Inset(50, 50)
theDoc.FontSize = 96
theDoc.AddText("Landscape Orientation")

```

Drawing the text rotated does not rotate the page itself. To change the default orientation of the document we need to apply a rotation to the root page object. By doing this we ensure that every page in the document is viewed rotated.

### [C#]

```

// adjust the default rotation and save
int theID = theDoc.GetInfoInt(theDoc.Root,
"Pages");
theDoc.SetInfo(theID, "/Rotate", "90");
theDoc.Save(Server.MapPath("landscape.pdf"))
theDoc.Clear();

```

Rotate

### [Visual Basic]

```

' adjust the default rotation and save

```

```
Dim theID As Integer =  
theDoc.GetInfoInt(theDoc.Root, "Pages")  
theDoc.SetInfo(theID, "/Rotate", "90")  
theDoc.Save(Server.MapPath("landscape.pdf"))  
theDoc.Clear()
```



Landscape  
Orientation

---

Results

landscape.pdf

## Small Table Example

---



This example shows how to draw a single page table. ABCpdf does not provide table drawing routines itself so this example uses a Table Class to position the table elements.

You can find the full project and classes under the ABCpdf menu item. The project includes code for laying out a small table, a large table spreading over more than one page, an invoice and a product list.

We start by creating our document object and reading the data for our table. For the purposes of this example we'll assume that our data is in a standard tab delimited format.

### [C#]

```
string theText =  
ReadDataFromFile(theRez +  
"text6.txt");  
Doc theDoc = new Doc();  
// set up document  
theDoc.FontSize = 16;  
theDoc.Rect.Inset(20, 20);
```

Setup

### [Visual Basic]

```
Dim theText As String =  
ReadDataFromFile(theRez +  
"text6.txt")  
Dim theDoc As Doc = New Doc()  
' set up document  
theDoc.FontSize = 16
```

```
theDoc.Rect.Inset(20, 20)
```

We create a new table object passing in values to tell it what rectangle it can occupy (it takes the current document rectangle) and how many columns of data it should be prepared for.

Columns are assigned relative widths and expand horizontally to fit the table rectangle. Here we're specifying five columns.

Most of our columns will be right aligned so we set the default horizontal alignment to 1.

### [C#]

Rotate

```
PDFTable theTable = new  
PDFTable(theDoc, 5);  
theTable.CellPadding = 5;  
theTable.HorizontalAlignment = 1;
```

### [Visual Basic]

```
Dim theTable As PDFTable = New  
PDFTable(theDoc, 5)  
theTable.CellPadding = 5  
theTable.HorizontalAlignment = 1;
```

We iterate through the table data adding rows as we go. We override the right alignment for the first column and we shade alternating rows using a light gray color. Finally we frame the table and save the document.

## [C#]

```
theText = theText.Trim();
theText = theText.Replace("\r\n", "\r");
string[] theRows = theText.Split(new
char[] {'\r'});

for (int i = 0; i < theRows.Length; i++) {
    theTable.NextRow();
    string[] theCols = theRows[i].Split(new
char[] {'\t'});
    theCols[0] = "<stylerun hpos=0>" +
theCols[0] + "</stylerun>";
    theTable.AddHtml(theCols);
    if ((i % 2) == 1)
        theTable.FillRow("220 220 220", i);
}
theTable.Frame();

theDoc.Flatten();
theDoc.Save(Server.MapPath("table1.pdf"));
theDoc.Clear();
```

---

Add

## [Visual Basic]

```
theText = theText.Trim()
theText = theText.Replace(vbCrLf, vbCr)
Dim theRows() As String
theRows = theText.Split(New Char() {vbCr})

For i As Integer = 0 To theRows.Length - 1
    theTable.NextRow()
    Dim theCols As String() =
theRows(i).Split(New Char() {vbTab})
    theCols(0) = "<stylerun hpos=0>" +
theCols(0) + "</stylerun>"
    theTable.AddHtml(theCols)
    If (i Mod 2) = 1 Then
```

```
        theTable.FillRow("220 220 220", i)
    End If
Next
theTable.Frame()

theDoc.Flatten()
theDoc.Save(Server.MapPath("table1.pdf"))
theDoc.Clear()
```

Using the following input data:

Planet	Distance From Sun (miles)	Diameter (miles)	Year Length (days)	Day Length (days)
Mercury	36,000,000	3,030	88	58.00
Venus	67,000,000	7,520	225	225.00
Earth	93,000,000	7,925	365	1.00
Mars	142,000,000	4,210	687	1.00
Jupiter	484,000,000	88,730	4,344	0.40
Saturn	888,000,000	74,975	10,768	0.40
Uranus	1,800,000,000	31,760	30,660	0.70
Neptune	2,800,000,000	30,600	60,150	0.65
Pluto	3,600,000,000	1,410	90,520	0.25

We get the following output.

---

## Results

Planet	Distance From Sun (miles)	Diameter (miles)	Year Length (days)	Day Length (days)
Mercury	36,000,000	3,030	88	58.00
Venus	67,000,000	7,520	225	225.00
Earth	93,000,000	7,925	365	1.00
Mars	142,000,000	4,210	687	1.00
Jupiter	484,000,000	88,730	4,344	0.40
Saturn	888,000,000	74,975	10,768	0.40
Uranus	1,800,000,000	31,760	30,660	0.70
Neptune	2,800,000,000	30,600	60,150	0.65
Pluto	3,600,000,000	1,410	90,520	0.25

table1.pdf

# Large Table Example

---



This example shows how to draw a multi-page table. ABCpdf does not provide table drawing routines itself so this example uses a Table Class to position the table elements.

You can find the full project and classes under the ABCpdf menu item. The project includes code for laying out a small table, a large table spreading over more than one page, an invoice and a product list.

We start by creating our document object and reading the data for our table. For the purposes of this example we'll assume that our data is in a standard tab delimited format.

## [C#]

```
string theText =  
ReadDataFromFile(theRez +  
"text7.txt");  
Doc theDoc = new Doc();  
// set up document  
theDoc.FontSize = 12;  
theDoc.Rect.Inset(20, 20);
```

Setup

## [Visual Basic]

```
Dim theText As String =  
ReadDataFromFile(theRez +  
"text7.txt")  
Dim theDoc As Doc = New Doc()  
' set up document  
theDoc.FontSize = 12
```

```
theDoc.Rect.Inset(20, 20)
```

We create a new table object passing in values to tell it what rectangle it can occupy (it takes the current document rectangle) and how many columns of data it should be prepared for.

Columns are assigned relative widths and expand horizontally to fit the table rectangle. Here we're specifying six columns and a number of relative widths. We're padding the cells so there are gaps between the rows and columns. Finally we specify a header which repeats as new pages are added.

### [C#]

Rotate

```
PDFTable theTable = new  
PDFTable(theDoc, 6);  
// some columns extra width  
theTable.SetColumnWidths(new double  
[] {2, 1, 3 , 2, 1, 4});  
theTable.CellPadding = 5;  
theTable.RepeatHeader = true;
```

### [Visual Basic]

```
Dim theTable As PDFTable = New  
PDFTable(theDoc, 6)  
' some columns extra width  
theTable.SetColumnWidths(New Double()  
{2, 1, 3 , 2, 1, 4})  
theTable.CellPadding = 5  
theTable.RepeatHeader = True
```

We iterate through the table data adding rows and columns as we go. Every time we add a row we check to see if the page number has changed and restart the shading if it has. This ensures the header is always unshaded. Finally we save the document.

### [C#]

```
theText = theText.Replace("\r\n", "\r");
string[] theRows = theText.Split(new
char[] {'\r'});
int thePage = 1;
bool theShade = false;
for (int i = 0; i < theRows.Length; i++) {
    theTable.NextRow();
    string[] theCols = theRows[i].Split(new
char[] {'\t'});
    theTable.AddHtml(theCols);
    if (theDoc.PageNumber > thePage) {
        thePage = theDoc.PageNumber;
        theShade = true;
    }
    if (theShade)
        theTable.FillRow("200 200 200",
theTable.Row);
    theShade = ! theShade;
}
theDoc.Flatten();
theDoc.Save(Server.MapPath("table2.pdf"));
theDoc.Clear();
```

---

Add

### [Visual Basic]

```
theText = theText.Replace(vbCrLf, vbCr)
Dim theRows() As String =
theText.Split(New Char() {vbCr})
Dim thePage As Integer = 1
```

```
Dim theShade As Boolean = False
For i As Integer = 0 To theRows.Length - 1
    theTable.NextRow()
    Dim theCols As String() =
theRows(i).Split(New Char() {vbTab})
    theTable.AddHtml(theCols)
    If theDoc.PageNumber > thePage Then
        thePage = theDoc.PageNumber
        theShade = true
    End If
    If theShade Then
        theTable.FillRow("200 200 200",
theTable.Row)
    End If
    theShade = Not theShade
Next
theDoc.Flatten()
theDoc.Save(Server.MapPath("table2.pdf"))
theDoc.Clear()
```

Using a large quantity of input data. We get the following output.



table2.pdf - [Page 4]

table2.pdf - [Page 5]

# Unicode Example

---



This example shows how to add complex scripts such as Chinese, Japanese and Korean. Here we choose to embed and subset our font to ensure our document renders correctly on all platforms.

First we create an ABCpdf Doc object and set the font size.

## [C#]

```
Doc theDoc = new Doc();  
theDoc.FontSize = 32;
```

---

Setup

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.FontSize = 32
```

We read in our Japanese text from a Unicode text file.

## [C#]

```
string thePath =  
Server.MapPath("../Rez/Japanese2.txt");  
// Utilities.ReadString is an external  
function not defined here  
string theText =  
Utilities.ReadString(thePath);
```

---

Read

## [Visual Basic]

```
Dim thePath As String =  
Server.MapPath("../Rez/Japanese2.txt")  
' Utilities.ReadString is an external  
function not defined here  
Dim theText As String =  
Utilities.ReadString(thePath)
```

Because we want to ensure that our document renders correctly on all platforms we're going to embed our font in Unicode format. We specify a left-to-right writing direction and we choose to subset our font.

Please note when embedding fonts you must ensure you have permission to embed and redistribute the embedded font as part of your PDF.

## [C#]

```
theDoc.Page = theDoc.AddPage();  
theDoc.Font = theDoc.EmbedFont("MS  
PGothic", LanguageType.Unicode, false,  
true);  
theDoc.AddText("Japanese" + theText);
```

Add

## [Visual Basic]

```
theDoc.Page = theDoc.AddPage()  
theDoc.Font = theDoc.EmbedFont("MS  
PGothic", LanguageType.Unicode, False,  
True)  
theDoc.AddText("Japanese" + theText)
```

Just to show how it works we'll also render a page in vertical writing mode.

### **[C#]**

```
theDoc.Page = theDoc.AddPage();
theDoc.Font = theDoc.EmbedFont("MS
PGothic", LanguageType.Unicode, true,
true);
theDoc.AddText("Japanese" + theText);
```

---

Add

### **[Visual Basic]**

```
theDoc.Page = theDoc.AddPage()
theDoc.Font = theDoc.EmbedFont("MS
PGothic", LanguageType.Unicode, True,
True)
theDoc.AddText("Japanese" + theText)
```

Finally we save at a specified location.

### **[C#]**

```
theDoc.Save(Server.MapPath("unicode.pdf"));
// finished
```

---

Save

### **[Visual Basic]**

```
theDoc.Save(Server.MapPath("unicode.pdf"))
' finished
```

We get the following output.

Japanese 発見日: 2001年9月18日  
最終更新日: 2001年12月26日 16:30

現時点では、W32.Nimda.A@mmの大量メール送信ルーチンが初回感染後10日間の潜伏期間を経て再び活性化したことによる顕著な感染拡大は確認されていません。

W32.Nimda.A@mmは、多数の感染方法を持つ新しい大量メール送信型ワームです。Nimdaという名称は、管理者権限を逆手にとることに由来しています(スベルが管理者を意味する"admin"の逆)。ワームは電子メールで自分自身を送り、有効なネットワーク共有を捜して、修正プログラムが適用されていないMicrosoft IS Web サーバにワーム自身をコピーしようと試みます。また、ローカルドライブ上とリモートネットワーク上の両方でファイルに感染するウイルスでもありません。

このワームはUnicode Web Traversal Web (Web サーバにフォルダへの侵入)と呼ばれるセキュリティホールを使ってこれを行いま

スネと1正日るす法W 大間量規 発J  
でっ試1フ身。本3 はをメ時 終J  
もとみSロをaとN持2 はをメ時 終J  
あワま グ送dににつ、 認経一色 終J  
りーずWラリm由m新N さ再送は 新日  
まク。eム。i美dしi れび信、 : s  
ず上まbが有nしあいm て活ルW : a  
。のた、適効てと大d い性一3 2 a  
。開、サ開なのいい道a ま化子2 0 発  
方ローさな送まらメ。 せしん 0 異  
でーバれつ)す名ーA んたがN 1 日  
フかにてト。(特ル@ 。こ回i 年:  
アルワいワウスは淡m と回m 1 2  
イドーなーべ、信m に感d 2 2  
ルラムいクムル管理は よ染a 月0  
にイ自M共はが理ワ、 る後 2 0  
送フ身;有電管者一多 題!A 6 1  
染上をeを子種種一多 者日m 9  
ずとコr理メ者限での なる日m 1  
るリビ。し。をを 感mの 1  
ウモー。aてル感送 染の 6 1  
イー。し。で味手 感mの 8  
ルトよf修自すに 播潜大 3 日  
う 分 と 伏 期 0

Results

unicode.pdf - [Page 1]

unicode.pdf - [Page 2]

# Paged HTML Example

---



This example shows how to import an HTML page into a multi-page PDF document.

We first create a Doc object and inset the edges a little so that the HTML will appear in the middle of the page.

## [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(72, 144);
theDoc.HtmlOptions.UseScript = false; //
set to true if your layout is JavaScript
dependent
```

---

Setup

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(72, 144)
theDoc.HtmlOptions.UseScript = False '
set to true if your layout is JavaScript
dependent
```

We add the first page of HTML. We save the returned ID as this will be used to add subsequent pages.

## [C#]

```
theDoc.Page = theDoc.AddPage();
int theID;
theID =
```

---

```
theDoc.AddImageUrl("http://www.yahoo.com/")
```

Page

### [Visual Basic]

```
theDoc.Page = theDoc.AddPage()  
Dim theID As Integer  
theID =  
theDoc.AddImageUrl("http://www.yahoo.com/")
```

We now chain subsequent pages together. We stop when we reach a page which wasn't truncated.

### [C#]

```
while (true) {  
    theDoc.FrameRect(); // add a black  
border  
    if (!theDoc.Chainable(theID))  
        break;  
    theDoc.Page = theDoc.AddPage();  
    theID = theDoc.AddImageToChain(theID);  
}
```

---

Chain

### [Visual Basic]

```
While True  
    theDoc.FrameRect() ' add a black  
border  
    If Not theDoc.Chainable(theID) Then  
        Exit While  
    End If  
    theDoc.Page = theDoc.AddPage()  
    theID = theDoc.AddImageToChain(theID)  
End While
```

After adding the pages we can flatten them. We can't do this until after the pages have been added because flattening will invalidate our previous ID and break the chain.

### [C#]

```
for (int i = 1; i <= theDoc.PageCount; i++) {  
    theDoc.PageNumber = i;  
    theDoc.Flatten();  
}
```

Flatten

### [Visual Basic]

```
For i As Integer = 1 To theDoc.PageCount  
    theDoc.PageNumber = i  
    theDoc.Flatten()  
Next
```

Finally we save.

### [C#]

```
theDoc.Save(Server.MapPath("pagedhtml.pdf"))  
theDoc.Clear();
```

Save

### [Visual Basic]

```
theDoc.Save(Server.MapPath("pagedhtml.pdf"))  
theDoc.Clear()
```

We get the following output.

Results



pagedhtml.pdf [Page 1]



pagedhtml.pdf [Page 2]

# Doc Properties Example

---



This example shows how to insert document properties. Document properties can be viewed from Acrobat Reader and most commonly provide information on the document Title and Author.

This example requires some knowledge of the [Adobe PDF Specification](#). Section 9.2.1 of the document details the way in which document properties are stored. Section 3.8.2 of the document details the way that dates are specified within PDF documents.

First we create an ABCpdf Doc object and add some simple content.

## [C#]

```
Doc theDoc = new Doc();
theDoc.Page = theDoc.AddPage();
theDoc.AddText("My first document...");
```

—Src

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Page = theDoc.AddPage()
theDoc.AddText("My first document...")
```

Looking at the Adobe PDF Specification we can see that the document properties we want to change are referenced from an entry called "/Info" in the document trailer. So our first step is to create a new PDF

dictionary and reference it from the this entry.

### [C#]

Dest

```
int theID = theDoc.AddObject("<< >>");  
theDoc.SetInfo(-1, "/Info:Ref",  
theID.ToString());
```

### [Visual Basic]

```
Dim theID As Integer =  
theDoc.AddObject("<< >>")  
theDoc.SetInfo(-1, "/Info:Ref",  
theID.ToString())
```

Now we have to insert our summary information into the object we've just added.

### [C#]

```
theDoc.SetInfo(theID, "/Title:Text",  
"ABCpdf");  
theDoc.SetInfo(theID, "/Author:Text",  
"WebSupergoo");  
theDoc.SetInfo(theID, "/Subject:Text",  
"ABCpdf Documentation");  
theDoc.SetInfo(theID, "/Keywords:Text",  
"ABCpdf,PDF,Docs");  
theDoc.SetInfo(theID, "/Creator:Text",  
"WebSupergoo");  
DateTime theDate = DateTime.Now;  
theDoc.SetInfo(theID,  
"/CreationDate:Text", theDate);  
theDoc.SetInfo(theID, "/ModDate:Text",  
theDate);  
theDoc.SetInfo(theID, "/Trapped:Name",
```

---

Add

```
"False");
```

### [Visual Basic]

```
theDoc.SetInfo(theID, "/Title:Text",  
"ABCpdf")  
theDoc.SetInfo(theID, "/Author:Text",  
"WebSupergoo")  
theDoc.SetInfo(theID, "/Subject:Text",  
"ABCpdf Documentation")  
theDoc.SetInfo(theID, "/Keywords:Text",  
"ABCpdf,PDF,Docs")  
theDoc.SetInfo(theID, "/Creator:Text",  
"WebSupergoo")  
Dim theDate As DateTime = DateTime.Now  
theDoc.SetInfo(theID,  
"/CreationDate:Text", theDate)  
theDoc.SetInfo(theID, "/ModDate:Text",  
theDate)  
theDoc.SetInfo(theID, "/Trapped:Name",  
"False")
```

Finally we save.

### [C#]

```
theDoc.Save(Server.MapPath("docprops.pdf"));  
// finished
```

---

Save

### [Visual Basic]

```
theDoc.Save(Server.MapPath("docprops.pdf"))  
' finished
```



# eForm Fields Example

---



This example shows how to change the values of eForm fields. In this example we simply replace each of the fields in a form with the name of that field.

First we create an ABCpdf Doc object and read in our template f

## [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/form.pdf")
```

---

Src

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/form.pdf")
```

We iterate through each of the top level fields. For each field we set the value of the field to be equal to the value of the name.

## [C#]

```
string[] theNames =
theDoc.Form.GetFieldNames();
foreach (string theName in theNames) {
    Field theField = theDoc.Form[theName];
    theField.Value = theField.Name;
}
```

---

Add

### [Visual Basic]

```
Dim theNames As String() =  
theDoc.Form.GetFieldNames()  
Dim theName As String  
For Each theName In theNames  
    Dim theField As Field =  
theDoc.Form(theName)  
    theField.Value = theField.Name  
Next
```

Finally we save.

### [C#]

```
theDoc.Save(Server.MapPath("eformfields.pdf"));
```

---

Save

### [Visual Basic]

```
theDoc.Save(Server.MapPath("eformfields.pdf"))
```

Given the following document.

Revised Form 1825  
January 2007  
Title 4, Code of Federal Regulations  
205.125

CLAIM AGAINST THE UNITED STATES  
FOR  
AMOUNTS DUE IN THE CASE OF A DECEASED CREDITOR

1. I, the undersigned, hereby make claim as \_\_\_\_\_ for amounts due from the  
(Relationship)

United States in the case of \_\_\_\_\_ who died on the \_\_\_\_\_ day  
(Name of decedent)

of \_\_\_\_\_ while domiciled in the State of \_\_\_\_\_  
(Month) (Year)

2. The basis of this claim is as follows:  
\_\_\_\_\_

3. I have been duly appointed \_\_\_\_\_ of the estate of the decedent, as evidenced  
(Executor or Administrator)

by certificate of appointment herewith, administration having been taken out in the interest of:

\_\_\_\_\_

and such appointment is still in full force and effect.

(If willing to act as the executor or administrator of the estate of the decedent, no certificate was required, but a court certificate of letters testamentary or of administration must be submitted. If you are the executor or administrator of the estate of the decedent, attach paragraphs 4, 5, and 6.)

4. If an executor or administrator has not been or will not be appointed, the following information should be furnished: The decedent is survived by \_\_\_\_\_

Name

Widow or widower of (name, no state)

Children (if more, no state):

Name	Age (if under 21)	Street Address (City, State, and ZIP Code)
_____	_____	_____
_____	_____	_____

Grandchildren (list only the children of deceased children of name, no state):

Name Age (if under 21) Street Address (City, State, and ZIP Code) Name of deceased parent of grandchild

Name	Age (if under 21)	Street Address (City, State, and ZIP Code)	Name of deceased parent of grandchild
_____	_____	_____	_____
_____	_____	_____	_____

form.pdf

This is the kind of output you might expect.

Results

CLAIM AGAINST THE UNITED STATES  
FOR  
AMOUNTS DUE IN THE CASE OF A DECEASED CREDITOR

1. I/we, the undersigned, hereby make claim as -- Relationship for amounts due from the  
(Relationship)

United States in the case of Name of decedent who died on the DAY day  
(Name of decedent)

of Month YEAR while domiciled in the State of STATE  
(Month) (Year)

2. The basis of this claim is as follows:

Basis of this claim

3. I/we have been duly appointed Executor of the estate of the deceased, as evidenced  
(Executor or Administrator)

by certificate of appointment herewith, administration having been taken out in the interest of

Name Address Relationship

(Name, address, and relationship of interested relative or relative)

and such appointment is still in full force and effect.

(If making claim as the executor or administrator of the estate of the deceased, no witnesses are required, but a short certificate of letters testamentary or of administration must be submitted.) (If you are the executor or administrator of the estate of the deceased, disregard paragraphs 4, 5, and 6.)

4. If an executor or administrator has not been or will not be appointed, the following information should be furnished. The deceased is survived by: Name

Name

Widow or widower (if none, so state):

Children (if none, so state):

Name Age (if under 21) Street Address City, State, and ZIP Code

Children Name I Age Street Address I

Children Name II Age Street Address II

Children Name III Age Street Address III

Grandchildren (list only the children of deceased children-if none, so state):

Name Age (if under 21) Street Address City, State, and ZIP Code Name of deceased parent of grandchild

Grandchildren Name I Age Grandchildren Street Address I

Grandchildren Name II Age Grandchildren Street Address II

Grandchildren Name III Age Grandchildren Street Address III

## eForm Placeholder Example

---



This example shows how to use eForm fields as placeholders for the insertion of text. In this example we simply replace each of the fields in a form with the name of that field.

First we create an ABCpdf Doc object and read in our template f

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/form.pdf");
theDoc.Font = theDoc.AddFont("Helvetica-Bold");
theDoc.FontSize = 16;
theDoc.Rect.Pin = (int)XRect.Corner.TopLeft;
```

Src

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/form.pdf");
theDoc.Font = theDoc.AddFont("Helvetica-Bold");
theDoc.FontSize = 16;
theDoc.Rect.Pin = XRect.Corner.TopLeft;
```

We iterate through each of the fields. For each field we focus on the field. We then color the rectangle light gray and draw the name of the field in dark red.

### [C#]

```
string[] theNames =
theDoc.Form.GetFieldNames();
```

```
foreach (string theName in theNames) {
    Field theField = theDoc.Form[theName];
    theField.Focus();
    theDoc.Color.String = "240 240 255";
    theDoc.FillRect();
    theDoc.Rect.Height = 16;
    theDoc.Color.String = "220 0 0";
    theDoc.AddText(theField.Name);
    theDoc.Delete(theField.ID);
}
```

---

Add

### [Visual Basic]

```
Dim theNames As String() =
theDoc.Form.GetFieldNames()
Dim theName As String
For Each theName In theNames
    Dim theField As Field =
theDoc.Form(theName)
    theField.Focus()
    theDoc.Color.String = "240 240 255"
    theDoc.FillRect()
    theDoc.Rect.Height = 16
    theDoc.Color.String = "220 0 0"
    theDoc.AddText(theField.Name)
    theDoc.Delete(theField.ID)
Next
```

Finally we save.

### [C#]

```
theDoc.Save(Server.MapPath("eform.pdf"));
```

---

Save

### [Visual Basic]

---

```
theDoc.Save(Server.MapPath("eform.pdf"))
```

Given the following document.

Standard Form 1005  
Revised 1967  
Title 4, CFR, Manual  
205-105

**CLAIM AGAINST THE UNITED STATES  
FOR  
AMOUNTS DUE IN THE CASE OF A DECEASED CREDITOR**

1. I, the undersigned, hereby make claim as \_\_\_\_\_ for amounts due from the  
(Relationship)  
United States in the case of \_\_\_\_\_ who died on the \_\_\_\_\_ day  
(Name of Decedent)  
of \_\_\_\_\_ while domiciled in the State of \_\_\_\_\_  
(Month) (Year)

2. The basis of this claim is as follows:  
\_\_\_\_\_

3. There has been duly appointed \_\_\_\_\_ of the estate of the decedent, as evidenced  
(Executor or Administrator)  
by certificate of appointment herewith, administration having been taken out in the interest of:  
\_\_\_\_\_  
\_\_\_\_\_

*(Name, address, and relationship of beneficiary as requested  
and such appointment is still in full force and effect.)*

*(If willing, such as the executor or administrator of the estate of the decedent, no releases are  
required, but a receipt certificate of delivery to beneficiary or administrator must be submitted if  
you are the executor or administrator of the estate of the decedent, as required paragraphs 4, 5,  
and 6.)*

4. If an executor or administrator has not been or will not be appointed, the following information should  
be furnished: The decedent is survived by \_\_\_\_\_  
Name

Widow or widower (if none, so state):  
Children (if none, so state):

Name	Age (if under 21)	Street Address, City, State, and ZIP Code
_____	_____	_____
_____	_____	_____
_____	_____	_____

Grandchildren (list only the children of deceased children if none, so state):  
Name Age (if under 21) Street Address, City, State, and ZIP Code Name of deceased parent of grandchild

Name	Age (if under 21)	Street Address, City, State, and ZIP Code	Name of deceased parent of grandchild
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

form.pdf

This is the kind of output you might expect.

Results

CLAIM AGAINST THE UNITED STATES  
FOR  
AMOUNTS DUE IN THE CASE OF A DECEASED CREDITOR

1. I, the undersigned, hereby make claim as Relationship for amounts due from the  
(Relationship)

United States in the case of Name of who died on the Day day  
(Name of decedent)

of Month Year while domiciled in the State of State  
(Month) (Year) (State)

2. The basis of this claim is as follows:  
**Basis of this claim**

3. This has been duly appointed Executor of the estate of the decedent, as evidenced  
(Executor or Administrator)

by certificate of appointment or bench warrant, administration having been taken out in the interest of:

**Name Address Relation**

(Check, return, and submit any information requested)

and such appointment is valid in full force and effect.

(If filing such as the executor or administrator of the estate of the decedent, no return can be  
required, but a final certificate of letters testamentary or of administration must be submitted if  
you are the executor or administrator of the estate of the decedent, although paragraphs 4, 5,  
and 6.)

4. If an executor or administrator has not been or will not be appointed, the following information should  
be furnished: The decedent is survived by **Name I**

Name

Widow or widower of (name, no state)

Children of (name, no state)

Name Age (if under 21) Street Address City, State, and ZIP Code

**Children Name I** **Chrl** **Street Address I**

**Children Name II** **Chrl** **Street Address II**

**Children Name III** **Chrl** **Street Address III**

Grandchildren (fill only the children of deceased children of name, no state)

Name Age (if under 21) Street Address City, State, and ZIP Code Name of deceased parent of grandchild

**Grandchildren** **Gran** **Grandchildren Street Address I**

**Grandchildren** **Gran** **Grandchildren Street Address II**

**Grandchildren** **Gran** **Grandchildren Street Address III**

eform.pdf

# eForm Stamp Example

---



This example shows how to stamp eForm fields into a document so that the values are indelibly marked

First we create an ABCpdf Doc object and read in our template f

## [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/form.pdf");
theDoc.Font = theDoc.AddFont("Helvetica-Bold");
```

---

Src

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/form.pdf");
theDoc.Font = theDoc.AddFont("Helvetica-Bold")
```

We set the values of selected fields and then we stamp all field values into the document.

## [C#]

```
theDoc.Form["Day"].Value = "23";
theDoc.Form["Month"].Value = "February";
theDoc.Form["Year"].Value = "2005";
theDoc.Form["State"].Value = "Arizona";
theDoc.Form.Stamp();
```

---

Add

## [Visual Basic]

```
theDoc.Form("Day").Value = "23"  
theDoc.Form("Month").Value = "February"  
theDoc.Form("Year").Value = "2005"  
theDoc.Form("State").Value = "Arizona"  
theDoc.Form.Stamp()
```

Finally we save.

#### **[C#]**

```
theDoc.Save(Server.MapPath("eformstamp.pdf"));
```

---

Save

#### **[Visual Basic]**

```
theDoc.Save(Server.MapPath("eformstamp.pdf"))
```

Given the following document.

Revised Form 1057  
January 2007  
Title 4, Code of Federal Regulations  
205.107

CLAIM AGAINST THE UNITED STATES  
FOR  
AMOUNTS DUE IN THE CASE OF A DECEASED CREDITOR

1. I, the undersigned, hereby make claim as \_\_\_\_\_ for amounts due from the

(Relationship)

United States in the case of \_\_\_\_\_ who died on the \_\_\_\_\_ day

(Date of Decedent)

of \_\_\_\_\_ while domiciled in the State of \_\_\_\_\_

(Month)

(Year)

2. The basis of this claim is as follows:

3. I have been duly appointed \_\_\_\_\_ of the estate of the decedent, as evidenced

(Executor or Administrator)

by certificate of appointment herewith, administration having been taken out in the interest of:

(Class, status, and substance of beneficiaries intended)

and such appointment is still in full force and effect.

(If willing to act as the executor or administrator of the estate of the decedent, no return on this required, but a final certificate of letters testamentary or of administration must be submitted. If you are the executor or administrator of the estate of the decedent, attach paragraphs 4, 5, and 6.)

4. If an executor or administrator has not been or will not be appointed, the following information should be furnished: The decedent is survived by \_\_\_\_\_

Name

Widow or widower of (name, so state)

Children (if more, so state)

Name

Age (if under 21)

Street Address (City, State, and ZIP Code)

Grandchildren (list only the children of deceased children of name, so state)

Name Age (if under 21) Street Address (City, State, and ZIP Code) Name of deceased parent of grandchild

form.pdf

Results

This is the kind of output you might expect.

CLAIM AGAINST THE UNITED STATES  
FOR  
AMOUNTS DUE IN THE CASE OF A DECEASED CREDITOR

1. I, the undersigned, hereby make claim as \_\_\_\_\_ for amounts due from the  
(Relationship)

United States in the case of \_\_\_\_\_ who died on the 22 day  
(Name of decedent)

of February 2005 while domiciled in the State of Arizona.  
(Month) (Year)

2. The basis of this claim is as follows:  
\_\_\_\_\_

3. I have been duly appointed \_\_\_\_\_ of the estate of the decedent, as evidenced  
(Executor or Administrator)

by certificate of appointment herewith, administration having been taken out in the interest of:

\_\_\_\_\_

(Class, status, and number of beneficiaries entitled)

and such appointment is still in full force and effect.

(If willing to act as the executor or administrator of the estate of the decedent, no other person was  
appointed, but a court certificate of letters testamentary or of administration must be submitted. If  
you are the executor or administrator of the estate of the decedent, attach said paragraphs 4, 5,  
and 6.)

4. If an executor or administrator has not been or will not be appointed, the following information should  
be furnished: The decedent is survived by \_\_\_\_\_

Name

Widow or widower of (name, so state)

Children (if more, so state)

Name	Age (if under 21)	Street Address, City, State, and ZIP Code
_____	_____	_____
_____	_____	_____

Grandchildren (list only the children of deceased children of name, so state)

Name	Age (if under 21)	Street Address, City, State, and ZIP Code	Name of deceased parent of grandchild
_____	_____	_____	_____
_____	_____	_____	_____

eformstamp.pdf

# eForm FDF Example

---



This example shows how to extract Unicode annotation values from an eForm FDF file.

First we create an ABCpdf Doc object and read in our FDF file

## [C#]

```
Doc theFDF = new Doc();  
theFDF.Read(Server.MapPath("../Rez/form.fdf"))
```

---

Src

## [Visual Basic]

```
Dim theFDF As Doc = New Doc()  
theFDF.Read(Server.MapPath("../Rez/form.fdf"))
```

We find out how many items there are in the FDF file and prepare to iterate through them.

## [C#]

```
string theValues = "";  
int theLastID =  
Convert.ToInt32(theFDF.GetInfo(0,  
"Count"));
```

---

Dest

## [Visual Basic]

```
Dim theValues As String = ""  
Dim theLastID As Integer =  
Convert.ToInt32(theFDF.GetInfo(0,
```

```
"Count"))
```

We go through each item. We check to see if it is an annotation. If it is we check to see if the annotation type is text. If we have found a text annotation we extract the content and add the value to our list.

### [C#]

```
// extract annotation values (for
insertion into PDF)
for (int i = 0; i <= theLastID; i++) {
    string theType = theFDF.GetInfo(i,
"Type");
    if (theType == "anno") {
        if (theFDF.GetInfo(i, "SubType") ==
"Text") {
            string theCont;
            theCont = theFDF.GetInfo(i,
"Contents");
            theValues = theValues + theCont +
"\r\n\r\n";
        }
    }
}
// extract field values (for demonstration
purposes)
for (int i = 0; i <= theLastID; i++) {
    int theN = theFDF.GetInfoInt(i,
"/FDF*/Fields*:Count");
    for (int j = 0; j < theN; j++) {
        string theName = theFDF.GetInfo(i,
"/FDF*/Fields*[" + j + "]/T:Text");
        string theValue = theFDF.GetInfo(i,
"/FDF*/Fields*[" + j + "]/V:Text");
```

```
    // here we would do something with the
    field value we've found
  }
}
```

---

Add

### [Visual Basic]

```
For i As Integer = 0 To theLastID
  Dim theType As String
  theType = theFDF.GetInfo(i, "Type")
  If theType = "anno" Then
    If theFDF.GetInfo(i, "SubType") =
    "Text" Then
      Dim theCont As String
      theCont = theFDF.GetInfo(i,
    "Contents")
      theValues = theValues + theCont +
    vbCrLf + vbCrLf
    End If
  End If
Next
' extract field values (for demonstration
purposes)
For i As Integer = 0 To theLastID
  Dim theN As Integer
  theN = theFDF.GetInfoInt(i,
"/FDF*/Fields*:Count")
  Dim j As Integer
  For j = 0 To [theN] - 1
    Dim theName As String =
theFDF.GetInfo(i, "/FDF*/Fields*[" + j +
"]*/T:Text")
    Dim theValue As String =
theFDF.GetInfo(i, "/FDF*/Fields*[" + j +
"]*/V:Text")
    ' here we would do something with the
    field value we've found
```

```
Next j  
Next i
```

Finally we add the Unicode text to a new document and save it.

### **[C#]**

```
Doc theDoc = new Doc();  
theDoc.Font = theDoc.EmbedFont("Arial",  
LanguageType.Unicode, false, true);  
theDoc.FontSize = 96;  
theDoc.Rect.Inset(10, 10);  
theDoc.AddText(theValues);  
theDoc.Save(Server.MapPath("fdf.pdf"));
```

---

Save

### **[Visual Basic]**

```
Dim theDoc As Doc = New Doc()  
theDoc.Font = theDoc.EmbedFont("Arial",  
LanguageType.Unicode, False, True)  
theDoc.FontSize = 96  
theDoc.Rect.Inset(10, 10)  
theDoc.AddText(theValues)  
theDoc.Save(Server.MapPath("fdf.pdf"))
```

This is the kind of PDF you might expect to produce.

Cyrillic лная

Greek Μεημ

Arabic ل ت ط

Hebrew ש ו א ג

---

Results

fdf.pdf



## Advanced Graphics Example

---

Virtually all the drawing operations you will require are achievable using methods such as [FrameRect](#), [FillRect](#), [AddLine](#) and [AddArc](#).

---

### Intro

However occasionally you may require more sophisticated control over your drawing operations. In these situations you need direct access to the PDF Content Stream.

Page content is defined by the page Content Stream. The Content Stream is a sequence of descriptions of graphics objects to be placed on the page. ABCpdf allows you to create or modify these content streams allowing you access to the full power of PDF drawing operators.

---

### Content

It was intentionally decided not to encapsulate this type of drawing within a closed API. Instead these examples are provided as source code. This allows you to adapt the classes to your needs.

You can find the full project and classes under the ABCpdf menu item.

Here we describe how to perform common tasks. We do not cover the entire range of possible operators and functions. For full details you should see the [Adobe PDF Specification](#).

A path object is a shape made up of straight lines,

rectangles and Bézier curves. It may intersect itself and may have disconnected sections and holes. After the path has been defined it may be painted, filled, used for clipping or a combination of these operations.

Each path is constructed of one or more subpaths. Each subpath is constructed of one or more connected segments. Subpaths may be open or closed. When a subpath is closed the start of the path is connected to the end.

All paths are located in the standard [Adobe PDF coordinate space](#). The following is a list of standard path construction operators.

Name	Parameters	Operator	Description
Move	x y	m	Begin a new subpath by moving to the coordinates specified.
Line	x y	l	Add a straight line from the current location to the coordinates specified.
Rect	x y w h	re	Add a rectangular subpath with the lower left corner at (x, y) with width w and height h.
Bézier	x1 y1 x2 y2 x3 y3	c	Add a Bézier curve from the current location to the coordinates specified (x3, y3) using the other coordinates (x1, y1) and (x2, y2) as control points.
Close		h	Close the current subpath joining the start

			to the end.
Stroke		S	Paint a line along the current path using the current stroke color.
Fill		f	<p>Fill the current path using the current fill color.</p> <p>This fill method uses the nonzero winding number rule. There are other PDF operators to allow the use of the even-odd rule but these are not generally useful.</p>
Clip		W n	<p>Intersect the path with the current clipping path to establish a new clipping path.</p> <p>This actually comprises two operators rather than one but they are almost invariably used in this combination.</p>

The graphics state defines the parameters within which the PDF operators work. For example the graphics state defines the current line width which will be used whenever a line is drawn. It also defines the current non-stroking color which will be used when a path is filled.

You can push copies of the graphics state onto a stack and then restore them later. This can be very useful for doing and undoing graphics state operations.

Name	Parameters	Operator	Description
Save State		q	Push a copy of the current graphics state onto the stack.
Restore State		Q	Restore the current graphics state from the top of the stack.
SetLineWidth	w	w	Set the width to be used when stroking lines.
SetGrayStrokeColor	w	G	Set the gray level to use for stroking operations. The component ranges between 0.0 and 1.0 (black and white respectively).
SetGrayNonStrokeColor	w	g	The same as G but for non-stroking operations.
SetRGBStrokeColor	r g b	RG	Set the RGB color to use for stroking operations. Each

			component ranges between 0.0 and 1.0.
SetRGBNonStrokeColor	r g b	rg	The same as SetRGBStrokeColor but for non-stroking operations.
SetCMYKStrokeColor	c m y k	K	Set the CMYK color to use for stroking operations. Each component ranges between 0.0 and 1.0.
SetCMYKNonStrokeColor	c m y k	k	The same as SetCMYKStrokeColor but for non-stroking operations.
			<p>Concatenate the given matrix with the current transform matrix. Common transforms include:</p> <ul style="list-style-type: none"> <li>• Translate: A matrix of the form <math>\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 \\ 0 &amp; 1 &amp; 0 &amp; 0 \\ 0 &amp; 0 &amp; 1 &amp; tx \\ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}</math> shifts the coordinate system by tx.</li> </ul>

Transform

a b c d e f

cm

- horizontal and vertical
- Scaling matrix of the form  $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$  scales the coordinate system a factor  $s_x$  horizontally and  $s_y$  vertically. Pinned at the origin.
  - Rotation matrix of the form  $\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$  rotates the coordinate system the angle  $\theta$  anti-clockwise around origin.
  - Skew:  $A$  matrix of the form  $\begin{bmatrix} 1 & \tan(\theta) \\ 0 & 1 \end{bmatrix}$

State

			<p><math>\tan(\text{ra})</math> : 0] skew the x ar axes by angle ra</p>
<p>SetLineCap</p>	<p>v</p>	<p>J</p>	<p>The line cap the ends of a lines to be stroked. Possible val are:</p> <ol style="list-style-type: none"><li>0. Butt. Th stroke is square the end the path and doe not proj beyond end of t path.</li><li>1. Round. semicirc is added the end the path projecti beyond endpoir</li><li>2. Projecti Square. The stro is squar but proj a distan</li></ol>

			of half the line width beyond ends of path.
SetLineJoin	v	j	<p>The line join the shape of joints between connected segments of path. Possible values are:</p> <p>0. Miter. The outer edges of the two segments are extended until they meet. This is the same way that wooden segments are joined to make a picture frame. If the segments meet at a steep angle, a bevel</p>

			<p>is used instead. The precise offset point is called the Miter Line (see below).</p> <ol style="list-style-type: none"> <li>1. Round. A pie slice is added to the junction of the two segments to produce a round corner.</li> <li>2. Bevel. The two segments are finished with butt caps and any notch between the two is filled in.</li> </ol>
			<p>The maximum length of mitered line joins for path</p> <p>The miter line is expressed in terms of the</p>

SetMiterLimit	v	M	<p>ratio of the thickness of line to the thickness of join.</p> <p>For example value of 1.5 allow the width of the line at join to be up one and a half times the thickness of width of an individual line segment.</p>
LineDash	a p	d	<p>The dash pattern to use for stroked lines. The parameters include a - a array for the pattern and the phase of dash.</p>

Paths can be stroked (drawn) using the current stroking color.

For example you might wish to draw a star.

### **[C#]**

```
PDFContent theContent = new
PDFContent(theDoc);
theContent.SaveState();
theContent.SetLineWidth(30);
theContent.SetLineJoin(2);
theContent.Move(124, 158);
theContent.Line(300, 700);
theContent.Line(476, 158);
theContent.Line(15, 493);
theContent.Line(585, 493);
theContent.Close();
theContent.Stroke();
theContent.RestoreState();
theContent.AddToDoc();
```

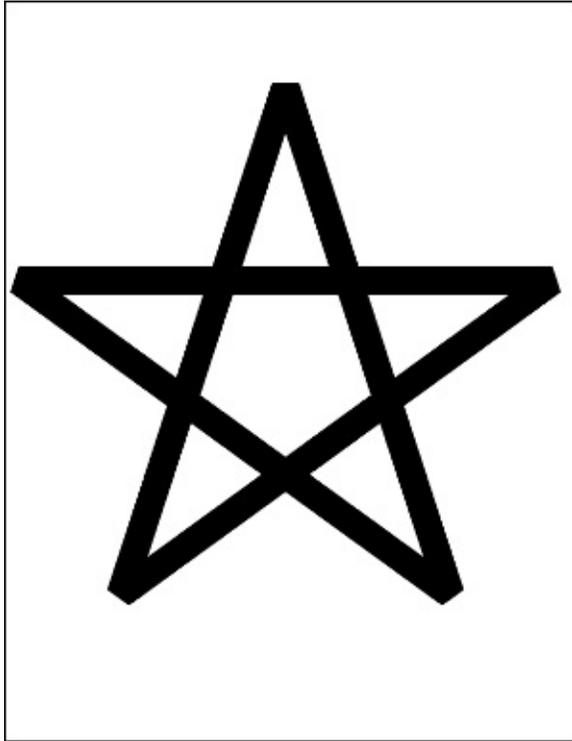
### **[Visual Basic]**

```
Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SaveState()
theContent.SetLineWidth(30)
theContent.SetLineJoin(2)
theContent.Move(124, 158)
theContent.Line(300, 700)
theContent.Line(476, 158)
theContent.Line(15, 493)
theContent.Line(585, 493)
```

---

Stroke

```
theContent.Close()  
theContent.Stroke()  
theContent.RestoreState()  
theContent.AddToDoc()
```



Paths can be filled with the current non-stroking color.

For example you might wish to construct a filled star.

### **[C#]**

```
PDFContent theContent = new  
PDFContent(theDoc);  
theContent.SaveState();  
theContent.SetLineWidth(30);  
theContent.SetLineJoin(2);
```

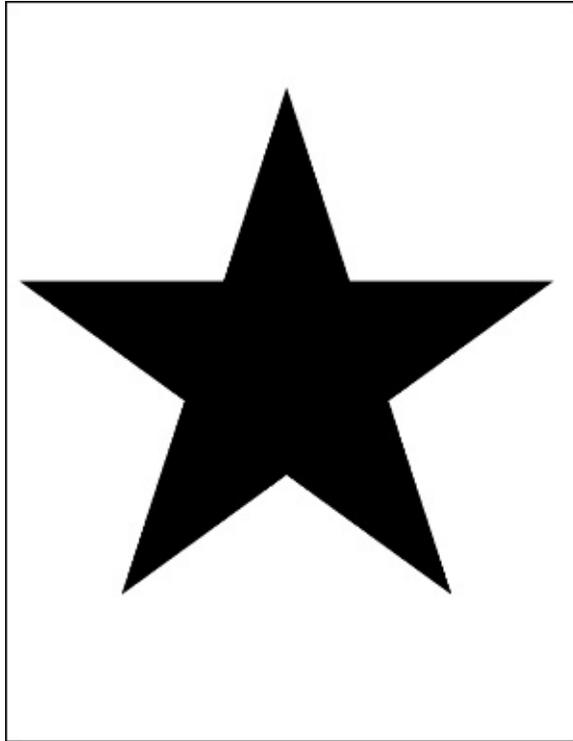
```
theContent.Move(124, 158);  
theContent.Line(300, 700);  
theContent.Line(476, 158);  
theContent.Line(15, 493);  
theContent.Line(585, 493);  
theContent.Close();  
theContent.Fill();  
theContent.RestoreState();  
theContent.AddToDoc();
```

### [Visual Basic]

```
Dim theContent As PDFContent = New  
PDFContent(theDoc)  
theContent.SaveState()  
theContent.SetLineWidth(30)  
theContent.SetLineJoin(2)  
theContent.Move(124, 158)  
theContent.Line(300, 700)  
theContent.Line(476, 158)  
theContent.Line(15, 493)  
theContent.Line(585, 493)  
theContent.Close()  
theContent.Fill()  
theContent.RestoreState()  
theContent.AddToDoc()
```

---

Fill



Paths can contain curved segments.

Curved segments are specified as cubic Bézier curves. These provide a flexible and practical way to draw curves and curved paths.

Each segment is defined by four points. The first point and the final point define the ends of the segment. The second and third points define the control points. The line is pulled towards the first control point as it leaves the start and it is pulled towards the second control point as it arrives at the end.

The easiest way to illustrate this is with an example. Note that in this example the Bézier curve takes relatively little code to define. Most of the code is related to illustrating how the control points affect

the shape of the curve.

### [C#]

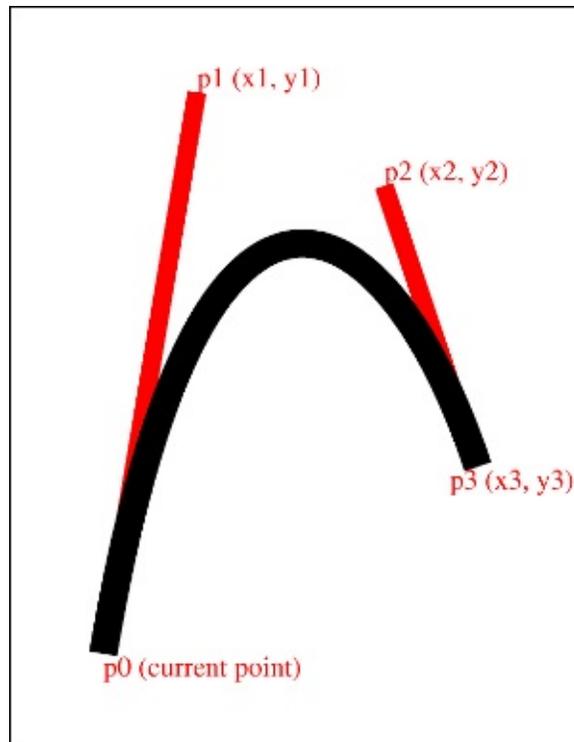
```
PDFContent theContent = new
PDFContent(theDoc);
theContent.SaveState();
theContent.SetLineWidth(30);
theContent.Move(100, 50);
theContent.Bezier(200, 650, 400,
550, 500, 250);
theContent.Stroke();
theContent.RestoreState();

// annotate Bezier curve in red
theDoc.Color.String = "255 0 0";
theDoc.Width = 20;
theDoc.FontSize = 30;
theDoc.Pos.String = "100 50";
theDoc.AddText("p0 (current
point)");
theDoc.Pos.String = "200 650";
theDoc.Pos.Y = theDoc.Pos.Y +
theDoc.FontSize;
theDoc.AddText("p1 (x1, y1)");
theDoc.Pos.String = "400 550";
theDoc.Pos.Y = theDoc.Pos.Y +
theDoc.FontSize;
theDoc.AddText("p2 (x2, y2)");
theDoc.Pos.String = "500 250";
theDoc.Pos.X = theDoc.Pos.X -
theDoc.FontSize;
theDoc.AddText("p3 (x3, y3)");
theDoc.AddLine(100, 50, 200, 650);
theDoc.AddLine(400, 550, 500, 250);
theContent.AddToDoc();
```

**[Visual Basic]**

```
Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SaveState()
theContent.SetLineWidth(30)
theContent.Move(100, 50)
theContent.Bezier(200, 650, 400,
550, 500, 250)
theContent.Stroke()
theContent.RestoreState()
```

```
' annotate Bezier curve in red
theDoc.Color.String = "255 0 0"
theDoc.Width = 20
theDoc.FontSize = 30
theDoc.Pos.String = "100 50"
theDoc.AddText("p0 (current point)")
theDoc.Pos.String = "200 650"
theDoc.Pos.Y = theDoc.Pos.Y +
theDoc.FontSize
theDoc.AddText("p1 (x1, y1)")
theDoc.Pos.String = "400 550"
theDoc.Pos.Y = theDoc.Pos.Y +
theDoc.FontSize
theDoc.AddText("p2 (x2, y2)")
theDoc.Pos.String = "500 250"
theDoc.Pos.X = theDoc.Pos.X -
theDoc.FontSize
theDoc.AddText("p3 (x3, y3)")
theDoc.AddLine(100, 50, 200, 650)
theDoc.AddLine(400, 550, 500, 250)
theContent.AddToDoc()
```



You can use a path to define a clipping area.

The graphics state holds a clipping path that restricts the areas on the page which can be painted on. Marks falling within the clipping area will be displayed and those falling outside will not.

The default clipping path is the entire page. You can intersect the current clipping path with a new path using the clipping path operators.

You cannot expand a clipping path. Instead you must save the graphics state before applying your clipping path and then restore the graphics state after you have finished using it.

Here we fill a rectangle clipped by our star shape.

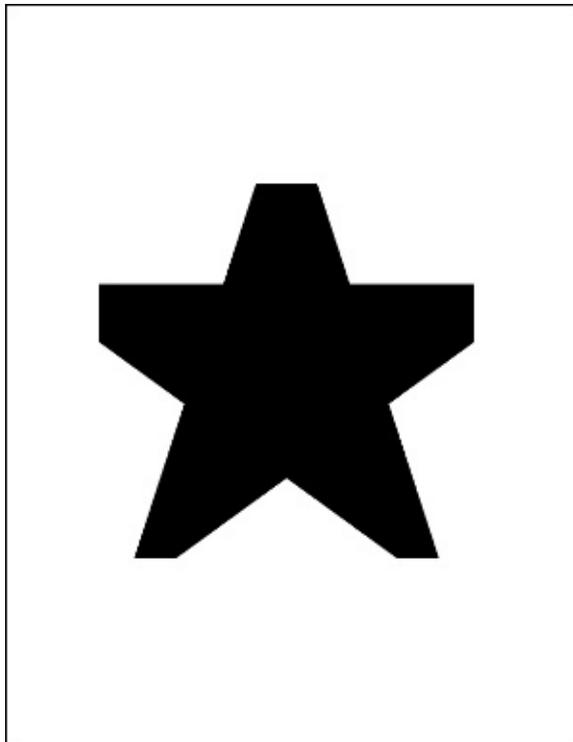
## [C#]

```
PDFContent theContent = new
PDFContent(theDoc);
theContent.SaveState();
theContent.SetLineWidth(30);
theContent.SetLineJoin(2);
theContent.Move(124, 158);
theContent.Line(300, 700);
theContent.Line(476, 158);
theContent.Line(15, 493);
theContent.Line(585, 493);
theContent.Clip();
theContent.Rect(100, 200, 400, 400);
theContent.Fill();
theContent.RestoreState();
theContent.AddToDoc();
```

## [Visual Basic]

Clip

```
Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SaveState()
theContent.SetLineWidth(30)
theContent.SetLineJoin(2)
theContent.Move(124, 158)
theContent.Line(300, 700)
theContent.Line(476, 158)
theContent.Line(15, 493)
theContent.Line(585, 493)
theContent.Clip()
theContent.Rect(100, 200, 400, 400)
theContent.Fill()
theContent.RestoreState()
theContent.AddToDoc()
```



You can define the way that the end points of your paths are capped.

The following options are available:

Butt. The stroke is square at the end of the path and does not project beyond the end of the path.

Round. A semicircle is added at the end of the path projecting beyond the endpoints.

Projecting Square. The stroke is square but projects a distance of half the line width beyond the ends of the path.

This example shows how different line caps are drawn. Note that the line cap themselves take relatively little code to define. Most of the code is

related to annotating the drawing so that you can see how the caps relate to the end points.

## [C#]

```
PDFContent theContent = new
PDFContent(theDoc);
theContent.SaveState();
theContent.SetLineWidth(100);
theContent.SetLineCap(0);
theContent.Move(100, 600);
theContent.Line(500, 600); // line
theContent.Stroke();

theContent.SetLineCap(1); // round
cap
theContent.Move(100, 400);
theContent.Line(500, 400);
theContent.Stroke();

theContent.SetLineCap(2);
theContent.Move(100, 200);
theContent.Line(500, 200);
theContent.Stroke();

// add capped lines
theContent.AddToDoc();

// annotate capped lines
theDoc.FontSize = 48;
theDoc.Pos.String = "50 720";
theDoc.AddText("0 - Butt Cap");
theDoc.Pos.String = "50 520";
theDoc.AddText("1 - Round Cap");
theDoc.Pos.String = "50 320";
int id = theDoc.AddText("2 -
Projecting Square Cap");
theDoc.Width = 20;
```

```
theDoc.Color.String = "255 255 255";
theDoc.AddLine(100, 200, 500, 200);
theDoc.Rect.String = "80 180 120
220";
theDoc.FillRect(20, 20);
theDoc.Rect.String = "480 180 520
220";
theDoc.FillRect(20, 20);
theDoc.AddLine(100, 400, 500, 400);
theDoc.Rect.String = "80 380 120
420";
theDoc.FillRect(20, 20);
theDoc.Rect.String = "480 380 520
420";
theDoc.FillRect(20, 20);
theDoc.AddLine(100, 600, 500, 600);
theDoc.Rect.String = "80 580 120
620";
theDoc.FillRect(20, 20);
theDoc.Rect.String = "480 580 520
620";
theDoc.FillRect(20, 20);
theDoc.Color.String = "0 0 0";
```

---

Caps

### [Visual Basic]

```
Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SaveState()
theContent.SetLineWidth(100)
theContent.SetLineCap(0)
theContent.Move(100, 600)
theContent.Line(500, 600) ' line
theContent.Stroke()

theContent.SetLineCap(1) ' round cap
```

```
theContent.Move(100, 400)
theContent.Line(500, 400)
theContent.Stroke()

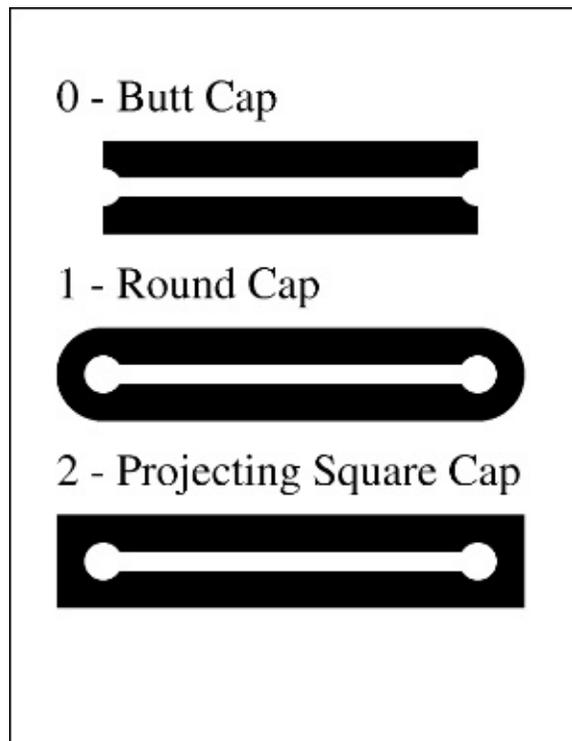
theContent.SetLineCap(2)
theContent.Move(100, 200)
theContent.Line(500, 200)
theContent.Stroke()

' add capped lines
theContent.AddToDoc()

' annotate capped lines
theDoc.FontSize = 48
theDoc.Pos.String = "50 720"
theDoc.AddText("0 - Butt Cap")
theDoc.Pos.String = "50 520"
theDoc.AddText("1 - Round Cap")
theDoc.Pos.String = "50 320"
int id = theDoc.AddText("2 -
Projecting Square Cap")
theDoc.Width = 20

theDoc.Color.String = "255 255 255"
theDoc.AddLine(100, 200, 500, 200)
theDoc.Rect.String = "80 180 120
220"
theDoc.FillRect(20, 20)
theDoc.Rect.String = "480 180 520
220"
theDoc.FillRect(20, 20)
theDoc.AddLine(100, 400, 500, 400)
theDoc.Rect.String = "80 380 120
420"
theDoc.FillRect(20, 20)
theDoc.Rect.String = "480 380 520
420"
```

```
theDoc.FillRect(20, 20)
theDoc.AddLine(100, 600, 500, 600)
theDoc.Rect.String = "80 580 120
620"
theDoc.FillRect(20, 20)
theDoc.Rect.String = "480 580 520
620"
theDoc.FillRect(20, 20)
theDoc.Color.String = "0 0 0"
```



You can define the way that your line segments are joined.

The following options are available:

Miter. The outer edges for the two segments are extended until they meet. This is the same way that

wooden segments are joined to make a picture frame. If the segments meet at an overly steep angle a bevel join is used instead. The precise cut-off point is called the Miter Limit.

Round. A pie slice is added to the junction of the two segments to produce a rounded corner.

Bevel. The two segments are finished with butt caps and any notch between the two is filled in.

This example shows how different line joins are drawn. Note that the line joins themselves take relatively little code to define. Most of the code is related to annotating the drawing.

### [C#]

```
PDFContent theContent = new
PDFContent(theDoc);
theContent.SetLineWidth(50);
theContent.SetLineJoin(0);
theContent.Move(300, 500);
theContent.Line(400, 700);
theContent.Line(500, 500);
theContent.Stroke();

theContent.SetLineJoin(1);
theContent.Move(300, 300);
theContent.Line(400, 500);
theContent.Line(500, 300);
theContent.Stroke();

theContent.SetLineJoin(2);
theContent.Move(300, 100);
theContent.Line(400, 300);
theContent.Line(500, 100);
theContent.Stroke();
theContent.AddToDoc();
```

```
theDoc.FontSize = 48;
theDoc.Pos.String = "50 700";
theDoc.AddText("0 - Miter");
theDoc.Pos.String = "50 500";
theDoc.AddText("1 - Round ");
theDoc.Pos.String = "50 300";
theDoc.AddText("2 - Bevel");
theDoc.Width = 10;
theDoc.Color.String = "255 255 255";
theDoc.AddLine(300, 500, 400, 700);
theDoc.AddLine(400, 700, 500, 500);
theDoc.Rect.String = "390 690 410
710";
theDoc.FillRect(10, 10);
theDoc.AddLine(300, 300, 400, 500);
theDoc.AddLine(400, 500, 500, 300);
theDoc.Rect.String = "390 490 410
510";
theDoc.FillRect(10, 10);
theDoc.AddLine(300, 100, 400, 300);
theDoc.AddLine(400, 300, 500, 100);
theDoc.Rect.String = "390 290 410
310";
theDoc.FillRect(10, 10);
theDoc.Color.String = "0 0 0";
```

---

Joins

### [Visual Basic]

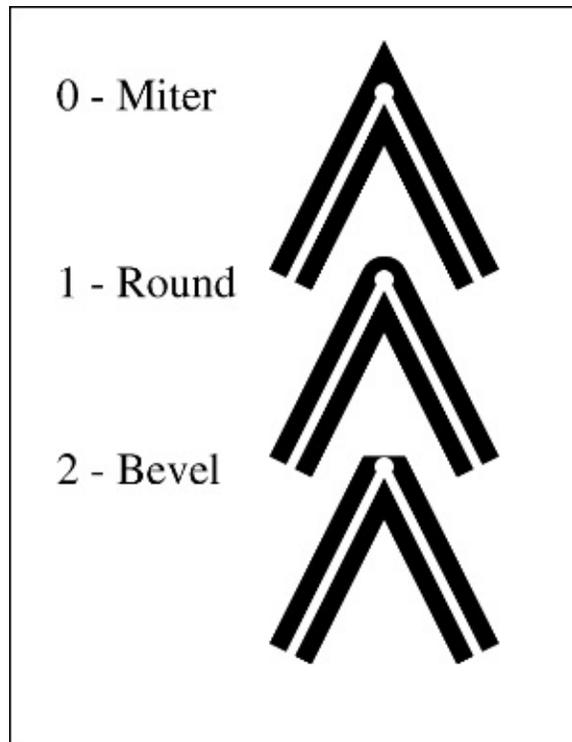
```
Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SetLineWidth(50)
theContent.SetLineJoin(0)
theContent.Move(300, 500)
theContent.Line(400, 700)
theContent.Line(500, 500)
theContent.Stroke()
```

```
theContent.SetLineJoin(1)
theContent.Move(300, 300)
theContent.Line(400, 500)
theContent.Line(500, 300)
theContent.Stroke()

theContent.SetLineJoin(2)
theContent.Move(300, 100)
theContent.Line(400, 300)
theContent.Line(500, 100)
theContent.Stroke()
theContent.AddToDoc()

theDoc.FontSize = 48
theDoc.Pos.String = "50 700"
theDoc.AddText("0 - Miter")
theDoc.Pos.String = "50 500"
theDoc.AddText("1 - Round ")
theDoc.Pos.String = "50 300"
theDoc.AddText("2 - Bevel")
theDoc.Width = 10
theDoc.Color.String = "255 255 255"
theDoc.AddLine(300, 500, 400, 700)
theDoc.AddLine(400, 700, 500, 500)
theDoc.Rect.String = "390 690 410
710"
theDoc.FillRect(10, 10)
theDoc.AddLine(300, 300, 400, 500)
theDoc.AddLine(400, 500, 500, 300)
theDoc.Rect.String = "390 490 410
510"
theDoc.FillRect(10, 10)
theDoc.AddLine(300, 100, 400, 300)
theDoc.AddLine(400, 300, 500, 100)
theDoc.Rect.String = "390 290 410
310"
```

```
theDoc.FillRect(10, 10)
theDoc.Color.String = "0 0 0"
```



You can define dashed lines.

The dash pattern is specified by a dash array and a dash phase. The dash array specifies the length of dashes and gaps. The dash phase specifies the distance into the array at which the line dashes should start.

When the lengths in the array are exhausted the dash pattern starts again at the beginning. You can use an empty array and zero phase to specify a solid line

Dashed lines can be applied to any kind of path including straight lines and curves. Each subpath in

a path is treated separately - the dash phase starts at the beginning again.

This example shows how different line dash patterns are drawn.

### [C#]

```
PDFContent theContent = new
PDFContent(theDoc);
theContent.SaveState();
theContent.SetLineWidth(20);
theContent.LineDash("[ ] 0");
theContent.Move(100, 650);
theContent.Line(500, 650);
theContent.Stroke();

theContent.LineDash("[ 90 ] 0");
theContent.Move(100, 500);
theContent.Line(500, 500);
theContent.Stroke();

theContent.LineDash("[ 60 ] 30");
theContent.Move(100, 350);
theContent.Line(500, 350);
theContent.Stroke();

theContent.LineDash("[ 60 30 ] 0");
theContent.Move(100, 200);
theContent.Line(500, 200);
theContent.Stroke();
theContent.RestoreState();

// annotate dashed lines
theDoc.Color.String = "0 0 0";
theDoc.FontSize = 36;
theDoc.Pos.String = "50 710";
theDoc.AddText("[ ] 0 - no dashes");
```

```
theDoc.Pos.String = "50 560";
theDoc.AddText("[ 90 ] 0 - 90 on, 90
off...");
theDoc.Pos.String = "50 410";
theDoc.AddText("[ 60 ] 30 - 30 on,
60 off, 60 on...");
theDoc.Pos.String = "50 260";
theDoc.AddText("[ 60 30 ] 0 - 60 on,
30 off, 60 on...");

// add dashed lines
theContent.AddToDoc();
```

---

Dash

### [Visual Basic]

```
Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SaveState()
theContent.SetLineWidth(20)
theContent.LineDash("[ ] 0")
theContent.Move(100, 650)
theContent.Line(500, 650)
theContent.Stroke()

theContent.LineDash("[ 90 ] 0")
theContent.Move(100, 500)
theContent.Line(500, 500)
theContent.Stroke()

theContent.LineDash("[ 60 ] 30")
theContent.Move(100, 350)
theContent.Line(500, 350)
theContent.Stroke()

theContent.LineDash("[ 60 30 ] 0")
theContent.Move(100, 200)
theContent.Line(500, 200)
```

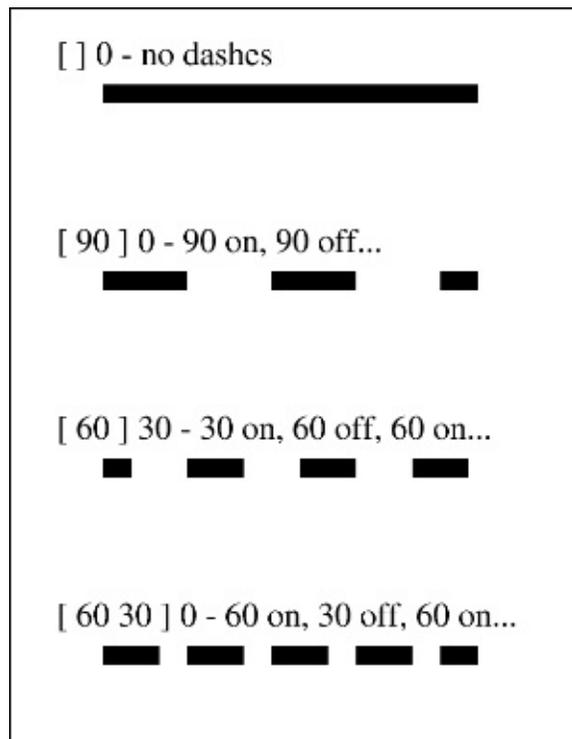
```

theContent.Stroke()
theContent.RestoreState()

' annotate dashed lines
theDoc.Color.String = "0 0 0"
theDoc.FontSize = 36
theDoc.Pos.String = "50 710"
theDoc.AddText("[ ] 0 - no dashes")
theDoc.Pos.String = "50 560"
theDoc.AddText("[ 90 ] 0 - 90 on, 90
off...")
theDoc.Pos.String = "50 410"
theDoc.AddText("[ 60 ] 30 - 30 on,
60 off, 60 on...")
theDoc.Pos.String = "50 260"
theDoc.AddText("[ 60 30 ] 0 - 60 on,
30 off, 60 on...")

' add dashed lines
theContent.AddToDoc()

```



You can define transforms which affect the world space.

A transform allows you to translate, scale, rotate, or skew objects. Multiple transforms can be concatenated so that you can perform a combination of these operations.

A transform is defined by six numbers. Common transforms include:

Translation: A matrix of the form  $[1 \ 0 \ 0 \ 1 \ t_x \ t_y]$  shifts the coordinate system by  $t_x$  horizontally and  $t_y$  vertically.

Scaling: A matrix of the form  $[s_x \ 0 \ 0 \ 0 \ 0 \ 0]$  scales the coordinate system by a factor of  $s_x$  horizontally and  $s_y$  vertically pinned at the origin.

Rotation: A matrix of the form  $[\cos(r_a) \ \sin(r_a) \ -\sin(r_a) \ \cos(r_a) \ 0 \ 0]$  rotates the coordinate system by the angle  $r_a$  anticlockwise around the origin.

Skew: A matrix of the form  $[1 \ \tan(r_a) \ \tan(r_a) \ 1 \ 0 \ 0]$  skews the x and y axes by the angle  $r_a$ .

This example shows how to apply a 45 degree rotation to a drawing of a star.

### [C#]

```
PDFContent star = new
PDFContent(theDoc);
star.Move(124, 108);
star.Line(300, 650);
star.Line(476, 108);
star.Line(15, 443);
```

```
star.Line(585, 443);
star.Close();
star.Stroke();

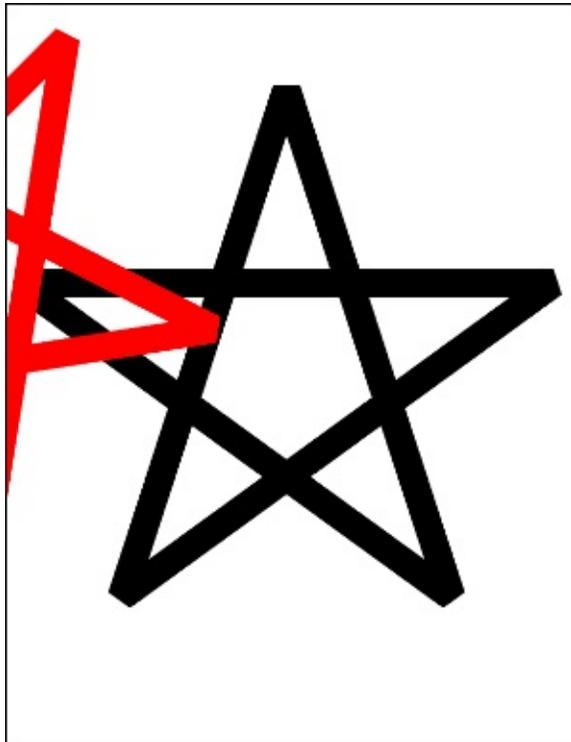
PDFContent theContent = new
PDFContent(theDoc);
theContent.SaveState();
theContent.SetLineWidth(30);
theContent.SetLineJoin(2);
theContent.AddContent(star);
theContent.SetRGBStrokeColor(1, 0,
0);
theContent.Transform(0.7, 0.7, -0.7,
0.7, 0, 0);
theContent.AddContent(star);
theContent.RestoreState();
theContent.AddToDoc();
```

### [Visual Basic]

```
Dim star As PDFContent = New
PDFContent(theDoc)
star.Move(124, 108)
star.Line(300, 650)
star.Line(476, 108)
star.Line(15, 443)
star.Line(585, 443)
star.Close()
star.Stroke()

Dim theContent As PDFContent = New
PDFContent(theDoc)
theContent.SaveState()
theContent.SetLineWidth(30)
theContent.SetLineJoin(2)
theContent.AddContent(star)
theContent.SetRGBStrokeColor(1, 0,
```

```
0)
theContent.Transform(0.7, 0.7, -0.7,
0.7, 0, 0)
theContent.AddContent(star)
theContent.RestoreState()
theContent.AddToDoc()
```



Other examples including text operators, transparency modes and blend modes can be found in the example project.

---

Other

You can find the full project and classes under the ABCpdf menu item.





## Fields, Markup and Movies Example

---

In general you will use PDF templates containing Fields and Annotations created by a designer.

---

### Intro

However occasionally you may need to create Fields and Annotations at run-time. You can perform this kind of operation using the low-level functionality within ABCpdf.

Annotations are a generic class of objects which exist outside the PDF Content Stream. Because they are independent of the PDF Content Stream they operate independently of the page. They float over the page rather than being embedded in it.

Movie Annotations, Note Annotations, Stamp Annotations, Line Annotations and Polygon Annotations are just a few of the many Annotation types which exist.

---

### Types

Fields are a specific type of Annotation combined with a name. It is the fact that the Field is an Annotation which allows you to interact with it. It is the fact that the Annotation is linked into the PDF hierarchy which allows it to take a value and be located by name.

Movies are a specific type of Annotation. You can embed a video in PDF in movie formats including Flash and WMV.

Creating these types of objects on the fly is complex. For this reason the examples here are simple summaries. You can find a full project and source classes under the ABCpdf menu item.

For full details of the way that Annotations work you should see the [Adobe PDF Specification](#).

You may wish to generate a PDF document with Fields created dynamically at run-time.

The code below creates a set of Fields including text boxes, buttons, checkboxes and signatures.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Font = theDoc.AddFont("Helvetica");
theDoc.FontSize = 36;

//Create interactive form
InteractiveForm form = new InteractiveForm
(theDoc);
theDoc.Pos.X = 40;
theDoc.Pos.Y = theDoc.MediaBox.Top - 40;
theDoc.AddText("Interactive Form annotation");

//Radio button items
form.AddRadioButtonGroup(new string[2]{"40
650", "40 660 80 700"}, "RadioGroupField",
theDoc.Pos.String = "100 696";
theDoc.AddText("RadioButton 1");
theDoc.Pos.String = "100 646";
theDoc.AddText("RadioButton 2");

//Text field 1
FormField text = form.AddTextField("40 530
580", "TextField1", "Hello World!");
text.DefaultAppearance = "/TimesRoman 36 Tf
rg";
text.BorderColor = "0 0 0";
text.FillColor = "220 220 220";
text.TextAlign = "Left";
```

---

## Fields

```
//Text field 2
text = form.AddTextField("40 460 300 510",
    "TextField2", "Text Field");
text.BorderColor = "0 0 0";
text.DefaultAppearance = "/TimesRoman 36 Tf
rg";
text.TextAlign = "Left";
text.SetFlag(FormField.FieldFlag.Password);

//Text field 3
text = form.AddTextField("320 460 370 580",
    "TextField3", "Vertical");
text.BorderColor = "0 0 0";
text.DefaultAppearance = "/TimesRoman 36 Tf
rg";
text.Rotate = 90;

//Combobox field
FormField combo = form.AddChoiceField("Comb
"40 390 300 440", "ComboBoxField");
combo.DefaultAppearance = "/TimesRoman 24 T
rg";
combo.AddOptions(new string[] {"ComboBox It
"ComboBox Item 2", "ComboBox Item 3"});

//Listbox field
FormField listbox = form.AddChoiceField("Li
"40 280 300 370", "ListBoxField");
listbox.DefaultAppearance = "/TimesRoman 24
0 rg";
listbox.AddOptions(new string[] {"ListBox I
"ListBox Item 2", "ListBox Item 3"});

//Checkbox field
form.AddCheckbox("40 220 80 260", "CheckBox
true);
```

```

theDoc.Pos.String = "100 256";
theDoc.AddText("Check Box");

//Pushbutton field
FormField button = form.AddButton("40 160 2
"ButtonField", "Button");
button.BorderColor = "0 0 0";
button.BorderStyle = "Beveled";

//Signature field
FormField signature = form.AddSignature("40
140", "Signature");
signature.BorderColor = "0 0 0";

```

Interactive Form annotations

RadioButton 1

RadioButton 2

Hello World!

\*\*\*\*\*

ComboBox Item 1

ListBox Item 1

ListBox Item 2

ListBox Item 3

Check Box

Button

Vertical

You may wish to generate a PDF document with markup created dynamically at run-time.

The code below creates a set of markup Annotations including squares, lines, text effects, circles and polygons.

### [C#]

```
//Markup annotations
theDoc.Page = theDoc.AddPage();
theDoc.Pos.X = 40;
theDoc.Pos.Y = theDoc.MediaBox.Top - 40;
theDoc.AddText("Markup annotations");

SquareAnnotation square = new
SquareAnnotation(theDoc, "40 560 300 670",
0, "0 0 255");
square.BorderWidth = 8;

LineAnnotation line = new LineAnnotation(th
"100 565 220 665", "255 0 0");
line.BorderWidth = 12;
line.RichTextCaption = "<span style= \"font
size:36pt; color:#FF0000\">Line</span>";

theDoc.FontSize = 24;
theDoc.Pos.String = "400 670";
int id = theDoc.AddText("Underline");
TextMarkupAnnotation markup = new
TextMarkupAnnotation(theDoc, id, "Underline
255 0");
theDoc.Pos.String = "400 640";
id = theDoc.AddText("Highlight");

markup = new TextMarkupAnnotation(theDoc, i
"Highlight", "255 255 0");
theDoc.Pos.String = "400 610";
id = theDoc.AddText("StrikeOut");

markup = new TextMarkupAnnotation(theDoc, i
"StrikeOut", "255 0 0");
```

---

**Markup**

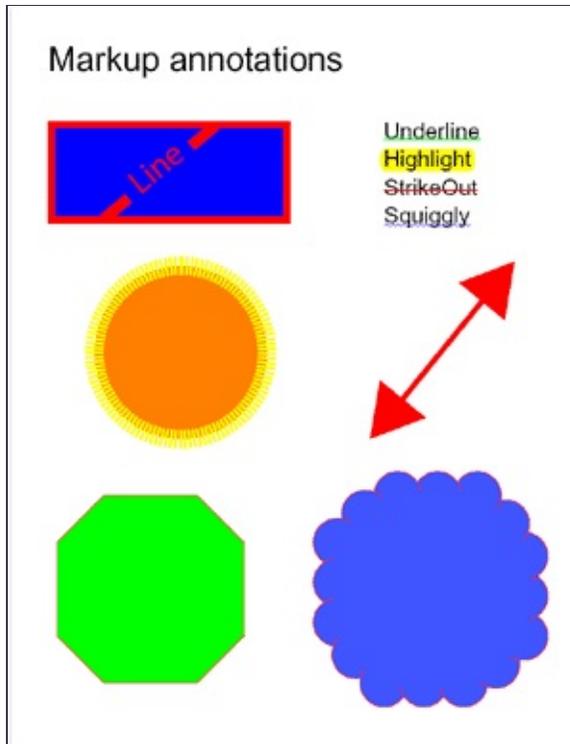
```
theDoc.Pos.String = "400 580";
id = theDoc.AddText("Squiggly");

markup = new TextMarkupAnnotation(theDoc, i
"Squiggly", "0 0 255");
theDoc.FontSize = 36;

CircleAnnotation circle = new
CircleAnnotation(theDoc, "80 320 285 525",
0, "255 128 0");
circle.BorderWidth = 20;
circle.BorderStyle = "Dashed";
circle.BorderDash = "[3 2]";

LineAnnotation arrowLine = new
LineAnnotation(theDoc, "385 330 540 520", "
0");
arrowLine.LineEndingsStyle = "ClosedArrow
ClosedArrow";
arrowLine.BorderWidth = 6;
arrowLine.FillColor = "255 0 0";

PolygonAnnotation polygon = new
PolygonAnnotation(theDoc, "100 70 50 120 50
270 200 270 250 220 250 120 200 70", "255 0
255 0");
PolygonAnnotation cloudyPolygon = new
PolygonAnnotation(theDoc, "400 70 350 120 3
400 270 500 270 550 220 550 120 500 70", "2
"64 85 255");
cloudyPolygon.CloudyEffect = 1;
```



You may wish to generate a PDF document with movies ins at run-time.

The code below inserts a Flash movie and a WMV movie.

### [C#]

```
//Movie annotations
//WMV is courtesy of NASA -
http://www.nasa.gov/wmv/30873main_cardiovas
theDoc.Page = theDoc.AddPage();
theDoc.Pos.X = 40;
theDoc.Pos.Y = theDoc.MediaBox.Top - 40;
theDoc.AddText("Multimedia features");

theDoc.FontSize = 24;

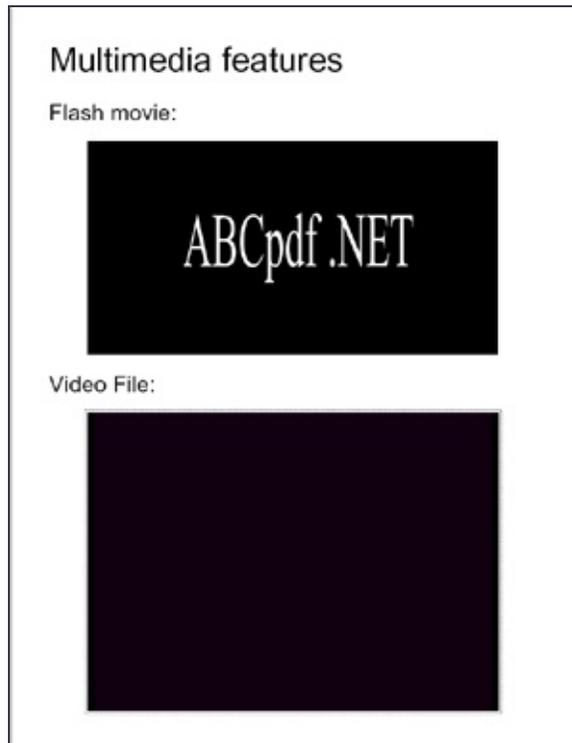
theDoc.Pos.String = "40 690";
```

---

## Movies

```
theDoc.AddText("Flash movie:");
MovieAnnotation movie1 = new MovieAnnotation(
    420 520 650", Server.MapPath("ABCpdf.swf"))

theDoc.Pos.String = "40 400";
theDoc.AddText("Video File:");
MovieAnnotation movie2 = new MovieAnnotation(
    40 520 360", Server.MapPath("video.wmv"));
```



You may wish to generate a PDF document with other types Annotation inserted dynamically at run-time.

The code below adds a sticky note, a file attachment and so stamps.

**[C#]**

```
theDoc.Page = theDoc.AddPage();
theDoc.FontSize = 36;
theDoc.Pos.X = 40;
theDoc.Pos.Y = theDoc.MediaBox.Top - 40;
theDoc.AddText("Other types of annotations"

//Sticky note annotation
theDoc.FontSize = 24;
theDoc.Pos.String = "40 680";
theDoc.AddText("Text annotation");
TextAnnotation textAnnotation = new
TextAnnotation(theDoc, "340 660 340 675", "
600 750", "6 sets of 13 pages. Trim to 5X7.

//File attachment annotation
theDoc.Pos.String = "40 640";
theDoc.AddText("File Attachment annotation"
FileAttachmentAnnotation fileAttachMent = n
FileAttachmentAnnotation(theDoc, "340 625 3
Server.MapPath("video.WMV") );

//StampAnnotations
theDoc.Pos.String = "40 600";
theDoc.AddText("Stamp annotations");
StampAnnotation stamp1 = new
StampAnnotation(theDoc, "340 560 540 600",
"0 0 128");
StampAnnotation stamp2 = new
StampAnnotation(theDoc, "340 505 540 545",
" 0 128 0");
StampAnnotation stamp3 = new
StampAnnotation(theDoc, "340 450 540 490",
APPROVED", "128 0 0");
```

---

Other

## Other types of annotations

Text annotation 

File Attachment annotation 

Stamp annotations

**DRAFT**

**FINAL**

**NOT APPROVED**



# PDF Rendering Example

---



This example shows how to render a PDF document.

For an example of how to render a PDF direct to screen and how to print a PDF see the ABCpdfView project and classes under the ABCpdf menu item.

We create an ABCpdf Doc object and read our source PDF.

**[C#]**

```
Doc theDoc = new Doc();  
theDoc.Read(Server.MapPath("../Rez/spaceshuttle.pdf"));
```

---

Read

**[Visual Basic]**

```
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../Rez/spaceshuttle.pdf"))
```

We specify our base rendering settings.

**[C#]**

```
theDoc.Rendering.DotsPerInch = 36;
```

---

Prefs

**[Visual Basic]**

```
theDoc.Rendering.DotsPerInch = 36
```

Finally we save the first four pages of the document in png format

### **[C#]**

```
for (int i = 1; i <= 4; i++) {  
    theDoc.PageNumber = i;  
    theDoc.Rect.String = theDoc.CropBox.String;  
    theDoc.Rendering.Save(Server.MapPath("shuttle  
+ i.ToString() + ".png"));  
}
```

---

Save

### **[Visual Basic]**

```
For i As Integer = 1 To 4  
    theDoc.PageNumber = i  
    theDoc.Rect.String = theDoc.CropBox.String  
    theDoc.Rendering.Save(Server.MapPath("shuttle  
+ i.ToString() + ".png"))  
Next
```



shuttle\_p1.gif



Results

shuttle\_p2.png

## ABETTERSPACE SHUTTLETAKES SHAPE

### Better Main Engines

The Space Shuttle's main engines operate at greater extremes of temperature and pressure than any other machine. Since 1981, these overhauls to the original design have more than tripled estimates of their safety. Now, a fourth major overhaul is planned that will make them even safer by 2005. The planned improvements include a high-tech optical and vibration sensor system and computing power in the engine that will "see" trouble coming a fraction of a second before it can do harm. Called the Advanced Health Monitoring System, the sensors will detect and track an almost microscopic flaw in an engine's performance in a split second, allowing the engine to be safely shut down before the situation can grow out of control. Also, the engine's main combustion chamber will be enlarged to reduce the pressures on internal components without reducing the thrust, and a new, simplified engine nozzle design will eliminate thousands of bolts — over 500 feet of them — and potential leaks.



### Safer Hydraulic Power

Aside from the main engines and solid rockets, the single highest-risk equipment on the Space Shuttle are the Auxiliary Power Units, generators that power the Shuttle's hydraulics. Today, those generators use a highly volatile and toxic nuclear fuel. But recent advances in battery and electrical power technology — much of it developed by the automotive industry — will replace that system by 2005, eliminating many hazards not only in flight but also on the ground. Electric motors, powered by a bank of lightweight batteries, will be developed to power the Shuttle's hydraulic system, providing greater reliability for its missions in flight and providing a safer workplace for ground crews.

### "Smart Cockpit"

The new "glue cockpit" that will be retained when Atlantis launches on STS-101 sets the stage for the next cockpit improvement, planned to fly by 2005: a "smart cockpit" that reduces the pilot's workload during critical periods. The enhanced display won't fly the Shuttle, but they will do much of the tedious reasoning required for a pilot to respond to a problem. By simplifying the pilot's job, the "smart cockpit" will allow astronauts to better focus on critical tasks in an emergency.



shuttle\_p3.png

## NOTYOURFATHER'SSPACE SHUTTLE



**April 1983, STS-6  
A Lighter Fuel Tank**  
A redesigned Light Weight External Tank — 80,000 pounds lighter than the original design — flew on STS-6 in 1983, increasing the Shuttle's cargo capacity by the same amount. In 1985, a Super Lightweight External Tank flew on STS-51, further reducing the tank's weight by 15,000 pounds and again increasing the Shuttle's cargo capacity by the same amount. The same super lightweight tank is now scheduled for a final test flight to test out design changes that are being tested in a 10 percent stronger than the previous design.



### Today's Space Shuttle

Since 1992, not only has the cargo capacity of the Shuttle increased by 8 tons, the annual cost of operating the Shuttle has decreased by 40 percent. Improvements to the main engines and other systems have reduced the estimated risks during launch by over 80 percent. And the number of all actual problems experienced by the Shuttle in flight has decreased by 70 percent. Although they have flown for almost 20 years, the Space Shuttle fleet has used only about a quarter of the lifetime for which it was designed. *Discovery*, the most flown Shuttle, has completed 27 trips to space out of 100 flights originally designed for each Shuttle.



**July 1986, STS-51-L  
Space Shuttle Main Engines**  
The Shuttle's main engines have had the highest safety record of any major engine in the world. After 15 flights in 1985, the first in-flight test of the SSME engine, we built design changes to strengthen the engine's independent engine provisions. The next flight, on the 18th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 19th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 20th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 21st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 22nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 23rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 24th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 25th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 26th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 27th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 28th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 29th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 30th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 31st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 32nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 33rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 34th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 35th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 36th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 37th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 38th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 39th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 40th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 41st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 42nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 43rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 44th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 45th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 46th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 47th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 48th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 49th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 50th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 51st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 52nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 53rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 54th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 55th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 56th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 57th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 58th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 59th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 60th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 61st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 62nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 63rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 64th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 65th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 66th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 67th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 68th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 69th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 70th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 71st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 72nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 73rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 74th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 75th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 76th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 77th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 78th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 79th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 80th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 81st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 82nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 83rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 84th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 85th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 86th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 87th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 88th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 89th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 90th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 91st SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 92nd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 93rd SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 94th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 95th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 96th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 97th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 98th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 99th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions. The next flight, on the 100th SSME engine on the STS-51-L mission, we tested the engine's independent engine provisions.

shuttle\_p4.png





## System.Drawing Example

---

This example shows how to port System.Drawing code for our

You may have System.Drawing code which writes output to the console. You may wish to modify this code to write to a PDF. ABCpdf comes and significantly ease this process.

Intro

You can find a full example project including the wrapper class

The wrapper contains the following namespaces:

```
WebSupergoo.ABCpdf10.Drawing;  
WebSupergoo.ABCpdf10.Drawing.Drawing2D;  
WebSupergoo.ABCpdf10.Drawing.Text;
```

These contain classes which are direct analogues for the classes in the System.Drawing namespaces. For example, a System.Drawing.Pen maps to a WebSupergoo.ABCpdf10.Drawing.Drawing2D.Pen and a System.Drawing.Bitmap maps to a WebSupergoo.ABCpdf10.Drawing.Bitmap.

The procedure for porting your System.Drawing code is simple:

Basics

1. Change namespaces from those in System.Drawing to those in WebSupergoo.ABCpdf10.Drawing (Drawing, Drawing.Text, Drawing.Drawing2D etc.)
2. Change all types from those in System.Drawing to the corresponding types in WebSupergoo.ABCpdf10.Drawing (Pen, Brush, Color etc.)
3. Remove any code which does not have an analogue in WebSupergoo.ABCpdf10.Drawing

In general, the number of calls which do not have analogues is small. However, we provide the full source code for System.Drawing to help you port your code.

As well as the standard functions analogous to those in System namespace also contains similar functions aimed at greater control. For example, the ABCpdf Color class contains a FromCmyk method for colors as well as RGB ones.

We start with our source System.Drawing code.

**[C#]**

```
using System;
using System.IO;
using System.Reflection;

using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Text;

...

// create a canvas for painting on
Bitmap pg = new Bitmap((int)(8.5 * 72), (int)
Graphics gr = Graphics.FromImage(pg);

// clear the canvas to white
Rectangle pgRect = new Rectangle(0, 0, pg.Width
SolidBrush solidWhite = new SolidBrush(Color
gr.FillRectangle(solidWhite, pgRect);
// load a new image and draw it centered on
Stream stm =
Assembly.GetExecutingAssembly().GetManifestResource
Image img = Image.FromStream(stm);
int w = img.Width * 2;
int h = img.Height * 2;
Rectangle rc = new Rectangle((pg.Width - w)
gr.DrawImage(img, rc);
```

```
img.Dispose();
stm.Close();
// frame the image with a black border
gr.DrawRectangle(new Pen(Color.Black, 4), rc
// add some text at the top left of the canv
Font fn = new Font("Comic Sans MS", 72);
SolidBrush solidBlack = new SolidBrush(Color
gr.DrawString("My Picture", fn, solidBlack,
(pg.Height * 0.1));

// save the output
pg.Save("../..../abcpdf.drawing.gif", System.D
```

---

Original

## [Visual Basic]

```
Imports System.IO
Imports System.Reflection

Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Text

...

' create a canvas for painting on
Dim pg As Bitmap = New Bitmap(CType((8.5 * 7
Dim gr As Graphics = Graphics.FromImage(pg)

' clear the canvas to white
Dim pgRect As Rectangle = New Rectangle(0,0,
Dim solidWhite As SolidBrush = New SolidBrus
gr.FillRectangle(solidWhite, pgRect)
' load a new image and draw it centered on o
Dim stm As Stream =
Assembly.GetExecutingAssembly().GetManifestR
Dim img As Image = Image.FromStream(stm)
Dim w As Integer = img.Width * 2
```

```

Dim h As Integer = img.Height * 2
Dim rc As Rectangle = New Rectangle((pg.Width
gr.DrawImage(img, rc)
img.Dispose()
stm.Close()
' frame the image with a black border
gr.DrawRectangle(New Pen(Color.Black,4),rc)
' add some text at the top left of the canva
Dim fn As Font = New Font("Comic Sans MS",72
Dim solidBlack As SolidBrush = New SolidBrus
gr.DrawString("My Picture", fn, solidBlack,
(pg.Height * 0.1), Integer))

' save the output
pg.Save("../..../abcpdf.drawing.gif", System.D

```

First, we swap out the old System.Drawing namespaces and Note that ABCpdf.Drawing uses structures like rectangles an System.Drawing. So, we create aliases to make it easy to ref

### [C#]

```

using System;
using System.IO;
using System.Reflection;

using WebSupergoo.ABCpdf10.Drawing;
using WebSupergoo.ABCpdf10.Drawing.Drawing2D
using WebSupergoo.ABCpdf10.Drawing.Text;
using Rectangle = System.Drawing.Rectangle;
using RectangleF = System.Drawing.RectangleF
using Point = System.Drawing.Point;
using PointF = System.Drawing.PointF;

```

## [Visual Basic]

```
Imports System
Imports System.IO
Imports System.Reflection

Imports WebSupergoo.ABCpdf10.Drawing
Imports WebSupergoo.ABCpdf10.Drawing.Drawing
Imports WebSupergoo.ABCpdf10.Drawing.Text
Imports Rectangle = System.Drawing.Rectangle
Imports RectangleF = System.Drawing.RectangleF
Imports Point = System.Drawing.Point
Imports PointF = System.Drawing.PointF
```

Because we're drawing on a page of a PDF document rather than a canvas, we need a few modifications to the first lines of code.

## [C#]

```
// create a canvas for painting on
PDFDocument doc = new PDFDocument();
Page pg = doc.AddPage((int)(8.5 * 72), (int)
Graphics gr = pg.Graphics;
```

Create

## [Visual Basic]

```
' create a canvas for painting on
Dim doc As PDFDocument = New PDFDocument()
Dim pg As Page = doc.AddPage(8.5 * 72, 11 * 72)
Dim gr As Graphics = pg.Graphics
```

The drawing code remains completely unchanged.

## [C#]

```
// clear the canvas to white
Rectangle pgRect = new Rectangle(0, 0, pg.Width, pg.Height);
SolidBrush solidWhite = new SolidBrush(Color.White);
gr.FillRectangle(solidWhite, pgRect);
// load a new image and draw it centered on the canvas
Stream stm = File.OpenRead("image.png");
Image img = Image.FromStream(stm);
int w = img.Width * 2;
int h = img.Height * 2;
Rectangle rc = new Rectangle((pg.Width - w) / 2, (pg.Height - h) / 2, w, h);
gr.DrawImage(img, rc);
img.Dispose();
stm.Close();
// frame the image with a black border
gr.DrawRectangle(new Pen(Color.Black, 4), rc);
// add some text at the top left of the canvas
Font fn = new Font("Comic Sans MS", 72);
SolidBrush solidBlack = new SolidBrush(Color.Black);
gr.DrawString("My Picture", fn, solidBlack, 10, 10);
```

---

Draw

## [Visual Basic]

```
' clear the canvas to white
Dim pgRect As Rectangle = New Rectangle(0,0, pg.Width, pg.Height)
Dim solidWhite As SolidBrush = New SolidBrush(Color.White)
gr.FillRectangle(solidWhite, pgRect)
' load a new image and draw it centered on the canvas
Dim stm As Stream = File.OpenRead("image.png")
Image img = Image.FromStream(stm)
Dim w As Integer = img.Width * 2
Dim h As Integer = img.Height * 2
Dim rc As Rectangle = New Rectangle((pg.Width - w) / 2, (pg.Height - h) / 2, w, h)
gr.DrawImage(img, rc)
```

```
img.Dispose()
stm.Close()
' frame the image with a black border
gr.DrawRectangle(New Pen(Color.Black,4),rc)
' add some text at the top left of the canva
Dim fn As Font = New Font("Comic Sans MS",72)
Dim solidBlack As SolidBrush = New SolidBrush(Color.Black)
gr.DrawString("My Picture", fn, solidBlack,
(pg.Height * 0.1), Integer))
```

Because we're saving to a PDF document, we need to make code.

### [C#]

```
// save the output
doc.Save(Server.MapPath("abcpdf.drawing.pdf"))
```

---

Save

### [Visual Basic]

```
' save the output
doc.Save(Server.MapPath("abcpdf.drawing.pdf"))
```

# My Picture



---

Results

abcpdf.drawing.png

# My Picture





# WPF Tables Example

---

This example shows how to import Windows Presentation Foundation Extensible Application Markup Language (XAML) tables into a PDF. Each table has been specified in a XAML file.

You can find the full project and classes under the ABCpdf menu project includes code for laying out four different types of tables. input data from text files and two from XML files.

The tables are intentionally very similar to those in the [Small Table](#) and [Large Table Example](#) to allow you to compare the two different methods.

Note that the code samples here are in C# only because the example is written in C#.



**WPF Limitations.** The WPF Table component does not support the following features:

WPF supports row and column backgrounds and table and cell backgrounds. However, it does not support row or column borders.

WPF does not support automatic header and footer repetition. If a table is split across pages. Some people appear to have extended the DocumentPaginator class to add headers manually in code during the XPS serialization process. However, our code does not demonstrate this technique.

WPF does not directly support list data binding. By this, we mean there is no way to define a row template and then have the table automatically add a row for each data item in the data provider. We can achieve this functionality by manipulating the XAML code in the XAML to add additional rows.

First, we define the table structure.

Our Small and Large tables have a fairly simple structure. For example, the small table structure is defined in a XAML design file - SmallTable

```
<Table Name="ItemsTable" CellSpacing="10"
BorderBrush="Black" BorderThickness="2"
TextAlignment="Justify">
  <Table.Columns>
    <TableColumn Width="120"/>
    <TableColumn Width="180"/>
    <TableColumn Width="120"/>
    <TableColumn Width="140"/>
    <TableColumn Width="140"/>
  </Table.Columns>
  <TableRowGroup Background="White" DataContext=
  {Binding Source={StaticResource InputData}, Path=
  <TableRow>
    <TableCell>
      <Paragraph>
        <TextBlock TextWrapping="Wrap" Text='
Path=Columns[0]}' />
      </Paragraph>
    </TableCell>
    <TableCell>
      <Paragraph>
        <TextBlock TextWrapping="Wrap" Text='
Path=Columns[1]}' />
      </Paragraph>
    </TableCell>
    <TableCell>
      <Paragraph>
        <TextBlock TextWrapping="Wrap" Text='
Path=Columns[2]}' />
      </Paragraph>
    </TableCell>
  </TableRow>
</TableRowGroup>
</Table>
```

---

**XAML**

```
        </Paragraph>
    </TableCell>
    <TableCell>
        <Paragraph>
            <TextBlock TextWrapping="Wrap" Text='
Path=Columns[3]}' />
        </Paragraph>
    </TableCell>
    <TableCell>
        <Paragraph>
            <TextBlock TextWrapping="Wrap" Text='
Path=Columns[4]}' />
        </Paragraph>
    </TableCell>
</TableRow>
</TableRowGroup>
</Table>;
```

Refer to the [MSDN WPF Table Overview](#) for more information or define WPF tables. Note that we are using Table and not Grid because we need to place our table in a Flow Document. Although Grid can be used as a UI container so as to appear in a flow document, it cannot be used on multiple pages.

Our table definition contains a single row with five columns. We will look on how to duplicate this row so that there is one for every item in our data source so as to achieve list binding in code. Our data source is defined as follows:

```
<c:TextDataProvider x:Key="InputData"
    FileName="text6.txt" />
```

Our TextDataProvider is a custom-written class that extracts text from a separated input file. It supplies this text to the targets via the binding provided by the DataContext and Binding attributes:

```
<TableRowGroup Background="White" DataContext='
```

```
Source={StaticResource InputData}, Path=[0]}">
```

... indicates that this row group is bound to the first line of the file

```
<TextBlock TextWrapping="Wrap" Text="{Binding  
Path=Columns[0]}"/>
```

... indicates that the text block in this cell is bound to the first column.

The large table structure is almost identical to the small table structure defined in LargeTable.xaml. Because the pagination is done automatically by the flow document paginator, the implementation of the two tables is almost identical apart from the small differences in the XAML table definitions. The main difference between the table definitions is that they bind to different input files and use different numbers of columns.

Next, we add the table rows

Because Table does not support binding to a collection of Items (like other WPF UI controls such as ListBox or ItemsControl) - also known as ItemsControl - we manually add table rows in code.

To do so, we load the XAML design file at runtime. We then use XamlReader to duplicate rows and bind them to different data items. To load XAML at runtime, we must set their type as "Content" in the Visual Studio project file. Visual Studio may also help to copy them into the binary location.

### [C#]

```
private MemoryStream ModifyXamlUsingTextProvider  
{  
    TextDataProvider dataProvider = new  
    TextDataProvider(mDataProvider);  
    XmlDocument xamlDoc = new XmlDocument();  
    FileStream xamlFile = new FileStream(mXamlFile
```

```

FileMode.Open);
    xamlDoc.Load(xamlFile);

    XmlNamespaceManager nsmgr = new
XmlNamespaceManager(xamlDoc.NameTable);
    nsmgr.AddNamespace("x",
"http://schemas.microsoft.com/winfx/2006/xaml/p

    XmlNode itemsTable =
xamlDoc.DocumentElement.SelectSingleNode(mTable
"[@Name='" + mTableName + "']", nsmgr);

    for (int i = 1; i < dataProvider.Count; i++)
        XmlNode rowGroup = itemsTable.LastChild;
        XmlNode newRowGroup = rowGroup.Clone();
        string bindingText =
newRowGroup.Attributes["DataContext"].Value;
        bindingText =
bindingText.Remove(bindingText.LastIndexOf('[')
"] }");
        newRowGroup.Attributes["DataContext"].Value
bindingText;
        newRowGroup.Attributes["Background"].Value
? "White" : "LightGray";
        itemsTable.InsertAfter(newRowGroup, rowGrou
    }

    MemoryStream memStream = new MemoryStream();
    xamlDoc.Save(memStream);
    xamlFile.Close();
    return memStream;
}

```

---

Bind

The data provider is loaded directly from code so we know how r (text file lines) are available. The XAML design document is load XmlDocument. We retrieve the XML node that corresponds to ou query passed to SelectSingleNode() in the document element. F

load the WPF namespace else the query will fail.

Once we have our items table, we get hold of the row group. The last child of the table. We duplicate the row group by using the X method and then change the data binding to point to the next row the row background colors. After all items have been added, we document into a memory stream.

### [C#]

```
Page page = XamlReader.Load(memStream) as Page;
FlowDocumentPageViewer docViewer =
LogicalTreeHelper.FindLogicalNode(page, "DocView
FlowDocumentPageViewer;
page.Content = null;
memStream.Close();
return docViewer;
```

We use XamlReader.Load() to load the root object specified in the (stored in the memory stream). This is not the Table, or the Flow then convert it directly to XPS and PDF as shown below); it is a Page because it is convenient to view the Table in the Visual Studio X/ chose to have a Page as the root element. This page contains a viewer and this allows the designer to show the table as we type simply discard the page and keep the document viewer if we want document in a window or we just keep the document if we only want XPS. The page content must be set to null; otherwise, the child element (document viewer and document) cannot be attached to a different window or the XPS document).

The table is saved as XPS first and then converted into PDF.

### [C#]

```
private void SaveToXps(Stream fileStream)
{
```

```

    Package package = Package.Open(fileStream,
    FileMode.Create, FileAccess.ReadWrite);
    XpsDocument doc = new XpsDocument(package);
    XpsDocumentWriter writer =
    XpsDocument.CreateXpsDocumentWriter(doc);
    IDocumentPaginatorSource document =
    CreateDocViewer().Document;
    writer.Write(document.DocumentPaginator);
    doc.Close();
    package.Close();
}

```

Save

```

public void SaveToPdf(string pdfFileName)
{
    MemoryStream memStream = new MemoryStream();
    SaveToXps(memStream);
    WebSupergoo.ABCpdf10.Doc pdfDoc = new
    WebSupergoo.ABCpdf10.Doc();
    pdfDoc.Read(memStream, null, "xps");
    pdfDoc.Save(pdfFileName);
    pdfDoc.Clear();
    memStream.Close();
}

```

We create an XPS file in memory using `MemoryStream`. We then pass the document paginator to the XPS writer. The document paginator source converts the document into a set of pages and writes them to the XPS document. The `document.Read()` method accepts a `Stream` containing the XPS document and a type. After the XPS document has been loaded into the `PDFDoc`, this can be saved to file.

The Invoice and Space table examples bind to an XML data source. The Invoice example binds to a more complex structure.

As such, please see the XAML design files, InvoiceTable.xaml and SpaceTable.xml, for the full table structure.

Because the table input data is specified as XML, we can use the standard data provider, XmlDataProvider, instead of our own custom TextDataProvider.

---

#### XML

Because the data provider is XML, we can use the XmlDocument files as well as XAML design files.

Each table item is mapped to an XML item so we can count the items and add a new row group for each one of them (bar of course the items defined in the design file). Note that the XmlDataProvider indexes items rather than zero based.

We get output very similar to that from the Tables Example project example.

Results

Author	Year	Title	Publisher	Pub No.	Notes
Ansburn, M. P.	1980	Stratigraphy, structure, and petrology of the Great Smoky Group and Murphy sequence near the northeast end of the Murphy Syncline, North Carolina.	Geol. Soc. Am. Abstr. Programs	No. 4	Notes: GEOREF no: 81-00460; Conference: The Geological Society of America, Southeastern Section, 29th annual meeting, Birmingham, Ala., March 27-28, 1980; LAT: N350000; N353000; LONG: W0833000; W084000; GEOREF Category Code: 12 Stratigraphy, Historical geology and paleontology (CC11); Georef subfile: B; Bibliography and Index of Geology (1969-present).
Bason, J. R.; Max, R. P.	1979	Contamination of Great Smoky Mountains trout streams by exposed Andriesta formations:	J. Environ. Qual.	No. 4	Notes: GEOREF no: 80-27361; 18 Refs.; Includes: illus, tables, sketch map; LAT: N352500; N354500; LONG: W0831000; W0834500; GEOREF Category Code: 22 Engineering and environmental geology (CC22); Georef subfile: D; Bibliography and Index of Geology (1969-present).
Bass, C. F. III; McLaughlin, S. B.	1984	Trace elements in tree rings: evidence of recent and historical air pollution:	Science	No. 4648	Notes: GEOREF no: 84-44197; 20 Refs.; Includes: illus; LAT: N353000; N360000; LONG: W0839000; W0841000; GEOREF Category Code: 22 Engineering and environmental geology (CC22); Georef subfile: S; Bibliography and Index of Geology (1969-present).
Bouillon, E. I.; Ferguson, R. B.	1979	Knoxville 1 degree x 2 degree NTMS area, North Carolina, South Carolina, and Tennessee: data release; National Uranium Resource Evaluation Program; hydrogeochemical and stream sediment reconnaissance:	E. I. du Pont de Nemours and Company, for DOE Rept.	No. DPST-78-146-S, GHX-75-79	Notes: GEOINDEX no: 171; GEOINDEX no: 463; GEOREF no: 79-37061; 34 p.; Includes: illus., tables; LAT: N350000; N360000; LONG: W0820000; W0840000; Scale: 1:250,000; GEOREF Category Code: 27 Environmental geology, Metals (CC27); Georef subfile: B; Bibliography and Index of Geology (1969-present); Availability: E. I. du Pont de Nemours and Co., Atton, S.C., United States.

table2.pdf - [Page 1]

Author	Year	Title	Publisher	Pub No.	Notes
Bass, C. F. III; McLaughlin, S. B.	1984	Trace elements in tree rings: evidence of recent and historical air pollution:	Science	No. 4648	Notes: GEOREF no: 84-44197; 20 Refs.; Includes: illus; LAT: N353000; N360000; LONG: W0839000; W0841000; GEOREF Category Code: 22 Engineering and environmental geology (CC22); Georef subfile: S; Bibliography and Index of Geology (1969-present).
Bouillon, E. I.; Ferguson, R. B.	1979	Knoxville 1 degree x 2 degree NTMS area, North Carolina, South Carolina, and Tennessee: data release; National Uranium Resource Evaluation Program; hydrogeochemical and stream sediment reconnaissance:	E. I. du Pont de Nemours and Company, for DOE Rept.	No. DPST-78-146-S, GHX-75-79	Notes: GEOINDEX no: 171; GEOINDEX no: 463; GEOREF no: 79-37061; 34 p.; Includes: illus., tables; LAT: N350000; N360000; LONG: W0820000; W0840000; Scale: 1:250,000; GEOREF Category Code: 27 Environmental geology, Metals (CC27); Georef subfile: B; Bibliography and Index of Geology (1969-present); Availability: E. I. du Pont de Nemours and Co., Atton, S.C., United States.

table2.pdf - [Page 2]

Author	Year	Title	Publisher	Pub No.	Notes
Bass, C. F. III; McLaughlin, S. B.	1984	Trace elements in tree rings: evidence of recent and historical air pollution:	Science	No. 4648	Notes: GEOREF no: 84-44197; 20 Refs.; Includes: illus; LAT: N353000; N360000; LONG: W0839000; W0841000; GEOREF Category Code: 22 Engineering and environmental geology (CC22); Georef subfile: S; Bibliography and Index of Geology (1969-present).
Bouillon, E. I.; Ferguson, R. B.	1979	Knoxville 1 degree x 2 degree NTMS area, North Carolina, South Carolina, and Tennessee: data release; National Uranium Resource Evaluation Program; hydrogeochemical and stream sediment reconnaissance:	E. I. du Pont de Nemours and Company, for DOE Rept.	No. DPST-78-146-S, GHX-75-79	Notes: GEOINDEX no: 171; GEOINDEX no: 463; GEOREF no: 79-37061; 34 p.; Includes: illus., tables; LAT: N350000; N360000; LONG: W0820000; W0840000; Scale: 1:250,000; GEOREF Category Code: 27 Environmental geology, Metals (CC27); Georef subfile: B; Bibliography and Index of Geology (1969-present); Availability: E. I. du Pont de Nemours and Co., Atton, S.C., United States.

Author	Year	Title	Publisher	Pub No.	Notes
Bass, C. F. III; McLaughlin, S. B.	1984	Trace elements in tree rings: evidence of recent and historical air pollution:	Science	No. 4648	Notes: GEOREF no: 84-44197; 20 Refs.; Includes: illus; LAT: N353000; N360000; LONG: W0839000; W0841000; GEOREF Category Code: 22 Engineering and environmental geology (CC22); Georef subfile: S; Bibliography and Index of Geology (1969-present).
Bouillon, E. I.; Ferguson, R. B.	1979	Knoxville 1 degree x 2 degree NTMS area, North Carolina, South Carolina, and Tennessee: data release; National Uranium Resource Evaluation Program; hydrogeochemical and stream sediment reconnaissance:	E. I. du Pont de Nemours and Company, for DOE Rept.	No. DPST-78-146-S, GHX-75-79	Notes: GEOINDEX no: 171; GEOINDEX no: 463; GEOREF no: 79-37061; 34 p.; Includes: illus., tables; LAT: N350000; N360000; LONG: W0820000; W0840000; Scale: 1:250,000; GEOREF Category Code: 27 Environmental geology, Metals (CC27); Georef subfile: B; Bibliography and Index of Geology (1969-present); Availability: E. I. du Pont de Nemours and Co., Atton, S.C., United States.

table2.pdf - [Page 3]

Author	Year	Title	Publisher	Pub No.	Notes
Bass, C. F. III; McLaughlin, S. B.	1984	Trace elements in tree rings: evidence of recent and historical air pollution:	Science	No. 4648	Notes: GEOREF no: 84-44197; 20 Refs.; Includes: illus; LAT: N353000; N360000; LONG: W0839000; W0841000; GEOREF Category Code: 22 Engineering and environmental geology (CC22); Georef subfile: S; Bibliography and Index of Geology (1969-present).
Bouillon, E. I.; Ferguson, R. B.	1979	Knoxville 1 degree x 2 degree NTMS area, North Carolina, South Carolina, and Tennessee: data release; National Uranium Resource Evaluation Program; hydrogeochemical and stream sediment reconnaissance:	E. I. du Pont de Nemours and Company, for DOE Rept.	No. DPST-78-146-S, GHX-75-79	Notes: GEOINDEX no: 171; GEOINDEX no: 463; GEOREF no: 79-37061; 34 p.; Includes: illus., tables; LAT: N350000; N360000; LONG: W0820000; W0840000; Scale: 1:250,000; GEOREF Category Code: 27 Environmental geology, Metals (CC27); Georef subfile: B; Bibliography and Index of Geology (1969-present); Availability: E. I. du Pont de Nemours and Co., Atton, S.C., United States.

table2.pdf - [Page 4]

table2.pdf - [Page 5]



# AddArc Function

---

Adds an arc to the current page.

## [C#]

```
int AddArc(double as, double ae, double  
cx, double cy, double rx, double ry)  
int AddArc(double as, double ae, double  
cx, double cy, double rx, double ry,  
bool filled)
```

## [Visual Basic]

Syntax

```
Function AddArc(as As Double, ae As  
Double, cx As Double, cy As Double, rx  
As Double, ry As Double) As Integer  
Function AddArc(as As Double, ae As  
Double, cx As Double, cy As Double, rx  
As Double, ry As Double, filled As  
Boolean) As Integer
```

Name	Description
as	The start angle of the arc in degrees.
ae	The end angle of the arc in degrees.
cx	The horizontal center of the arc.

---

**Params**

cy	The vertical center of the arc.
rx	The horizontal radius of the arc.
ry	The vertical radius of the arc.
filled	Whether to fill the arc rather than simply drawing it.
return	The Object ID of the newly added Graphic Object.

Adds an arc to the current page. The arc is drawn in the current **color** at the current **width** and with the current **options**.

---

**Notes**

The arc is fixed at the center coordinate and can have different horizontal and vertical radii. Drawing starts at the start angle and the arc is swept out until the end angle is reached. Angles are measured anti-clockwise with zero at three o'clock.

The AddArc function returns the Object ID of the newly added Graphic Object.

The following code adds an arc to a document.

**[C#]**

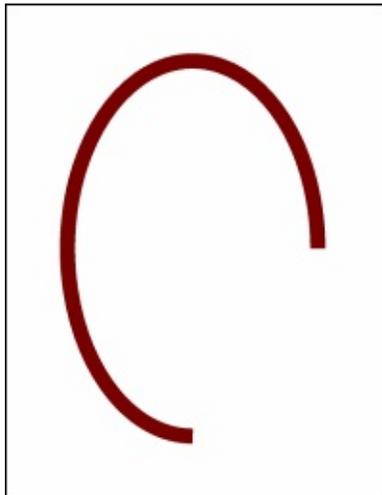
```
Doc theDoc = new Doc();  
theDoc.Width = 24;  
theDoc.Color.String = "120 0 0";
```

```
theDoc.AddArc(0, 270, 300, 400, 200, 300);  
theDoc.Save(Server.MapPath("docaddarc.pdf"))  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Width = 24  
theDoc.Color.String = "120 0 0"  
theDoc.AddArc(0, 270, 300, 400, 200, 300)  
theDoc.Save(Server.MapPath("docaddarc.pdf"))  
theDoc.Clear()
```

Example



docaddarc.pdf



# AddBookmark Function

---

Adds a bookmark pointing to the current page.

## [C#]

```
int AddBookmark(string path, bool open)
```

---

## Syntax

## [Visual Basic]

```
Function AddBookmark(path As String, open As Boolean) As Integer
```

---

## Params

Name	Description
path	The path to the bookmark.
open	Whether the bookmark should be displayed open (expanded) or closed (collapsed).
return	The Object ID of the newly added Bookmark Object

Adds a bookmark which points to the current page.

Bookmarks are specified as paths in much the same way as file paths. Headings and subheadings are separated by a backslash character. For example:

```
"1. Introduction\1.1 Aim and Methods\1.1.3 Diagrams"
```

When a bookmark is added, intermediate headings and subheadings will be created if they do not already exist.

To add multiple bookmarks all referring to the same page simply call the AddBookmark function multiple times.

This function returns the Object ID of the newly added Bookmark Object.

---

#### Notes

**Hyperlinks.** Sometimes you may wish to insert bookmarks which link to an external web page specified using a URI or URL. You can do this using code of the following form.

```
static int AddBookmarkToUri(Doc doc,
string bookmark, string uri) {
    int id = doc.AddBookmark(bookmark,
true);
    doc.SetInfo(id, "/Dest:Del", ""); //
remove link to page and add link to url
    doc.SetInfo(id, "/A", "<< /Type
/Action /S /URI /URI () >>");
    doc.SetInfo(id, "/A/URI:Text", uri);
    return id;
}
```

The following code adds a sequence of pages with a nested sequence of bookmarks. The image shows the appearance of the document outline. Note that none of the subject pages are visible because the chapter pages were added in a collapsed state.

## [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 64;
string theSection, theChapter, theSubject;
for (int i = 1; i < 3; i++) {
    theDoc.Page = theDoc.AddPage();
    theSection = i.ToString() + " Section";
    theDoc.AddText(theSection);
    theDoc.AddBookmark(theSection, true);
    for (int j = 1; j < 5; j++) {
        theDoc.Page = theDoc.AddPage();
        theChapter = theSection + "\\ " + j.ToSt
+ " Chapter";
        theDoc.AddText(theChapter);
        theDoc.AddBookmark(theChapter, false);
        for (int k = 1; k < 6; k++) {
            theDoc.Page = theDoc.AddPage();
            theSubject = theChapter + "\\ " +
k.ToString() + " Subject";
            theDoc.AddText(theSubject);
            theDoc.AddBookmark(theSubject, true);
        }
    }
}
theDoc.Save(Server.MapPath("docaddbookmark.
theDoc.Clear());
```

## [Visual Basic]

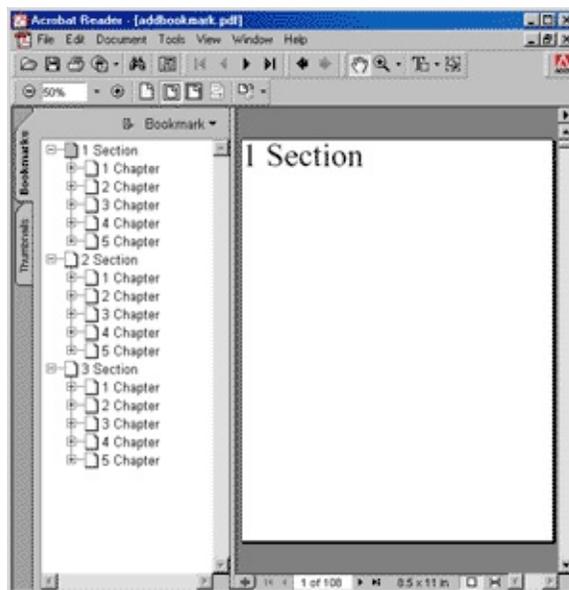
```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 64
Dim theSection, theChapter, theSubject As S
For i As Integer = 1 To 2
    theDoc.Page = theDoc.AddPage()
    theSection = i.ToString() + " Section"
    theDoc.AddText(theSection)
    theDoc.AddBookmark(theSection, True)
```

Example

```

For j As Integer = 1 To 4
    theDoc.Page = theDoc.AddPage()
    theChapter = theSection + "\" + j.ToStr
" Chapter"
    theDoc.AddText(theChapter)
    theDoc.AddBookmark(theChapter, False)
    For k As Integer = 1 To 5
        theDoc.Page = theDoc.AddPage()
        theSubject = theChapter + "\" + k.ToS
+ " Subject"
        theDoc.AddText(theSubject)
        theDoc.AddBookmark(theSubject, True)
    Next
Next
Next
theDoc.Save(Server.MapPath("docaddbookmark.
theDoc.Clear()

```



docaddbookmark.pdf



## AddColorSpaceFile Function

---

Adds an ICC based color space to the document.

### [C#]

```
int AddColorSpaceFile(string path)
```

### [Visual Basic]

```
Function AddColorSpaceFile(path As String)  
Integer
```

Syntax

- may throw Exception()

Params

Name	Description
path	The path to the ICC color space file.
return	The Object ID of the newly added ColorSpace Obj

Adds an ICC based color space to the document returning the ID of the newly added object.

ICC Color Spaces allow the precise representation of colors in an independent way. The ICC format is defined by the International Color Consortium (<http://www.color.org/>).

ABCpdf will allow you to add ICC profiles in the Gray, RGB, Lab color spaces. This method will throw an error if the file is inaccessible or invalid.

---

#### Notes

The current color space is defined by the [ColorSpace](#) property. The current color is defined by the [Color](#) property. The [ColorSpace](#) property is used when the [Color](#) is of a matching type. If the color type does not match then the default - device - color space is used.

For example you add a CMYK color space and assign it to the [ColorSpace](#) property. All CMYK Colors you use will be defined in terms of this color space. However RGB and Grayscale colors will be defined in terms of the default - device - color spaces.

In this example we add some CMYK text defined in an ICC color space.

#### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in tres, quarum unam incolunt Belgae, aliam Aquitania, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
theDoc.FontSize = 96;
string thePath = Server.MapPath("../mypics/");
theDoc.ColorSpace = theDoc.AddColorSpaceFile(thePath, "CMYK.icc");
theDoc.Color.String = "200 20 20 20";
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("docaddcolorspace.pdf"));
theDoc.Clear();
```

#### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
```

---

Example

```
Dim theText As String = "Gallia est omnis c  
partes tres, quarum unam incolunt Belgae, a  
Aquitani, tertiam qui ipsorum lingua Celtae  
Galli appellantur."  
theDoc.Rect.Inset(20, 40)  
theDoc.FontSize = 96  
Dim thePath As String =  
Server.MapPath("../mypics/cmyk.icc")  
theDoc.ColorSpace = theDoc.AddColorSpaceFil  
theDoc.Color.String = "200 20 20 20"  
theDoc.AddText(theText)  
theDoc.Save(Server.MapPath("docaddcolorspac  
theDoc.Clear()
```



Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,

docaddcolorspacefile.pdf



## AddColorSpaceSpot Function

---

Adds a separation color space to the document.

### [C#]

```
int AddColorSpaceSpot(string name, string c
```

---

### Syntax

### [Visual Basic]

```
Function AddColorSpaceSpot(name As String,  
String) As Integer
```

---

### Params

Name	Description
name	The name of the color.
color	The display representation of the color. This should be supplied in the same format as the <a href="#">XColor.String</a> us
return	The Object ID of the newly added ColorSpace Obje

Adds a separation color space to the document returning the newly added object.

Separation color spaces allow you to add spot colors or iso of individual colorants. A separation color space represents

colorant and allows you to specify the amount of that color applied.

For example you might define a separation color space call a display representation of CMYK yellow. You might then draw text with different amounts of Gold. When viewing the output on a standard monitor the display representation of the color would be yellow. However when printed - using appropriate software and an appropriate printer - the name of the color might be used to select a gold ink.

---

Notes

The current color space is defined by the `ColorSpace` property. The current color is defined by the `Color` property. Because separations represent a single value property you should use single color values for grayscale - colors to specify the amount of spot colorant to use.

There are two special color spaces:

'All' refers collectively to all colorants on an output device. It is useful for printing registration marks.

'None' indicates no colorant and will never produce any visible output.

In this example we define a colorant called Gold and add some text with varying amounts of our colorant.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(20, 20);
theDoc.FontSize = 300;
theDoc.ColorSpace = theDoc.AddColorSpaceSpot("Gold", "0 100 0");
for (int i = 1; i <= 10; i++) {
    theDoc.Color.Gray = 255 / i;
    theDoc.AddText(theDoc.Color.Gray.ToString(), "Gold " + i);
    theDoc.Rect.Move(25, -50);
}
```

```
}  
theDoc.Save(Server.MapPath("docaddcolorspac  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Rect.Inset(20, 20)  
theDoc.FontSize = 300  
theDoc.ColorSpace = theDoc.AddColorSpaceSp  
0 100 0")  
For i As Integer = 1 To 10  
    theDoc.Color.Gray = 255 / i  
    theDoc.AddText(theDoc.Color.Gray.ToString  
theDoc.Rect.Move(25, -50)  
Next  
theDoc.Save(Server.MapPath("docaddcolorspac  
theDoc.Clear()
```

Example



docaddcolorspacespot.pdf



## AddFont Function

---

Adds a font reference to the document.

### [C#]

```
int AddFont(string name)
int AddFont(string name, LanguageType
language)
int AddFont(string name, LanguageType
language, bool vertical)
```

### [Visual Basic]

Syntax

```
Function AddFont(name As String) As
Integer
Function AddFont(name As String, language
As LanguageType) As Integer
Function AddFont(name As String, language
As LanguageType, vertical As Boolean) As
Integer
```

Name	Description
name	The name of the font typeface.
	The language type to use. The LanguageType enumeration may take the following values: <ul style="list-style-type: none"><li>• Latin</li><li>• Unicode - NB: This requires that the font</li></ul>

---

**Params**

language	<p>be embedded. If you wish to use this selector use the EmbedFont call.</p> <ul style="list-style-type: none"><li>• Korean</li><li>• Japanese</li><li>• ChineseS</li><li>• ChineseT</li></ul> <p>See the <a href="#">Fonts and Languages</a> section for details on language types. The default language type is Latin.</p>
vertical	<p>Whether the text direction should be vertical.</p> <p>See the <a href="#">Fonts and Languages</a> section for details on writing directions. The default is false to indicate standard left to right layout.</p>
return	<p>The Object ID of the newly added Font Object.</p>

Adds a font to the document.

The font name and a description of the font style is held within the document. However the actual font itself is not added to the document unless the language is Unicode. (When Unicode is specified, this method will embed the font without subsetting.) If you wish to embed a font or use the Unicode language, you should use the [EmbedFont](#) method.

When a client opens the PDF Acrobat will attempt to find the exact same font on the client system. If the exact font is not available then a substitute font will be chosen using the font description to determine the best match.

The following fonts are guaranteed to be available on every system.

- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Symbol
- ZapfDingbats

Additionally you can add any TrueType, OpenType or Type 1 font that you have installed on your system. The name you should use is the one referenced in your fonts folder. For example.

- Arial
- Arial Black
- Arial Black Italic
- ...
- Book Antiqua
- Book Antiqua Bold
- Book Antiqua Bold Italic
- Book Antiqua Italic
- ...
- Venetian301BT
- Venetian301BT Bold
- ...

---

**Notes**

The AddFont function returns the Object ID of the newly added Font Object. Typically you will want to assign this

return value to the document Font property using code of the form.

```
theDoc.Font = theDoc.AddFont("Courier")
```

If the specified font could not be found then you will get an Object ID of zero returned. You may wish to check for this possibility otherwise a default font will be used.

Fonts are cached so newly added fonts will not be available to ABCpdf until the application is restarted. If you need to dynamically load a font you can pass this method a path to your font file. This will add the font to the cache and make it available for use. You should not move, rename or delete a font file which has been dynamically loaded using this technique.

### **Why is my language a string?**

In older versions of ABCpdf the language parameter was a string. So you might find code of this form.

```
theDoc.AddFont("Courier", "Latin")
```

In more recent releases the language parameter has been changed to a true enumeration. This is a safer way of coding as it allows the compiler to ensure that the values you are using are valid. Your new code should look like this.

```
theDoc.AddFont("Courier",  
LanguageType.Latin)
```

The names of the items in the LanguageType enumeration are the same as the values of the

strings used in previous versions. So changing your code should be a simple search and replace operation. If you see a language of "" this equates to LanguageType.Latin.

The following code adds two pieces of text to a document. The first piece is in Times-Roman and the second in Helvetica-E

### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 48;
string theFont = "Times-Roman ";
theDoc.Font = theDoc.AddFont(theFont);
theDoc.AddText(theFont);
theFont = "Helvetica-Bold";
theDoc.Font = theDoc.AddFont(theFont);
theDoc.AddText(theFont);
theDoc.Save(Server.MapPath("docaddfont.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 48
Dim theFont As String = "Times-Roman "
theDoc.Font = theDoc.AddFont(theFont)
theDoc.AddText(theFont)
theFont = "Helvetica-Bold"
theDoc.Font = theDoc.AddFont(theFont)
theDoc.AddText(theFont)
theDoc.Save(Server.MapPath("docaddfont.pdf"))
```

**Example**

```
theDoc.Clear()
```

Times-Roman **Helvetica-**  
**Bold**

docaddfont.pdf



## AddGrid Function

---

Adds a visible grid to the current page.

### [C#]

```
int AddGrid()
```

---

### Syntax

### [Visual Basic]

```
Function AddGrid() As Integer
```

Name	Description
return	The Object ID of the newly added Grid Object.

---

### Params

Adds a visible grid to the current page. The grid shows locations on the page and the effect of the current transform. It is designed to help with object positioning during development.

---

### Notes

The following code modifies the page transform and then a grid to show how the transform has affected the page.

### [C#]

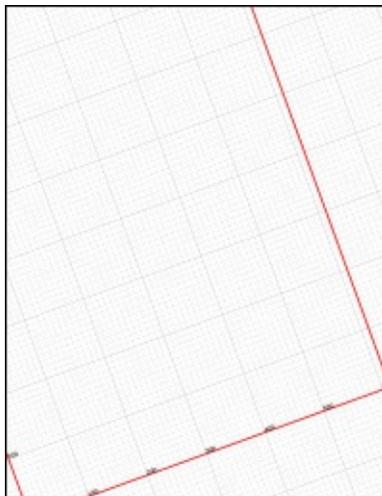
```
Doc theDoc = new Doc();
```

```
theDoc.Page = theDoc.AddPage();  
theDoc.Transform.Rotate(20, 100, 100);  
theDoc.AddGrid();  
theDoc.Save(Server.MapPath("docaddgrid.pdf")  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Page = theDoc.AddPage()  
theDoc.Transform.Rotate(20, 100, 100)  
theDoc.AddGrid()  
theDoc.Save(Server.MapPath("docaddgrid.pdf")  
theDoc.Clear()
```

Example



docaddgrid.pdf



## AddHtml Function

---

Adds a block of HTML styled text to the current page.

### [C#]

```
int AddHtml(string text)
int AddHtml(string dummy, int chainid)
```

### [Visual Basic]

#### Syntax

```
Function AddHtml(text As String) As Integer
Function AddHtml(dummy As String, chainid As Integer) As Integer
```

#### Params

Name	Description
text	The HTML to be added to the page.
dummy	A dummy parameter. The contents are not used.
chainid	The Object ID of a previously added text block.
return	The Object ID of the newly added Text Object.

Adds a block of HTML styled text to the current page.

This function works in a similar manner to the [AddText](#) function but it allows you to add multi-styled text by

inserting simple HTML tags. A listing of supported tags is given in the [HTML Styled Text](#) section of the documentation. Please see [Pos](#) for details on positioning when using vertical fonts.

You can chain together multiple text/HTML blocks so that text flows from one to the next. To do this you need to first add a block of text/HTML using [AddText](#) or `AddHtml`. Then add multiple new text/HTML blocks using `Doc.AddHtml` each time passing in the ID obtained from the previous call after adjusting the target location (such as the rectangle or the page).

When no more text is available the `AddHtml` function will return zero. Alternatively you can query the [TextLayer.Truncated](#) property of the returned object before adding a new item to the chain.

---

**Notes**

This function returns the Object ID of the newly added Text Object. If no text could be added then zero is returned. This will happen if a zero length string was supplied or if the rectangle was too small for even one character to be displayed.

Typically you will get a return value of zero if your text was too large to fit in your [Rect](#) or if the [Pos](#) was at the end of the Rect.

Text styles for the entire HTML content are determined at the point at which the first item in a text chain is created. This means that varying document styles will not affect the way in which subsequent items in the chain are displayed.

For an example of chaining see the [Text Flow Example](#).

Note that there is no requirement that blocks be organized in a linear chain. If you wish you can create trees of HTML blocks, flowing text from a chain head through multiple display areas on your document.

The following code adds some HTML styled text to a docun

### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 72;
theDoc.AddHtml("<b>Gallia</b> est omnis div
in partes tres, quarum unam incolunt
<b>Belgae</b>, aliam <b>Aquitani</b>, terti
qui ipsorum lingua <b>Celtae</b>, nostra
<b>Galli</b> appellantur.");
theDoc.Save(Server.MapPath("docaddhtml.pdf")
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 72
theDoc.AddHtml("<b>Gallia</b> est omnis div
in partes tres, quarum unam incolunt
<b>Belgae</b>, aliam <b>Aquitani</b>, terti
qui ipsorum lingua <b>Celtae</b>, nostra
<b>Galli</b> appellantur.")
theDoc.Save(Server.MapPath("docaddhtml.pdf")
theDoc.Clear()
```

---

Example

**Gallia** est omnis  
divisa in partes tres,  
quarum unam  
incolunt **Belgae**,  
aliam **Aquitani**,  
tertiam qui ipsorum  
lingua **Celtae**, nostra  
**Galli** appellantur.

docaddhtml.pdf



## AddImage Function

---

Adds an image to the current page.

### [C#]

```
int AddImage(XImage image)
int AddImage(string path)
int AddImage(byte[] data)
int AddImage(string path, int
page)
int AddImage(byte[] data, int
page)
int AddImage(int id)
int
AddImage(System.Drawing.Bitmap
bm)
int AddImage(Doc doc, int page)
int AddImage(Doc doc, int page,
XRect src)
```

### [Visual Basic]

```
Function AddImage(image As
XImage) As Integer
Function AddImage(path As String)
As Integer
Function AddImage(data() As Byte)
As Integer
Function AddImage(path As String,
page As Integer) As Integer
Function AddImage(data() As Byte,
page As Integer) As Integer
Function AddImage(id As Integer)
```

Syntax

```
As Integer
Function AddImage(bm As
System.Drawing.Bitmap) As Integer
Function AddImage(doc As Doc,
page As Integer) As Integer
Function AddImage(doc As Doc,
page As Integer, src As XRect) As
Integer
```

- may throw Exception()

Name	Description
image	An <a href="#">XImage</a> containing the image to be added to the page.
path	A file path, URL or html string to be added to the page.
data	The raw JPEG data to be added to the page.
id	An existing image object ID to be copied to the page again.
bm	A .NET Bitmap to be added to the page.
doc	A PDF document page to be added to the page.

---

**Params**

page	The page of the document to be added.
src	The source area of the document page to be added.
return	The ID of the newly added Image Object.

Adds an image to the current page returning the ID of the newly added object.

This method attempts to make a sensible decision as to which of the following methods to call dependent on the content of the ImageSpec.

- [AddImageFile](#)
- [AddImageData](#)
- [AddImageCopy](#)
- [AddImageUrl](#)
- [AddImageHtml](#)
- [AddImageToChain](#)
- [AddImageDoc](#)
- [AddImageObject](#)
- [AddImageBitmap](#)

---

Notes

This function is provided for backwards compatibility purposes. You should call the appropriate method directly rather than working via this method.

---

**Example**

None.



## AddImageBitmap Function

---

Adds a System.Drawing.Bitmap to the current page.

### [C#]

```
int AddImageBitmap(System.Drawing.Bitmap bm  
bool transparent)
```

### [Visual Basic]

```
Function AddImageBitmap(bm As  
System.Drawing.Bitmap, transparent As Boolean  
As Integer
```

Syntax

- may throw Exception()

Params

Name	Description
bm	A .NET Bitmap to be added to the page.
transparent	Whether transparency information should be preserved.
return	The ID of the newly added Image Object.

Adds a `System.Drawing.Bitmap` to the current page.

If the `Transparent` parameter is set to `true` then any transparency information will be preserved. This allows you to add format as transparent GIF and PNG with alpha channel.

---

**Notes**

The image is scaled to fill the current `Rect`. It is transformed by the current `Transform`.

The following code adds a transparent PNG to a document.

**[C#]**

```
Doc theDoc = new Doc();
string thePath =
Server.MapPath("../mypics/mypic.png");
Bitmap bm = new Bitmap(thePath);
theDoc.Rect.Inset(20, 20);
theDoc.Color.String = "0 0 200";
theDoc.FillRect();
theDoc.AddImageBitmap(bm, true);
bm.Dispose();
theDoc.Save(Server.MapPath("docaddimagebitr
theDoc.Clear());
```

**[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
Dim thePath As String =
Server.MapPath("../mypics/mypic.png")
Dim bm As Bitmap = New Bitmap(thePath)
theDoc.Rect.Inset(20, 20)
theDoc.Color.String = "0 0 200"
theDoc.FillRect()
theDoc.AddImageBitmap(bm, True)
bm.Dispose()
```

---

**Example**

```
theDoc.Save(Server.MapPath("docaddimagebitr  
theDoc.Clear()
```



docaddimagebitmap.pdf



## AddImageCopy Function

---

Adds a copy of an existing image in the Doc, to the current

### [C#]

```
int AddImageCopy(int id)
```

### [Visual Basic]

Syntax

```
Function AddImageCopy(id As Integer) As Int
```

- may throw Exception()

Params

Name	Description
id	An existing image object ID to be copied to the page again.
return	The ID of the newly added Image Object.

Adds a copy of an image which has already been inserted elsewhere in the document.

You can use this facility to add commonly used graphics such as watermarks. The raw image data is inserted only once which means that PDF size is greatly reduced.

---

**Notes**

This method only works with raster or bitmap images. So you must have been obtained from a previous call to [AddImage](#), [AddImageData](#) or [AddImageObject](#).

The image is scaled to fill the current [Rect](#). It is transformed by the current [Transform](#).

This example shows how to read an existing PDF document and insert a background image into every page.

We start by reading our template PDF document and finding out the information we will need to reference each page.

**[C#]**

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.pdf"));
int theCount = theDoc.PageCount;
```

**[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample.pdf"))
Dim theCount As Integer = theDoc.PageCount
```

We cycle through the pages inserting images as we go.

We set the layer property to ensure that the image gets added to the background rather than on top of existing content.

The first time we add an image file. Subsequent times we reference the image ID. This means that we embed only one copy of the image and simply reference that data from each page.

Finally we save the modified PDF.

**[C#]**

---

```

int theID = 0;
for (int i = 1; i <= theCount; i++) {
    theDoc.PageNumber = i;
    theDoc.Layer = theDoc.LayerCount + 1;
    if (i == 1) {
        string thePath =
Server.MapPath("../mypics/light.jpg");
        theID = theDoc.AddImageFile(thePath, 1)
    }
    else
        theDoc.AddImageCopy(theID);
}
theDoc.Save(Server.MapPath("watermark.pdf"))
theDoc.Clear();

```

### [Visual Basic]

```

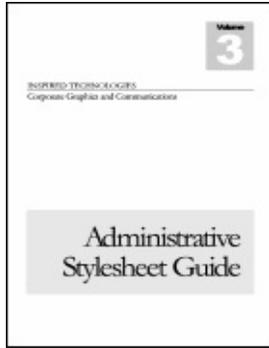
Dim theID As Integer = 0
For i As Integer = 1 To theCount
    theDoc.PageNumber = i
    theDoc.Layer = theDoc.LayerCount + 1
    If i = 1 Then
        Dim thePath As String =
Server.MapPath("../mypics/light.jpg")
        theID = theDoc.AddImageFile(thePath, 1)
    Else
        theDoc.AddImageCopy(theID)
    End If
Next
theDoc.Save(Server.MapPath("watermark.pdf"))
theDoc.Clear()

```

Given the following document.

---

**Example**



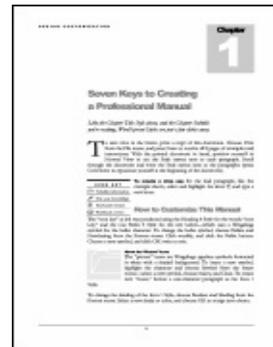
sample.pdf - [Page 1]



sample.pdf - [Page 2]



sample.pdf - [Page 3]

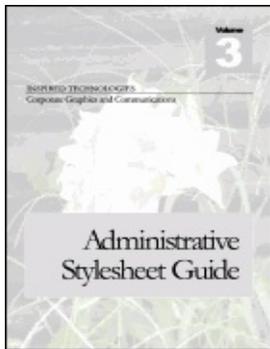


sample.pdf - [Page 4]

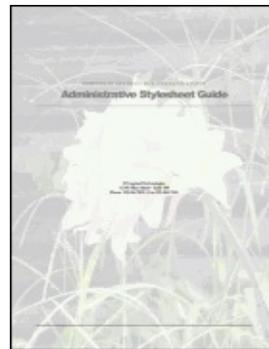
And the following image.



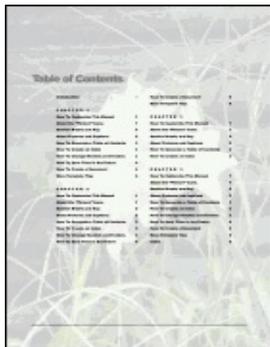
This is the kind of output you might expect.



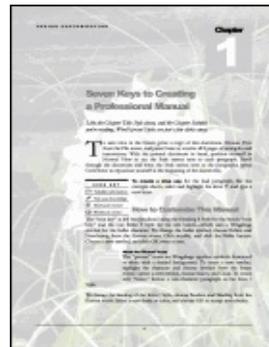
watermark.pdf - [Page 1]



watermark.pdf - [Page



watermark.pdf - [Page 3]



watermark.pdf - [Page

# AddImageData Function

---



Extract an image from data and add it to the current page.

## [C#]

```
int AddImageData(byte[] data)
int AddImageData(byte[] data, int
frame)
```

## [Visual Basic]

```
Function AddImageData(data() As
Byte) As Integer
Function AddImageData(data() As
Byte, frame As Integer) As
Integer
```

Syntax

- may throw Exception()

Params

Name	Description
data	The data containing the image in a format such as JPEG or TIFF.
frame	Some image types support multiple frames or pages. This is the one based index specifying the required frame (default one).

return	The ID of the newly added Image Object.

Extract an image from a chunk of data and add it to the current page returning the ID of the newly added object.

---

**Notes**

This method is essentially the same as [AddImageFile](#) but it allows you add images specified as raw data rather than as a file path

---

**Example**

None.

# AddImageDoc Function

---



Draw a page from one PDF document onto the current page document.

## [C#]

```
int AddImageDoc(Doc doc, int page, XRect rect)
int AddImageDoc(Doc doc, int page, XRect rect,
bool copyAnnotations)
int AddImageDoc(Doc doc, int page, XRect rect,
double alpha)
```

## [Visual Basic]

Syntax

```
Function AddImageDoc(doc As Doc, page As Integer,
rect As XRect) As Integer
Function AddImageDoc(doc As Doc, page As Integer,
rect As XRect, copyAnnotations As Boolean)
As Integer
Function AddImageDoc(doc As Doc, page As Integer,
rect As XRect, copyAnnotations As Boolean,
alpha As Double) As Integer
```

Name	Description
doc	The document to be used as the source.
page	The page you want drawn. Use one to in first page.

---

**Params**

rect	The portion of the page you want drawn. to specify the entire page.  The format of this string should be the same that obtained via the <a href="#">XRect.String</a> property.
copyAnnotations	Whether to copy fields and annotations - false.
alpha	The level of alpha to apply to the drawn page transparent through to fully opaque (0 to 1).
return	The ID of the newly added Image Object

Draw a page from one PDF document onto the current page document returning the ID of the newly added object.

The page is scaled to fill the current [Rect](#). It is transformed current [Transform](#).

Many field and annotation types can only exist as a simple rectangle with sides parallel to the page borders. For this reason you should be cautious about the transforms you use when specifying that annotations should be copied. A transform which involves only translation will be fine but one involving rotation and skew will result in unusual output if the field or annotation does not support that combination.

---

**Notes**

The [Refactor](#) setting determines whether new/modified redacted objects are eliminated. The [Preflight](#) setting determines whether objects in the destination document are validated before this operation is performed. Unless the document and the pages are big in memory use and have many common objects, it is faster to refactor and preflight for adding the pages and enable them on the document. You can use [SetInfo](#) to change these settings.

Pages may be rotated. As such, when drawing one page on another, you may wish to copy the [Page.Rotation](#) from the source page to the destination page. More complex example code to rotate a page may be found under the documentation for the [Page.Rotation](#).

This example shows how to draw one PDF into another. It reads a source document and creates a 'four-up' summary document by drawing a quarter of each page on each page of the new document.

First we create an ABCpdf Doc object and read in our source document.

#### [C#]

```
Doc theSrc = new Doc();
theSrc.Read(Server.MapPath("../Rez/spaceship.pdf"));
int theCount = theSrc.PageCount;
```

#### [Visual Basic]

```
Dim theSrc As Doc = New Doc()
theSrc.Read(Server.MapPath("../Rez/spaceship.pdf"))
Dim theCount As Integer = theSrc.PageCount
```

Next we create a destination Doc object and set the [MediaBox](#) property. The destination page size will match that of the source document. We change the destination page size so that it occupies a quarter of the page with room to accommodate a margin.

#### [C#]

```
Doc theDst = new Doc();
theDst.MediaBox.String = theSrc.MediaBox.String;
theDst.Rect.String = theDst.MediaBox.String;
theDst.Rect.Magnify(0.5, 0.5);
theDst.Rect.Inset(10, 10);
double theX, theY;
```

```
theX = theDst.MediaBox.Width / 2;
theY = theDst.MediaBox.Height / 2;
```

### [Visual Basic]

```
Dim theDst As Doc = New Doc()
theDst.MediaBox.String = theSrc.MediaBox.St
theDst.Rect.String = theDst.MediaBox.String
theDst.Rect.Magnify(0.5, 0.5)
theDst.Rect.Inset(10, 10)
Dim theX As Double,theY As Double
theX = theDst.MediaBox.Width / 2
theY = theDst.MediaBox.Height / 2
```

We go through every page in the source document drawing each page at a different position on our four-up document. I we add a new page into our destination document.

### [C#]

```
for (int i = 1; i <= theCount; i++) {
    switch (i % 4) {
        case 1:
            theDst.Page = theDst.AddPage();
            theDst.Rect.Position(10, theY + 10);
            break;
        case 2:
            theDst.Rect.Position(theX + 10, theY
            break;
        case 3:
            theDst.Rect.Position(10, 10);
            break;
        case 0:
            theDst.Rect.Position(theX + 10, 10);
            break;
    }
    theDst.AddImageDoc(theSrc, i, null);
    theDst.FrameRect();
```

---

Example

```
}
```

### [Visual Basic]

```
For i As Integer = 1 To theCount
  Select Case i Mod 4
    Case 1
      theDst.Page = theDst.AddPage()
      theDst.Rect.Position(10, theY + 10)
    Case 2
      theDst.Rect.Position(theX + 10, theY
    Case 3
      theDst.Rect.Position(10, 10)
    Case 0
      theDst.Rect.Position(theX + 10, 10)
  End Select
  theDst.AddImageDoc(theSrc, i, Nothing)
  theDst.FrameRect()
Next
```

Finally we save.

### [C#]

```
theDst.Save(Server.MapPath("fourup.pdf"));
// finished
```

### [Visual Basic]

```
theDst.Save(Server.MapPath("fourup.pdf"))
' finished
```

We get the following output.



fourup.pdf - [Page 1]



fourup.pdf - [Page 2]



## AddImageFile Function

---

Extract an image from a file and add it to the current page.

### [C#]

```
int AddImageFile(string path)
int AddImageFile(string path, int frame)
```

### [Visual Basic]

```
Function AddImageFile(path As String) As Integer
Function AddImageFile(path As String,
frame As Integer) As Integer
```

Syntax

- may throw Exception()

Params

Name	Description
path	A file path, URL or html string to be added to the page.
frame	Some image types support multiple frames or pages. This is the one based index specifying the required frame (default one).
return	The Object ID of the newly added Image Object.

Adds an image to the current page returning the ID of the newly added object.

Images embedded using this method are always inserted using [pass-through](#) mode. Pass-through mode is faster than [indirect](#) mode. It allows the preservation of compression settings, native color spaces and ICC color profiles. It allows vector graphics to be maintained in vector format. However, it supports a limited range of image formats - JPEG, JPEG 2000, TIFF, EMF, WMF, PS (PostScript) or EPS (Encapsulated PostScript). JPEG images may be grayscale, RGB or CMYK. TIFF images may be black and white, grayscale, RGB, CMYK, TIFF or Lab in a range of bit depths and with or without alpha. Because PDF does not support 32 bit High Dynamic Range (HDR) encodings, TIFFs in this format will be downsampled to 16 bits per component.

Note that not all EMF or WMF files can be directly imported this way. If this is the case you should look at using the [XImage](#) object. Using the XImage object with the default ReadModule is fast but will result in the image being rasterized. Using the XpsAny ReadModule will not be as fast but will preserve the vector nature of practically all such files.

The image is scaled to fill the current [Rect](#). It is transformed using the current [Transform](#).

---

#### Notes

**Transparency.** Occasionally you may find that you need to invert the transparency of your image. To do this you can assign a decode array using the ID returned from this function.

To invert the transparency:

```
theDoc.SetInfo(theDoc.GetInfoInt(theID,
```

```
"XObject"), "/SMask*/Decode", "[1 0]")
```

A similar technique can be used for inverting or altering color levels on the image itself.

To invert an RGB image:

```
theDoc.SetInfo(theDoc.GetInfoInt(theID,  
"XObject"), "/Decode", "[1 0 1 0 1 0]")
```

The following code adds an image to the current page position at the bottom left. The width and height of the image are automatically inferred from the file supplied.

### [C#]

```
Doc theDoc = new Doc();  
theDoc.Rect.String = "0 0 0 0";  
string thePath =  
Server.MapPath("../mypics/pic.jpg");  
theDoc.AddImageFile(thePath, 1);  
theDoc.Save(Server.MapPath("docaddimage.pdf"));  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Rect.String = "0 0 0 0"  
Dim thePath As String =  
Server.MapPath("../mypics/pic.jpg")  
theDoc.AddImageFile(thePath, 1)  
theDoc.Save(Server.MapPath("docaddimage.pdf"))
```

Example

```
theDoc.Clear()
```



docaddimage.pdf



# AddImageHtml Function

---

Renders a web page specified as HTML.

## [C#]

```
int AddImageHtml(string html)
int AddImageHtml(string html,
bool paged, int width, bool
disableCache)
```

## [Visual Basic]

```
Function AddImageHtml(html As
String) As Integer
Function AddImageHtml(html As
String, paged As Boolean, width
As Integer, disableCache As
Boolean) As Integer
```

Syntax

- may throw Exception()

Name	Description
html	The HTML to be rendered.
paged	Allows you to override the default <a href="#">XHtmlOptions.Paged</a> property.
	Allows you to override the default

---

**Params**

width	<a href="#">XHtmlOptions.BrowserWidth</a> property.
disableCache	Allows you to override and disable the page cache. See the <a href="#">XHtmlOptions.PageCacheEnabled</a> property for details.
return	The ID of the newly added object.

This method is essentially the same as the [AddImageUrl](#) method but it allows you to use raw HTML rather than having to specify a URL.

ABCpdf saves this HTML into a temporary file and renders the file using a 'file://' protocol specifier. So this is a convenience function - it doesn't offer any performance enhancements.

Sometimes the IIS users don't have full access to the temp directory. This is determined by the system setup you have on your machine. If this is the case you will get errors returned.

So if you are working from ASP you may find that you need to enable access to the temp directory for the ASPNET user, the IUSR\_MACHINENAME user or the IWAM\_MACHINENAME user.

---

**Notes**

**Styles and Images.** HTML does not exist within a file and so it does not have a

location.

External stylesheets and images are often referenced via relative URLs. Because the HTML has no location it is impossible to resolve these relative reference.

So you need to provide your stylesheet and image links as absolute references. Or you may be able to use the HTML BASE element to specify an appropriate base location. Or you can save your HTML to file in an appropriate location and then use AddImageUrl.

Note that the HTML BASE tag must appear in the HEAD and it must appear before other references.

---

**Example**

None.



## AddImageObject Function

---

Adds an XImage based image to the current page.

### [C#]

```
int AddImageObject(XImage image)
int AddImageObject(XImage image, bool
transparent)
```

### [Visual Basic]

Syntax

```
Function AddImageObject(image As XImage) As Integer
Function AddImageObject(image As XImage,
transparent As Boolean) As Integer
```

- may throw Exception()

Name	Description
image	An <a href="#">XImage</a> containing the image to be added page.
transparent	Whether transparency information should be preserved. This parameter overrides the <a href="#">PreserveTransparency</a> property of the <a href="#">XReadOptions</a> used to create the <a href="#">XImage</a> . If this parameter is specified and the <a href="#">XImage</a> has been created with <a href="#">XReadOptions</a> , this parameter is defaulted to

## Params

return	The ID of the newly added Image Object. If the image is added as multiple objects, such as if it has been created with an <a href="#">XReadOptions</a> with <a href="#">ReadModule</a> <a href="#">SwfVector</a> (with a non-null <a href="#">Operation</a> ), <a href="#">Xps</a> , or <a href="#">XpsAny</a> , the ID is zero. If <a href="#">XReadOptions.Operation</a> has been null, the temporary <a href="#">SwfImportOperation</a> allows this method to return the ID of the <a href="#">GraphicLayer</a> for the frame of <a href="#">XReadOptions</a> .

If the [XImage](#) has been created with an [XReadOptions](#) with [ReadModule](#) that uses an [Operation](#), the operation will be inserted and the result depends on the operation. Otherwise, this method gets an image from the Image object and adds it to the current frame, returning the ID of the newly added object.

Adds the [Selection](#) of the current [Frame](#) returning the ID of the newly added object.

Images embedded using this method are always inserted using [indirect](#) mode. Indirect mode is not as fast as [pass-through](#) mode. However it allows greater flexibility and the use of many different image formats.

If the [Transparent](#) parameter is set to true then any transparency information will be preserved. This allows you to add format as transparent GIF, PNG with alpha channel, or images with transparency set using the [Image.SetMask](#) method.

The image is scaled to fill the current [Rect](#). It is transformed using the current [Transform](#).

**Hyperlinks.** Sometimes you may wish to insert annotations such as links to an external web page specified using a

URL. The AddHtml method allows you to specify links to text. If, for example, you want to insert a link over code such as an image you need to add it separately. You can do this using code of the following form.

```
static int AddLinkToUri(Doc doc, XRect rect, string uri) {
    int id = doc.AddObject("<< /Type /Annot /Subtype /Link /A << /Type /Action /S /URI /URI ( ) >> /Border [0 0 0] >>");
    doc.SetInfo(doc.Page, "/Annots*[]:Ref", id);
    doc.SetInfo(id, "/Rect:Rect", doc.Rect.String);
    doc.SetInfo(id, "/A/URI:Text", uri);
    return id;
}
```

If you wish to link to a particular page specified by page number, you can use code of the following form.

```
static int AddLinkToPage(Doc doc, XRect rect, int pageID) {
    int id = doc.AddObject("<< /Type /Annot /Subtype /Link /Border [0 0 0] /A << /Type /Action /S /GoTo /D [ /XYZ null null 0 ] >>");
    doc.SetInfo(doc.Page, "/Annots*[]:Ref", id);
    doc.SetInfo(id, "/Rect:Rect", doc.Rect.String);
    doc.SetInfo(id, "/A/D[0]:Ref", pageID.ToString());
    return id;
}
```

---

The following code adds a transparent GIF against a gray b

### [C#]

```
XImage theImg = new XImage();
theImg.SetFile(Server.MapPath("../mypics/my
Doc theDoc = new Doc();
theDoc.Color.String = "200 200 200";
theDoc.FillRect();
theDoc.Rect.String = "0 0 0 0";
theDoc.AddImageObject(theImg, true);
theImg.Clear();
theDoc.Save(Server.MapPath("docaddimageobje
theDoc.Clear());
```

### [Visual Basic]

```
Dim theImg As XImage = New XImage()
theImg.SetFile(Server.MapPath("../mypics/my
Dim theDoc As Doc = New Doc()
theDoc.Color.String = "200 200 200"
theDoc.FillRect()
theDoc.Rect.String = "0 0 0 0"
theDoc.AddImageObject(theImg, True)
theImg.Clear()
theDoc.Save(Server.MapPath("docaddimageobje
theDoc.Clear()
```

**Example**



docaddimageobject.pdf

# AddImageToChain Function

---



Adds a new page from a paged HTML or PostScript render.

## [C#]

```
int AddImageToChain(int id)
```

## [Visual Basic]

```
Function AddImageToChain(id As Integer)  
As Integer
```

Syntax

- may throw Exception()

Name	Description
id	The ID of the previous object in the chain.
return	The ID of the newly added object.

Params

Some web pages are too large to fit on one PDF page.

To split a web page across multiple PDF pages you need to call [AddImageUrl](#) or [AddImageHtml](#) to render the first page. The ID returned from these calls represents the head of the chain.

To add subsequent pages to the chain you need to make calls to `AddImageToChain` passing in the previous image from the chain each time.

As well as using chaining to split web pages across multiple PDF pages you can also use it to split your web pages across multiple columns within a page. You can even split your chain to generate multiple copies of the same page.

---

**Notes** More information can be found in the [HTML / CSS Rendering](#) section of the documentation.

Similarly some PostScript (PS) files contain more than one page of content.

To split a PS file across multiple PDF pages you need to call `AddImageFile` or `AddImageData` to render the first page. The ID returned from these calls represents the head of the chain.

To add subsequent pages to the chain you need to make calls to `AddImageToChain` passing in the previous image from the chain each time.

This example shows how to import an HTML page into a multi-page PDF document.

We first create a `Doc` object and inset the edges a little so that the HTML will appear in the middle of the page.

**[C#]**

```
Doc theDoc = new Doc();  
theDoc.Rect.Inset(72, 144);
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Rect.Inset(72, 144)
```

We add the first page of HTML. We save the returned ID as this will be used to add subsequent pages.

### [C#]

```
int theID;  
theID =  
theDoc.AddImageUrl("http://www.yahoo.com/")
```

### [Visual Basic]

```
Dim theID As Integer  
theID =  
theDoc.AddImageUrl("http://www.yahoo.com/")
```

We now chain subsequent pages together. We stop when we reach a page which wasn't truncated.

### [C#]

```
while (true) {  
    theDoc.FrameRect();  
    if (!theDoc.Chainable(theID))  
        break;  
    theDoc.Page = theDoc.AddPage();  
    theID = theDoc.AddImageToChain(theID);  
}
```

### [Visual Basic]

```
While True  
    theDoc.FrameRect()  
    If Not theDoc.Chainable(theID) Then  
        Exit While  
    End If
```

---

**Example**

```
theDoc.Page = theDoc.AddPage()  
theID = theDoc.AddImageToChain(theID)  
End While
```

After adding the pages we can flatten them. We can't do this until after the pages have been added because flattening will invalidate our previous ID and break the chain.

**[C#]**

```
for (int i = 1; i <= theDoc.PageCount; i++)  
    theDoc.PageNumber = i;  
    theDoc.Flatten();  
}
```

**[Visual Basic]**

```
For i As Integer = 1 To theDoc.PageCount  
    theDoc.PageNumber = i  
    theDoc.Flatten()  
Next
```

Finally we save.

**[C#]**

```
theDoc.Save(Server.MapPath("pagedhtml.pdf"))  
theDoc.Clear();
```

**[Visual Basic]**

```
theDoc.Save(Server.MapPath("pagedhtml.pdf"))  
theDoc.Clear()
```

We get the following output.





# AddImageUrl Function

---

Renders a web page specified by URL.

## [C#]

```
int AddImageUrl(string url)
int AddImageUrl(string url, bool paged,
int width, bool disableCache)
```

## [Visual Basic]

Syntax

```
Function AddImageUrl(url As String) As Integer
Function AddImageUrl(url As String, paged As Boolean, width As Integer, disableCache As Boolean) As Integer
```

- may throw `Exception()`

Name	Description
url	The URL for the page to be rendered. The value may be modified depending on the value of <code>disableCache/XHtmlOptions.PageCacheEnabled</code> .
paged	Allows you to override the default <code>XHtmlOptions.Paged</code> property.
	Allows you to override the default

Params	
width	<a href="#">XHtmlOptions.BrowserWidth</a> property.
disableCache	Allows you to override and disable the page cache. See the <a href="#">XHtmlOptions.PageCacheEnabled</a> property for details.
return	The ID of the newly added object.

This method adds a web page to a document.

The page is added in accordance with the current [XHtmlOptions](#) settings. As a convenience you can override the more commonly used settings as detailed above.

Only the first page of the document is drawn. Subsequent pages can be drawn using the [AddImageToChain](#) method.

The web page is scaled to fill the current [Rect](#). It is transformed using the current [Transform](#).

**Caching.** Sometimes you may find that pages appear to be cached.

If you are using [AddImageUrl](#) it is possible that the URL is in some way being cached. So the PDF may be changing but the content within it may be staying the same. See the [HTML / CSS Rendering](#) section of the documentation for details.

Alternatively it is possible that the PDF itself is being cached. Most commonly this can happen if you're

streaming the PDF direct to the browser and you have certain IIS settings (like Expire Content) disabled.

Your first step should be to narrow down the problem. Why not save the PDF to disk at the same time as sending it to the client? That way you can establish whether the PDF itself is being cached or whether the content is in some way being cached (resulting in the same PDF being created again and again).

If the PDF is being cached you will need to look at your IIS settings. ABCpdf is not doing the caching (and indeed it cannot cache the PDF in this way) it will be something which is happening either in IIS/ASP or on an intervening proxy server or on the client.

In addition to accepting URLs to web pages, this method also accepts file based URLs to MHT (MIME HTML) files.

MHT files contain a web page and any associated resources (such as images and style sheets) in one compact archive. You can save web pages in MHT format using IE.

Note that MHT files saved from more complex web pages sometimes omit some required resources. In this situation ABCpdf will attempt to download missing items from the original URL on the web. However if these items are no longer available then ABCpdf may not be able to produce a perfect output.

We create an ABCpdf Doc object, add our URL and save. T

it!

### [C#]

```
Doc theDoc = new Doc();  
theDoc.AddImageUrl("http://www.google.com/"  
theDoc.Save(Server.MapPath("htmlimport.pdf")  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.AddImageUrl("http://www.google.com/"  
theDoc.Save(Server.MapPath("htmlimport.pdf")  
theDoc.Clear()
```

### Example



htmlimport.pdf

For an example of how to use paged HTML see the [AddImageToChain](#) method.





## AddLine Function

---

Adds a line to the current page.

### [C#]

```
int AddLine(double x1, double y1, double x2, double y2)
```

---

### Syntax

### [Visual Basic]

```
Function AddLine(x1 As Double, y1 As Double, x2 As Double, y2 As Double) As Integer
```

---

### Params

Name	Description
x1	The horizontal offset of the start point.
x2	The horizontal offset of the end point.
y1	The vertical offset of the start point.
y2	The vertical offset of the end point.
return	The Object ID of the newly added Graphic Object.

Adds a line to the current page. The line is drawn in the current **color** at the current **width** and with the current **options**.

---

**Notes** The AddLine function returns the Object ID of the newly added Graphic Object.

The following code adds two horizontal lines to a document first is blue and the second is green.

### [C#]

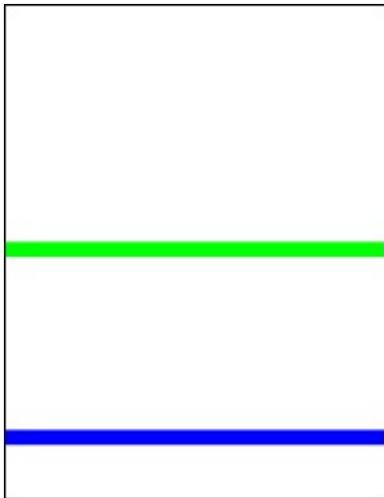
```
Doc theDoc = new Doc();
theDoc.Width = 24;
theDoc.Color.String = "0 0 255";
theDoc.AddLine(-50, 100, 999, 100);
theDoc.Color.String = "0 255 0";
theDoc.AddLine(-50, 400, 999, 400);
theDoc.Save(Server.MapPath("docaddline.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Width = 24
theDoc.Color.String = "0 0 255"
theDoc.AddLine(-50, 100, 999, 100)
theDoc.Color.String = "0 255 0"
theDoc.AddLine(-50, 400, 999, 400)
theDoc.Save(Server.MapPath("docaddline.pdf"))
theDoc.Clear()
```

---

**Example**



docaddline.pdf



# AddObject Function

---

Adds a native PDF object to the document.

## [C#]

```
int AddObject(string text)
```

---

## Syntax

## [Visual Basic]

```
Function AddObject(text As String) As Integer
```

---

## Params

Name	Description
text	The raw native object to be added.
return	The Object ID of the newly added Object.

You will not normally need to use this feature. However it can be useful if you wish to add objects which are defined in the PDF specification but not supported by ABCpdf .NET.

Be aware that the text you pass this function must be in native PDF format. This means that unusual characters in text strings must be appropriately escaped. For full details of the way that PDF objects are represented you should see the Adobe PDF Specification.

---

## Notes

Your newly added object needs to be referenced from somewhere in the PDF document. If you do not reference your object it

orphaned and will be deleted when the document is saved.

The following code adds a document information section to PDF document. First it adds an empty dictionary and refers to the document trailer. Then it adds an Author, Title and Subject saving.

There are multiple places that metadata can be put into a PDF. Commonly used are the Info entry of the Trailer and the Metadata entry of the Catalog. The Info entry is the older and most widely used location. The Metadata entry is a more recent XML based standard. It is important that information within these stores is consistent. If the information is inconsistent then you'll find that the metadata from different applications is different. To ensure that the data is consistent we're going to delete any XML Metadata entry that may be present. The way we force applications to report the Info store. However, it is a difficult matter to load up any XML in the Metadata entry as well as that as well as the Info entry.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.pdf"));
if (theDoc.GetInfo(-1, "/Info") == "")
    theDoc.SetInfo(-1, "/Info:Ref", theDoc.Ac
    << >>).ToString());
theDoc.SetInfo(-1, "/Info*/Author:Text", "A
Dent");
theDoc.SetInfo(-1, "/Info*/Title:Text", "Mu
Life");
theDoc.SetInfo(-1, "/Info*/Subject:Text",
"Philosophy");
theDoc.SetInfo(theDoc.Root, "/Metadata:Del
theDoc.Save(Server.MapPath("docaddobject.pdf"));
theDoc.Clear();
```

Example

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample1.jpg"))
If theDoc.GetInfo(- 1, "/Info") = "" Then
    theDoc.SetInfo(- 1, "/Info:Ref",
theDoc.AddObject("<< >>").ToString())
End If
theDoc.SetInfo(- 1, "/Info*/Author:Text", "M. J. Dent")
theDoc.SetInfo(- 1, "/Info*/Title:Text", "My Life")
theDoc.SetInfo(- 1, "/Info*/Subject:Text", "Philosophy")
theDoc.SetInfo(theDoc.Root, "/Metadata:Delete")
theDoc.Save(Server.MapPath("docaddobject.png"))
theDoc.Clear()
```



# AddOval Function

---

Adds an oval to the current page.

## [C#]

```
int AddOval(bool filled)
```

---

## Syntax

## [Visual Basic]

```
Function AddOval(filled As Boolean) As Integer
```

---

## Params

Name	Description
filled	Whether to fill the oval rather than simply outline it.
return	The Object ID of the newly added Graphic Object.

Adds an oval to the current page. The oval is drawn in the current [color](#) at the current [width](#) and with the current [options](#). It is scaled to fill the current [rectangle](#). The oval may be outlined or filled.

---

## Notes

The AddOval function returns the Object ID of the newly added Graphic Object.

The following code adds two ovals to a document. The outline oval is semi-transparent.

### [C#]

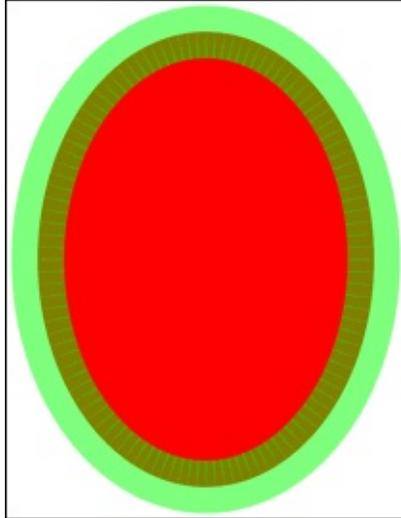
```
Doc theDoc = new Doc();
theDoc.Width = 80;
theDoc.Rect.Inset(50, 50);
theDoc.Color.String = "255 0 0";
theDoc.AddOval(true);
theDoc.Color.String = "0 255 0 a128";
theDoc.AddOval(false);
theDoc.Save(Server.MapPath("docaddoval.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Width = 80
theDoc.Rect.Inset(50, 50)
theDoc.Color.String = "255 0 0"
theDoc.AddOval(True)
theDoc.Color.String = "0 255 0 a128"
theDoc.AddOval(False)
theDoc.Save(Server.MapPath("docaddoval.pdf"))
theDoc.Clear()
```

---

Example



docaddoval.pdf



## AddPage Function

---

Adds a page to the current document.

### [C#]

```
int AddPage()  
int AddPage(int page)
```

---

### Syntax

### [Visual Basic]

```
Function AddPage() As Integer  
Function AddPage(page As Integer) As  
Integer
```

---

### Params

Name	Description
page	The page insertion location. By default pages are added at the end of the document.
return	The Object ID of the newly added Page Object.

Adds a page to the current document.

The AddPage function returns the Object ID of the newly

added Page Object. Typically you will want to assign this return value to the document Page property using code of the form.

```
theDoc.Page = theDoc.AddPage()
```

---

**Notes**

Pages are added at the end of the document. However you can use the PageNum parameter to insert pages at other locations. The following code inserts a page at the start of a document.

```
theDoc.Page = theDoc.AddPage(1)
```

Any existing page and all subsequent pages will be shifted towards the end of the document to make room for the insertion.

The following code adds three pages to a document. Each is marked with the page number and page Object ID.

**[C#]**

```
Doc theDoc = new Doc();
theDoc.FontSize = 96; // big text
theDoc.TextStyle.HPos = 0.5; // centered
theDoc.TextStyle.VPos = 0.5; // ...
for (int i = 1; i <= 3; i++) {
    theDoc.Page = theDoc.AddPage();
    string txt;
    txt = "Page " + i.ToString() + ", ID=";
    txt = txt + theDoc.Page.ToString();
    theDoc.AddText(txt);
}
```

```
theDoc.Save(Server.MapPath("docaddpage.pdf")
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96 ' big text
theDoc.TextStyle.HPos = 0.5 ' centered
theDoc.TextStyle.VPos = 0.5 ' ...
For i As Integer = 1 To 3
    theDoc.Page = theDoc.AddPage()
    Dim txt As String
    txt = "Page " + i.ToString() + ", ID="
    txt = txt + theDoc.Page.ToString()
    theDoc.AddText(txt)
```

Example

```
Next
theDoc.Save(Server.MapPath("docaddpage.pdf")
theDoc.Clear()
```

Page 1, ID=4

Page 2, ID=7

Page 3, ID=9

docaddpage.pdf



## AddPie Function

---

Adds a pie slice to the current page.

### [C#]

```
int AddPie(double angleStart, double angleEnd, bool filled)
```

---

### Syntax

### [Visual Basic]

```
Function AddPie(angleStart As Double, angleEnd As Double, filled As Boolean) As Integer
```

---

### Params

Name	Description
angleStart	The start angle of the pie slice in degrees.
angleEnd	The end angle of the pie slice in degrees.
filled	Whether to fill the pie slice rather than simply outline it.
return	The Object ID of the newly added Graphic Object.

Adds a pie slice to the current page. The slice is drawn in the current [color](#) at the current [width](#) and with the current [options](#).

---

**Notes**

The pie slice represents a segment of the oval which would fill the current [rectangle](#). Drawing starts at the start angle and the arc is swept out until the end angle is reached. Angles are measured anti-clockwise with zero at three o'clock. The slice may be outlined or filled depending on the values passed to the function.

The AddPie function returns the Object ID of the newly added Graphic Object.

The following code adds two pie slices to a document.

**[C#]**

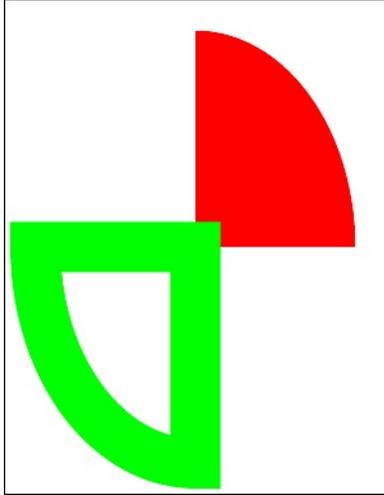
```
Doc theDoc = new Doc();
theDoc.Width = 80;
theDoc.Rect.Inset(50, 50);
theDoc.Color.String = "255 0 0";
theDoc.AddPie(0, 90, true);
theDoc.Color.String = "0 255 0";
theDoc.AddPie(180, 270, false);
theDoc.Save(Server.MapPath("docaddpie.pdf"));
theDoc.Clear();
```

**[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
theDoc.Width = 80
theDoc.Rect.Inset(50, 50)
theDoc.Color.String = "255 0 0"
theDoc.AddPie(0, 90, True)
theDoc.Color.String = "0 255 0"
theDoc.AddPie(180, 270, False)
theDoc.Save(Server.MapPath("docaddpie.pdf"))
theDoc.Clear()
```

---

**Example**



docaddpie.pdf



# AddPoly Function

---

Adds a polygon to the current page.

## [C#]

```
int AddPoly(string points, bool filled)
int AddPoly(double[] points, bool filled)
int AddPoly(double[] points, int index,
int count, bool filled)
```

## [Visual Basic]

### Syntax

```
Function AddPoly(points As String, filled
As Boolean) As Integer
Function AddPoly(points() As Double,
filled As Boolean) As Integer
Function AddPoly(points() As Double,
index As Integer, count As Integer,
filled As Boolean) As Integer
```

### Params

Name	Description
points	The coordinates of the vertices of the polygon.
index	The index of the first coordinate into the array points.
count	The number of coordinates in the array points to use.
filled	Whether to fill the polygon rather than simply outline it.
return	The Object ID of the newly added Graphic

## Object.

Adds a polygon to the current page. The polygon is drawn in the current **color** at the current **width** and with the current **options**. The polygon may be outlined or filled.

The points string is a sequence of numbers representing the coordinates of the polygon. The string should be of the format "x1 y1 x2 y2 ... xN yN". The numbers may be delimited with spaces, commas or semicolons. If the first point is equal to the last then the path is closed before outlining.

### Notes

The AddPoly function returns the Object ID of the newly added Graphic Object.

The following code adds a transparent green outlined star c the top of a red filled star.

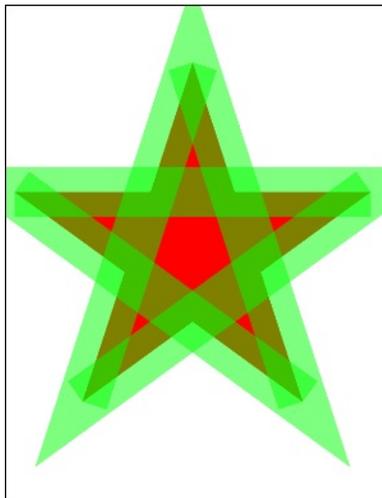
### [C#]

```
Doc theDoc = new Doc();
theDoc.Width = 80;
theDoc.Color.String = "255 0 0";
theDoc.AddPoly("124 158 300 700 476 158 15
585 493 124 158", true);
theDoc.Color.String = "0 255 0 a128";
theDoc.AddPoly("124 158 300 700 476 158 15
585 493 124 158", false);
theDoc.Save(Server.MapPath("docaddpoly.pdf")
theDoc.Clear();
```

## [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()  
theDoc.Width = 80  
theDoc.Color.String = "255 0 0"  
theDoc.AddPoly("124 158 300 700 476 158 15  
585 493 124 158", True)  
theDoc.Color.String = "0 255 0 a128"  
theDoc.AddPoly("124 158 300 700 476 158 15  
585 493 124 158", False)  
theDoc.Save(Server.MapPath("docaddpoly.pdf")  
theDoc.Clear()
```



docaddpoly.pdf



## AddRect Function

---

Add a rectangle to the current page.

### [C#]

```
int AddRect(bool filled)
```

---

### Syntax

### [Visual Basic]

```
Function AddRect(filled As  
Boolean) As Integer
```

---

### Params

Name	Description
filled	Whether to fill the rectangle rather than simply outline it.
return	The Object ID of the newly added Graphic Object.

Add a rectangle drawn in the current [color](#) at the current [width](#) and with the current [options](#). The rectangle may be outlined or filled.

---

### Notes

Note that this is subtly different from the way that `FrameRect` works. When a rectangle is framed the line goes around the outside of the rectangle. When a rectangle is added the line is centered on the rectangle - so half is inside and

half is outside.

Example

None.



## AddText Function

---

Adds a block of text to the current page.

### [C#]

```
int AddText(string text)
```

---

### Syntax

### [Visual Basic]

```
Function AddText(text As String) As Integer
```

---

### Params

Name	Description
text	The text to be added to the page.
return	The Object ID of the newly added Text Object.

Adds a block of single styled text to the current page.

For adding multi-styled text or for chaining text from one page to another you should use the [AddHtml](#) method which is used for adding [HTML styled text](#).

The text is in the current [style](#), [size](#) and [color](#) and starts at the location specified in the current [position](#). If the text is long it will will wrap and extend downwards until it fills the current [rectangle](#). Text positioning in the rectangle is determined by the [horizontal](#) and [vertical](#) positioning.

You can chain together multiple text blocks so that text flows from one to the next. To do this you need to first add a block of text using `AddText`. Then add multiple new text blocks using `AddHtml` each time passing in the ID obtained from the previous call after adjusting the target location (such as the rectangle or the page).

The `AddText` function returns the Object ID of the newly added Text Object. If no text could be added then zero is returned. This will happen if a zero length string was supplied or if the rectangle was too small for even one character to be displayed.

---

#### Notes

Typically you will get a return value of zero if your text was too large to fit in your `Rect` or if the `Pos` was at the end of the `Rect`. So if you are expecting text to be displayed and are seeing a return value of zero, check your text size, check your `Rect` is where you think it is by framing it using `FrameRect` and ensure your `Pos` is set at the top left of the `Rect`.

Text is drawn word-wrapped within the current rectangle with the first character at the location specified by the `Pos` property. Normally the `Pos` property reflects the top left position of the current rectangle. However if you need to alter the position at which text drawing starts you can modify the `Pos` property after changing the `Rect`. When the text has been drawn the `Pos` will be updated to reflect the next text insertion point.

Character positioning is specified from the top left of the character. Please see `Pos` for details on positioning when using vertical fonts. The `FontSize` determines the total line height and the character baseline is 80% of the way down from the top of the line.

The following code adds a number of chunks of text to a document. Each chunk is in a different style. This sample makes use of the fact that the Pos is updated to point to the next text insertion point after adding a piece of text. However note that when inserting multi-styled text it is generally more efficient to use the [AddHtml](#) method.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Page = theDoc.AddPage();
theDoc.FontSize = 48;
int theF1 = theDoc.AddFont("Times-Roman");
int theF2 = theDoc.AddFont("Times-Bold");
theDoc.Font = theF1;
theDoc.AddText("Gallia est omnis ");
theDoc.Font = theF2;
theDoc.AddText("tertiam Galli appellantur ");
theDoc.Font = theF1;
theDoc.AddText("divisa in partes tres, ");
theDoc.Font = theF2;
theDoc.AddText("quarum unam incolunt ");
theDoc.Font = theF1;
theDoc.AddText("Belgae, aliam Aquitani. ");
theDoc.Font = theF2;
theDoc.AddText("tertiam Galli appellantur");
theDoc.Save(Server.MapPath("docaddtext.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Page = theDoc.AddPage()
theDoc.FontSize = 48
Dim theF1 As Integer = theDoc.AddFont("Time
Roman")
Dim theF2 As Integer = theDoc.AddFont("Time
Bold")
```

---

**Example**

```
theDoc.Font = theF1
theDoc.AddText("Gallia est omnis ")
theDoc.Font = theF2
theDoc.AddText("tertiam Galli appellantur ")
theDoc.Font = theF1
theDoc.AddText("divisa in partes tres, ")
theDoc.Font = theF2
theDoc.AddText("quarum unam incolunt ")
theDoc.Font = theF1
theDoc.AddText("Belgae, aliam Aquitani. ")
theDoc.Font = theF2
theDoc.AddText("tertiam Galli appellantur")
theDoc.Save(Server.MapPath("docaddtext.pdf"))
theDoc.Clear()
```

Gallia est omnis **tertiam Galli**  
**appellantur** divisa in partes  
tres, **quarum unam incolunt**  
Belgae, aliam Aquitani.  
**tertiam Galli appellantur**

docaddtext.pdf

# AddXObject Function

---



Add a Form or Image XObject to the current page.

## [C#]

```
int AddXObject(Objects.Pixmap pm)
int AddXObject(Objects.FormXObject pm)
```

---

## Syntax

## [Visual Basic]

```
Function AddXObject(pm As
Objects.Pixmap) As Integer
Function AddXObject(pm As
Objects.FormXObject) As Integer
```

---

## Params

Name	Description
pm	The image to be added to the page.
return	The Object ID of the newly added Image Object.

Add a Form or Image XObject into the current [rectangle](#) on the current page.

Form and Image XObjects represent a drawing sequence external to the main content stream of

the page. Image XObjects represent a raster bitmap in one of a variety of color spaces. Form XObjects represent a separate content stream with a separate sequence of drawing commands.

---

#### Notes

For example a Form XObject might be used to represent a "Draft" stamp which might then be applied on various pages at various locations. The fact that this stamp is an external object allows the viewing application to cache the representation and also allows changes to the stamp to affect the appearance of multiple instances.

This example shows how to load an image into a Pixmap and then draw it on the current page using the AddXObject method.

#### [C#]

```
using (Doc doc = new Doc()) {
    doc.Rect.Inset(50, 50);
    doc.Transform.Rotate(20, 200, 200);
    doc.Color.SetRgb(200, 200, 255);
    doc.FillRect();
    Pixmap pm = new
Pixmap(doc.ObjectSoup);
    Image img =
Image.FromFile("mypic.png");
    pm.SetBitmap(img as Bitmap, true);
    doc.AddXObject(pm);
    doc.Save("examplePixmapBitmap.pdf");
}
```

## [Visual Basic]

Example

```
Sub ...
    Using doc As New Doc()
        doc.Rect.Inset(50, 50)
        doc.Transform.Rotate(20, 200, 200)
        doc.Color.SetRgb(200, 200, 255)
        doc.FillRect()
        Dim pm As New
        PixMap(doc.ObjectSoup)
        Dim img As Image =
        Image.FromFile("mypic.png")
        pm.SetBitmap(TryCast(img, Bitmap),
        True)
        doc.AddXObject(pm)
        doc.Save("examplePixMapBitmap.pdf")
    End Using
End Sub
```



examplePixMapBitmap.pdf





# Append Function

---

Appends a PDF to the end of the document.

## [C#]

```
void Append(Doc doc)
```

---

## Syntax

## [Visual Basic]

```
Sub Append(doc As Doc)
```

---

## Params

Name	Description
doc	The document to add to the end of this one.

Use this method to append one PDF to the end of another one.

Individual pages from one PDF can be drawn into another using the [AddImageDoc](#) method.

If you are inserting a number of pages it is much faster to use the Append method than to draw pages individually. It also has the advantage of maintaining other information such as bookmarks.

If you are inserting pages that contain [form](#) fields, you

---

**Notes**

may want to call [MakeFieldsUnique](#) to avoid sharing fields across pages.

The [Refactor](#) setting determines whether new/modified redundant objects are eliminated. The [Preflight](#) setting determines whether objects in the destination document are validated before this operation is performed. Unless the document and the pages are big in terms of memory use and have many common objects, it is faster to disable refactor and preflight for adding the pages and enable them for saving the document. You can use [SetInfo](#) to change these settings.

The following code snippet illustrates how one might join two PDF documents together.

**[C#]**

```
Doc theDoc1 = new Doc();
theDoc1.FontSize = 192;
theDoc1.TextStyle.HPos = 0.5;
theDoc1.TextStyle.VPos = 0.5;
theDoc1.AddText("Hello");
Doc theDoc2 = new Doc();
theDoc2.FontSize = 192;
theDoc2.TextStyle.HPos = 0.5;
theDoc2.TextStyle.VPos = 0.5;
theDoc2.AddText("World");
theDoc1.Append(theDoc2);
theDoc1.Save(Server.MapPath("docjoin.pdf"));
theDoc1.Clear();
theDoc2.Clear();
```

**[Visual Basic]**

```
Dim theDoc1 As Doc = New Doc()  
theDoc1.FontSize = 192  
theDoc1.TextStyle.HPos = 0.5  
theDoc1.TextStyle.VPos = 0.5  
theDoc1.AddText("Hello")  
Dim theDoc2 As Doc = New Doc()  
theDoc2.FontSize = 192  
theDoc2.TextStyle.HPos = 0.5  
theDoc2.TextStyle.VPos = 0.5  
theDoc2.AddText("World")  
theDoc1.Append(theDoc2)  
theDoc1.Save(Server.MapPath("docjoin.pdf"))  
theDoc1.Clear()  
theDoc2.Clear()
```

---

**Example**



docjoin.pdf [Page 1]

**World**

docjoin.pdf [Page 2]



# Chainable Function

---

Determines if an object is chainable.

## [C#]

```
bool Chainable(int id)
```

---

## Syntax

## [Visual Basic]

```
Function Chainable(id As Integer)  
As Boolean
```

---

## Params

Name	Description
id	The Object ID of the object to be tested.
return	Whether the object is chainable or not.

Use this method to determine if an object is chainable.

Some objects can be chained. A chunk of text may be chained from page to page on the output PDF. Similarly a web page may be chained from page to page.

---

**Notes**

This method allows you to determine if the object you have added is chainable or whether it is at the end of the chain.

Only text, PostScript images and web pages can be chainable. So your ID should have been obtained from a previous call to one of the [AddImage](#) family of calls or from [AddHtml](#).

---

**Example**

See the [AddImageToChain](#) method.



# Clear Function

---

Clears the document.

## [C#]

```
void Clear()
```

---

### Syntax

## [Visual Basic]

```
Sub Clear()
```

---

### Params

Name	Description
none	

---

### Notes

Use this method to release resources and return the document to a just-created state.

---

### Example

None.





# ClearCachedDecompressedStreams Function

---

Clears the cached, decompressed data for stream objects.

## [C#]

```
void ClearCachedDecompressedStreams()  
void  
ClearCachedDecompressedStreams(StreamObject  
types)
```

---

## Syntax

## [Visual Basic]

```
Sub ClearCachedDecompressedStreams()  
Sub ClearCachedDecompressedStreams(types As  
StreamObjectTypes)
```

---

## Params

Name	Description
types	The types of stream objects whose cached, decompressed data are discarded. The default value is All.

Use this method to discards the cached, decompressed data for multiple stream objects.

The StreamObjectTypes enumeration may take a combination of the following values:

- None – no stream object.
- All – all stream objects.
- Others – stream objects not of the below types.
- PixMap
- FormXObject – Form XObject.
- TilingPattern – tiling pattern object.
- StreamShadingObject – shading objects that are streams.
- StreamFunction – function objects that are streams.

---

Notes

You can apply the effect to a single stream object using [StreamObject.ClearCachedDecompressed](#).

None.

---

Example



## Delete Function

---

Deletes an object previously added to the document.

### [C#]

```
void Delete(int id)
```

---

### Syntax

### [Visual Basic]

```
Sub Delete(id As Integer)
```

---

### Params

Name	Description
id	The Object ID of the object to be deleted.

Use this method to delete an object previously added to the document.

Deletion is most commonly applied to pages to remove them from the document. For example to delete the current page you might use the following code:

```
theDoc.Delete theDoc.Page
```

---

### Notes

The following code snippet illustrates how one might add an image and then delete it if the image color space is CMYK.

### [C#]

```
Doc theDoc = new Doc();
string thePath =
Server.MapPath("../mypics/mypic.jpg");
int theID1 = theDoc.AddImageFile(thePath, 1);
int theID2 = theDoc.GetInfoInt(theID1,
"XObject");
int theComps = theDoc.GetInfoInt(theID2,
"Components");
if (theComps == 4) theDoc.Delete(theID1);
theDoc.Save(Server.MapPath("docdelete.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim thePath As String =
Server.MapPath("../mypics/mypic.jpg")
Dim theID1 As Integer =
theDoc.AddImageFile(thePath, 1)
Dim theID2 As Integer =
theDoc.GetInfoInt(theID1, "XObject")
Dim theComps As Integer =
theDoc.GetInfoInt(theID2, "Components")
If theComps = 4 Then theDoc.Delete(theID1)
theDoc.Save(Server.MapPath("docdelete.pdf"))
theDoc.Clear()
```

---

Example



docdelete.pdf



# EmbedFont Function

---

Embeds a font into the document.

## [C#]

```
int EmbedFont(string name)
int EmbedFont(string name, LanguageType
language)
int EmbedFont(string name, LanguageType
language, bool vertical)
int EmbedFont(string name, LanguageType
language, bool vertical, bool subset)
int EmbedFont(string name, LanguageType
language, bool vertical, bool subset, bool
force)
```

## [Visual Basic]

—Syntax

```
Function EmbedFont(name As String) As
Integer
Function EmbedFont(name As String, language
As LanguageType) As Integer
Function EmbedFont(name As String, language
As LanguageType, vertical As Boolean) As
Integer
Function EmbedFont(name As String, language
As LanguageType, vertical As Boolean,
subset As Boolean) As Integer
Function EmbedFont(name As String, language
As LanguageType, vertical As Boolean,
subset As Boolean, force As Boolean) As
Integer
```

Name	Description
name	The name of the font typeface.
language	<p>The language type to use. The LanguageType enumeration may take the following values:</p> <ul style="list-style-type: none"> <li>• Latin</li> <li>• Unicode</li> <li>• Korean</li> <li>• Japanese</li> <li>• ChineseS</li> <li>• ChineseT</li> </ul> <p>See the <a href="#">Fonts and Languages</a> section for details on language types. The default language type is LanguageType.Latin.</p>
vertical	<p>Whether the text direction should be vertical.</p> <p>See the <a href="#">Fonts and Languages</a> section for details on writing directions. The default is false to indicate standard left to right layout.</p>
subset	<p>Whether to subset the font.</p> <p>See the <a href="#">Fonts and Languages</a> section for details on subsetting. The default is false to indicate that the font should not be subsetting.</p>
	<p>Whether to override permissions on the font.</p> <p>Fonts often contain embedded licensing</p>

---

**Params**

force	<p>information. By default ABCpdf will prevent you from embedding fonts which indicate that embedding is not permitted.</p> <p>You can force the font to be embedded by passing true for this value.</p>
return	<p>The Object ID of the newly embedded Font Object.</p>

Embeds a font into the document.

The font name, a description of the font style and the the font glyph descriptions themselves are placed into the document. This ensures that the document will always display perfectly on every system and that font substitutions will never need to be made.

There are a number of reasons you may not wish to embed fonts and instead use the [AddFont](#) method. Embedding fonts can increase the size of your PDF considerably. Additionally by distributing a PDF with an embedded font you are actually redistributing the font itself. You should check with your font supplier or legal department that you have permission to do this.

For information on how to specify font names see the [AddFont](#) method.

The EmbedFont function returns the Object ID of the newly added Font Object. Typically you will want to assign this return value to the document Font property using code of the form.

```
theDoc.Font = theDoc.AddFont("Courier");
```

If the specified font could not be found then you will get an Object ID of zero returned. You may wish to check for this possibility otherwise a default font will be used.

Fonts are cached so newly added fonts will not be available to ABCpdf until the application is restarted. If you need to dynamically load a font you can pass this method a path to your font file. This will add the font to the cache and make it available for use. You should not move, rename or delete a font file which has been dynamically loaded using this technique.

---

#### Notes

### **Why is my language a string?**

In older versions of ABCpdf the language parameter was a string. So you might find code of this form.

```
theDoc.EmbedFont("Courier", "Unicode")
```

In more recent releases the language parameter has been changed to a true enumeration. This is a safer way of coding as it allows the compiler to ensure that the values you are using are valid. Your new code should look like this.

```
theDoc.EmbedFont("Courier",  
LanguageType.Unicode)
```

The names of the items in the LanguageType enumeration are the same as the values of the strings used in previous versions. So changing your code should be a simple search and replace operation. If you see a language of "" this equates to LanguageType.Latin.

The following code embeds the font 'Comic Sans MS' into the document and then adds some text.

### **[C#]**

```
Doc theDoc = new Doc();
theDoc.FontSize = 216;
string theFont = "Comic Sans MS";
theDoc.Font = theDoc.EmbedFont(theFont);
theDoc.AddText(theFont);
theDoc.Save(Server.MapPath("docembedfont.pc
theDoc.Clear());
```

### **[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 216
Dim theFont As String = "Comic Sans MS"
theDoc.Font = theDoc.EmbedFont(theFont)
theDoc.AddText(theFont)
theDoc.Save(Server.MapPath("docembedfont.pc
theDoc.Clear())
```

---

**Example**

Comic  
Sans  
MS

docembedfont.pdf



## FillRect Function

---

Adds a painted rectangle to the current page.

### [C#]

```
int FillRect()  
int FillRect(double radiusX, double  
radiusY)
```

---

### Syntax

### [Visual Basic]

```
Function FillRect() As Integer  
Function FillRect(radiusX As Double,  
radiusY As Double) As Integer
```

Name	Description
radiusX	The horizontal radius to use for rounded corners.
radiusY	The vertical radius to use for rounded corners.
return	The Object ID of the newly added Graphic Object.

---

### Params

Adds a painted rectangle to the current page. The rectangle location and size is determined by the current [rectangle](#), the fill color is determined by the current [color](#) and any options are determined by the current [options](#).

By specifying values for the horizontal and vertical radius

parameters you can draw rectangles with rounded corners. The values refer to the radii of the ellipse used to draw the corners.

---

#### Notes

By setting the horizontal radius parameter to half the width of the rect and the vertical radius parameter to half the height of the rect you can draw filled ovals and circles.

The FillRect function returns the Object ID of the newly added Graphic Object.

The following code adds a blue filled rectangle to a document. The frame is inset from the edges of the document by 200 points horizontally and 100 points vertically.

#### [C#]

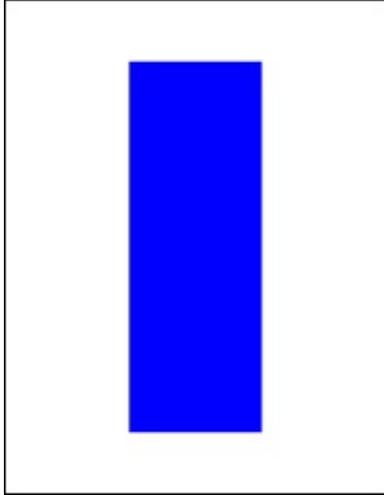
```
Doc theDoc = new Doc();
theDoc.Rect.Inset(200, 100);
theDoc.Color.Blue = 255;
theDoc.FillRect();
theDoc.Save(Server.MapPath("docfillrect.pdf"));
theDoc.Clear();
```

#### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(200, 100)
theDoc.Color.Blue = 255
theDoc.FillRect()
theDoc.Save(Server.MapPath("docfillrect.pdf"))
theDoc.Clear()
```

---

#### Example



docfillrect.pdf



## FitHtml Function

---

Fit a block of HTML styled text into the current rectangle.

### [C#]

```
int FitHtml(string text)
```

---

### Syntax

### [Visual Basic]

```
Function FitHtml(text As String)  
As Integer
```

---

### Params

Name	Description
text	The HTML to be added to the page.
return	The Object ID of the newly added Text Object.

Fit a block of HTML styled text into the current rectangle on the current page.

---

### Notes

This function is similar to [AddHtml](#) but can be used in situations in which you have a set area into which you know your text should be fitted. The call will take the base text supplied and scale it appropriately until it fits as exactly as possible into the current [Rect](#).

---

**Example**

None.



## FitText Function

---

Fit a block of text into the current rectangle on the current page.

### [C#]

```
int FitText(string text)
```

---

### Syntax

### [Visual Basic]

```
Function FitText(text As String)  
As Integer
```

---

### Params

Name	Description
text	The text to be added to the page.
return	The Object ID of the newly added Text Object.

Fit a block of text into the current rectangle on the current page.

This function is similar to [AddText](#) but can be used in situations in which you have a set area into which you know your text should be fitted. The call will take the base text supplied and scale it appropriately until it fits as exactly as possible into the current [Rect](#).

---

### Notes

For fitting multi-styled text you should use the `FitHtml` method which is used for adding `HTML` styled text.

---

**Example**

None.



# Flatten Function

---

Flattens and compresses the current page.

## [C#]

```
int Flatten()
```

---

## Syntax

## [Visual Basic]

```
Function Flatten() As Integer
```

---

## Params

Name	Description
return	n/a.

Objects added to a page are stored as individual layers. Calling this method combines all the layers on the current page and then re-compresses the layer data.

For pages that contain only a few layers the reduction in size will be minimal. However for pages which contain complex tables with many items, flattening can reduce the size of the output PDF by a factor of five or more.

---

## Notes

Note that flattening will delete all the items currently on the page and replace them with a new compressed item. This means that Object IDs previously obtained from calls such as [AddText](#) or [FrameRect](#) will no longer be valid.

---

**Example**

See the [Small Table Example](#) and [Large Table Example](#).



# FrameRect Function

---

Adds a rectangular frame to the current page.

## [C#]

```
int FrameRect()  
int FrameRect(bool inside)  
int FrameRect(double radiusX, double  
radiusY)  
int FrameRect(double radiusX, double  
radiusY, bool inside)
```

## [Visual Basic]

### Syntax

```
Function FrameRect() As Integer  
Function FrameRect(inside As Boolean) As  
Integer  
Function FrameRect(radiusX As Double,  
radiusY As Double) As Integer  
Function FrameRect(radiusX As Double,  
radiusY As Double, inside As Boolean) As  
Integer
```

### Params

Name	Description
radiusX	The horizontal radius to use for rounded corners.
radiusY	The vertical radius to use for rounded corners.
inside	Whether to draw the frame inside the rectangle.
return	The Object ID of the newly added Graphic Object.

Adds a rectangular frame to the current page. The frame location and size is determined by the current **rectangle**, the line color is determined by the current **color**, the line width is determined by the current **width** and any options are determined by the current **options**.

By specifying values for the horizontal and vertical radius parameters you can draw rectangles with rounded corners. The values refers to the radii of the ellipse used to draw the corners.

---

**Notes**

By setting the horizontal radius parameter to half the width of the rect and the vertical radius parameter to half the height of the rect you can draw ovals and circles.

By default frames are drawn round the outside of the curren rectangle. This allows you to add content and then frame it ensuring that the frame and the content do not overlap. However sometimes you may wish to draw the frame round the inside of the rectangle. You can do this by setting the **inside** parameter to true.

The **FrameRect** function returns the Object ID of the newly added Graphic Object.

The following code adds a black frame to a document. The inset from the edges of the document by 50 points horizontal; 100 points vertically.

**[C#]**

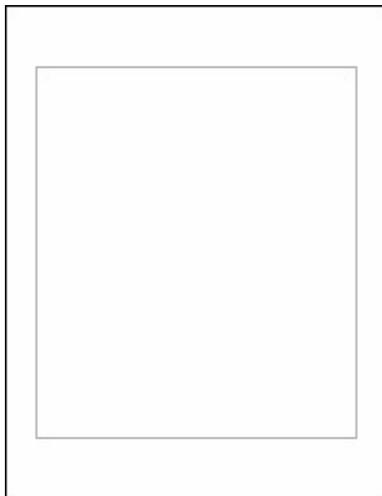
```
Doc theDoc = new Doc();
theDoc.Rect.Inset(50, 100);
theDoc.FrameRect();
```

```
theDoc.Save(Server.MapPath("docframerect.pdf"))
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(50, 100)
theDoc.FrameRect()
theDoc.Save(Server.MapPath("docframerect.pdf"))
theDoc.Clear()
```

**Example**



docframerect.pdf



## GetData Function

---

Saves a document to memory.

### [C#]

```
byte[] GetData()
```

### [Visual Basic]

#### Syntax

```
Function GetData() As Byte()
```

- may throw `Exception()`

#### Params

Name	Description
return	The PDF document as an array of bytes.

Normally, you will want to save your documents using the [Save](#) method. However, sometimes you will need to obtain your PDF as raw data rather than in a file. The `GetData` method allows you to do this.

You may wish to write a PDF directly to a client browser rather than going through an intermediate file. The data you obtain using `GetData` can be written direct to an HTTP stream using `Response.BinaryWrite`. Similarly, you may wish to obtain raw data for insertion into a database.

The [Refactor](#) setting determines whether new/modified redundant objects are eliminated when the document's data is obtained. You can use [SetInfo](#) to change the setting.

**PDFs and Internet Explorer.** The combination of Internet Explorer and Acrobat is not always trouble free - particularly under https or using older versions of Explorer. It can be difficult to know exactly where the problem lies because there is an interaction of the Operating System, Explorer and Acrobat. Any of these can be the cause.

Sometimes, Explorer may get 'stuck' on a particular content type and insist on displaying your PDF as HTML. In this case, you will see random text starting with %PDF. Sometimes, this can happen if you stream PDF data to a window which was previously displaying HTML.

Sometimes, server-side debugging results in extra data being inserted into the content stream. While this may not matter for HTML, it will corrupt binary documents like PDF.

Sometimes, your code may inadvertently insert extra data into the content stream. Again, this will corrupt the PDF. Error messages are a common cause of this kind of corruption.

HTTP compression may result in PDF streams being compressed before return to the browser. While browsers are able to deal with this type of compression, Acrobat may not be able to. You can see if this type of compression is being used by examining the headers returned to the browser using a tool like IEWatch. If you are using IIS 6 compression, you can disable it on a page by page basis using the IIS Metabase Explorer from the IIS 6 resource kit.

Sometimes, the use of HTTPS with PDF content can be problematic. For example, see [Microsoft KB812935](#).

For testing purposes, you may wish to change the content-disposition from 'inline' to 'attachment'. This will allow you to download the data rather than view it in your browser. You can then check the downloaded document using Acrobat or a hex editor.

Alternatively, if the problem is that PDFs seem to be cached you may wish to check the 'Enable Content Expiration' checkbox you will find under the Web Site Properties.

We would suggest two steps:

1. Your first step should be to eliminate ABCpdf as the cause. Why not save the PDF to disk at the same time as sending it to the client? That way you can establish that the PDF is fine. If you want to take it further, you can then read the PDF data from disk and stream it direct to the client.
2. The example site streams PDF data direct to the client. So, install the example site into a new virtual directory and establish if the same issue exists for the example site. If it works, then it's simply a matter of moving your current code base and the example site code base towards each other until you find the cause of the problem.

The following code illustrates how one might add text to a PDF and then write it direct to the client browser. This code is an entire ASP.NET page - hello.aspx.

### **[C#]**

```
<% @Page Language="C#" %>
<% @Import
Namespace="WebSupergoo.ABCpdf10"
%>
<%
Doc theDoc = new Doc();
theDoc.FontSize = 96;
```

```
theDoc.AddText("Hello World");
byte[] theData =
theDoc.GetData();
Response.Clear();
Response.ContentType =
"application/pdf";
Response.AddHeader("content-
disposition", "inline;
filename=MyPDF.PDF");
Response.AddHeader("content-
length",
theData.Length.ToString());
Response.BinaryWrite(theData);
Response.End();
%>
```

### [Visual Basic]

---

#### Example

```
<% @Page Language="Visual Basic"
%>
<% @Import
Namespace="WebSupergoo.ABCpdf10"
%>
<%
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.AddText("Hello World")
Dim theData() As Byte =
theDoc.GetData()
Response.Clear()
Response.ContentType =
"application/pdf"
Response.AddHeader("content-
disposition", "inline;
filename=MyPDF.PDF")
Response.AddHeader("content-
length",
```

```
theData.Length.ToString()  
Response.BinaryWrite(theData)  
Response.End()  
%>
```

Hello World

hello.asp



## GetInfo Function

---

Gets string information about an object.

### [C#]

```
string GetInfo(int id, string  
type)
```

---

### Syntax

### [Visual Basic]

```
Function GetInfo(id As Integer,  
type As String) As String
```

---

### Params

Name	Description
id	The Object ID of the object to be queried.
type	The type of information to be retrieved.
return	The returned information.

After you modify a document you may want to get back information about the objects you have added. For example you might want to find out the natural dimensions of an image or you might

want to find out how many characters were drawn when you inserted some text. This method allows you access to this information.

There are core information types that all objects support. There are also information types specific to particular types of object. You can use core information types to allow you to iterate through every object in your document finding out specific information about each of them.

---

#### Notes

If the object does not exist or does not support the requested type of information then an empty string is returned.

Every object supports the ID and Type properties. For more detailed information about this and other properties see the [Object Paths](#) section of this document.

PDF objects are case sensitive so be sure you use the correct case when specifying information.

The following code snippet illustrates how one might report the natural dimensions of an image.

#### [C#]

```
Doc theDoc = new Doc();
string thePath =
Server.MapPath("../mypics/mypic.jpg");
int theID1 =
theDoc.AddImageFile(thePath, 1);
int theID2 = theDoc.GetInfoInt(theID1,
"XObject");
```

```
string theWidth =
theDoc.GetInfo(theID2, "Width");
string theHeight =
theDoc.GetInfo(theID2, "Height");
Response.Write("Width " + theWidth + "
<br>");
Response.Write("Height " + theHeight +
"<br>");
theDoc.Clear();
```

### [Visual Basic]

#### Example

```
Dim theDoc As Doc = New Doc()
Dim thePath As String =
Server.MapPath("../mypics/mypic.jpg")
Dim theID1 As Integer =
theDoc.AddImageFile(thePath, 1)
Dim theID2 As Integer =
theDoc.GetInfoInt(theID1, "XObject")
Dim theWidth As String =
theDoc.GetInfo(theID2, "Width")
Dim theHeight As String =
theDoc.GetInfo(theID2, "Height")
Response.Write("Width " + theWidth + "
<br>")
Response.Write("Height " + theHeight +
"<br>")
theDoc.Clear()
```

This is the kind of output you might expect.

```
Width 480
Height 640
```





# GetInfoDate Function

---

Gets date information about an object.

## [C#]

```
DateTime GetInfoDate(int id,  
string type)  
DateTime GetInfoDate(int id,  
string type, bool allowLocal)
```

## [Visual Basic]

Syntax

```
Function GetInfoDate(id As  
Integer, type As String) As  
DateTime  
Function GetInfoDate(id As  
Integer, type As String,  
allowLocal As Boolean) As  
DateTime
```

Name	Description
id	The Object ID of the object to be queried.
type	The type of information to be retrieved.

---

**Params**

allowLocal	For dates containing time zone information, if the parameter is true, the returned values will be local times (local to the time zone of the dates, and not to the local machine); if the parameter is false, the returned values will be UTC times. The default is false.
return	The returned value.

This function behaves identically to the [GetInfo](#) method but returns a `DateTime` rather than a string. If the information cannot be obtained or is not a date, then the return value will be the zero `DateTime`.

---

**Notes**

PDF dates also contain times. They are stored as strings in PDF so this function is mostly used with the `:Text` object type.

See the [GetInfo](#) function.

---

**Example**



## GetInfoDouble Function

---

Gets numeric information about an object.

### [C#]

```
double GetInfoDouble(int id,  
string type)
```

---

### Syntax

### [Visual Basic]

```
Function GetInfoDouble(id As  
Integer, type As String) As  
Double
```

---

### Params

Name	Description
id	The Object ID of the object to be queried.
type	The type of information to be retrieved.
return	The returned value.

---

This function behaves identically to the [GetInfo](#) method but returns a double rather than a string. If the information cannot be obtained or

**Notes** is not numeric, the return value will be zero.

See the [GetInfo](#) function.

---

**Example**



## GetInfoInt Function

---

Gets numeric information about an object.

### [C#]

```
int GetInfoInt(int id, string  
type)
```

---

### Syntax

### [Visual Basic]

```
Function GetInfoInt(id As  
Integer, type As String) As  
Integer
```

---

### Params

Name	Description
id	The Object ID of the object to be queried.
type	The type of information to be retrieved.
return	The returned integer.

---

This function behaves identically to the [GetInfo](#) method but returns an integer rather than a string. If the information cannot be obtained or

**Notes** is not numeric, the return value will be zero.

See the [GetInfo](#) function.

---

**Example**



## GetInfoInt64 Function

---

Gets numeric information about an object.

### [C#]

```
long GetInfoInt64(int id, string  
type)
```

---

### Syntax

### [Visual Basic]

```
Function GetInfoInt64(id As  
Integer, type As String) As Long
```

---

### Params

Name	Description
id	The Object ID of the object to be queried.
type	The type of information to be retrieved.
return	The returned integer.

---

### Notes

This function behaves identically to the [GetInfo](#) method but returns a 64-bit integer rather than a string. If the information cannot be obtained or is not numeric, the return value will be zero.

See the [GetInfo](#) function.

---

**Example**



## GetStream Function

---

Gets a document as raw data stream.

### [C#]

```
Stream GetStream()
```

### [Visual Basic]

Syntax

```
Function GetStream() As Stream
```

- may throw `Exception()`

Params

Name	Description
return	The PDF document as a stream.

Normally, you will want to save your documents using the [Save](#) method. However, sometimes you will need to obtain your PDF as raw data rather than in a file. The `GetStream` method allows you to do this.

You may wish to write a PDF directly to a client

browser rather than going through an intermediate file. The data you obtain using `GetStream` can be written direct to an HTTP stream using `Response.BinaryWrite`. Similarly, you may wish to obtain raw data for insertion into a database.

Because of the CLR limit of 2 GB per object, the `GetData` method cannot return the data for a document larger than 2 GB. Use this method for documents larger than 2 GB. Dispose of the returned stream as soon as it is no longer needed for small memory footprint.

The `Refactor` setting determines whether new/modified redundant objects are eliminated when the document's data is obtained. You can use `SetInfo` to change the setting.

**PDFs and Internet Explorer.** The combination of Internet Explorer and Acrobat is not always trouble free - particularly under https or using older versions of Explorer. It can be difficult to know exactly where the problem lies because there is an interaction of the Operating System, Explorer and Acrobat. Any of these can be the cause.

Sometimes, Explorer may get 'stuck' on a particular content type and insist on displaying your PDF as HTML. In this case, you will see random text starting with %PDF. Sometimes, this can happen if you stream PDF data to a window which was previously displaying HTML.

Sometimes, server-side debugging results in extra data being inserted into the content stream. While this may not matter for HTML, it will corrupt binary documents like PDF.

Sometimes, your code may inadvertently insert extra data into the content stream. Again, this will corrupt the PDF. Error messages are a common cause of this kind of corruption.

HTTP compression may result in PDF streams being compressed before return to the browser. While browsers are able to deal with this type of compression, Acrobat may not be able to. You can see if this type of compression is being used by examining the headers returned to the browser using a tool like IEWatch. If you are using IIS 6 compression, you can disable it on a page by page basis using the IIS Metabase Explorer from the IIS 6 resource kit.

Sometimes, the use of HTTPS with PDF content can be problematic. For example, see [Microsoft KB812935](#).

For testing purposes, you may wish to change the content-disposition from 'inline' to 'attachment'. This will allow you to download the data rather than view it in your browser. You can then check the downloaded document using Acrobat or a hex editor.

Alternatively, if the problem is that PDFs seem to be cached you may wish to check

the 'Enable Content Expiration' checkbox you will find under the Web Site Properties.

We would suggest two steps:

1. Your first step should be to eliminate ABCpdf as the cause. Why not save the PDF to disk at the same time as sending it to the client? That way you can establish that the PDF is fine. If you want to take it further, you can then read the PDF data from disk and stream it direct to the client.
2. The example site streams PDF data direct to the client. So, install the example site into a new virtual directory and establish if the same issue exists for the example site. If it works, then it's simply a matter of moving your current code base and the example site code base towards each other until you find the cause of the problem.

The following code illustrates how one might add text to a PDF and then write it direct to the client browser. This code is an entire ASP.NET page - hello.aspx.

**[C#]**

```
<% @Page Language="C#" %>  
<% @Import
```

```

Namespace="WebSupergoo.ABCpdf10"
%>
<%
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.AddText("Hello World");
using (Stream theStream =
theDoc.GetStream()) {
    Response.Clear();
    Response.ContentType =
"application/pdf";
    Response.AddHeader("content-
disposition", "inline;
filename=MyPDF.PDF");
    Response.AddHeader("content-
length",
theStream.Length.ToString());
    long theLen = theStream.Length;
    byte[] theData = new byte[theLen
>= 32768? 32768: (int)theLen];
    while (theLen > 0) {
        theStream.Read(theData, 0,
theData.Length);
        Response.BinaryWrite(theData);
        theLen -= theData.Length;
        if (theLen < theData.Length &&
theLen > 0)
            theData = new
byte[(int)theLen];
    }
}
Response.End();
%>

```

### [Visual Basic]

```

<% @Page Language="Visual Basic"

```

---

**Example**

```
%>
<% @Import
Namespace="WebSupergoo.ABCpdf10"
%>
<%
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.AddText("Hello World")
Using theStream As Stream =
theDoc.GetStream()
    Response.Clear()
    Response.ContentType =
"application/pdf"
    Response.AddHeader("content-
disposition", "inline;
filename=MyPDF.PDF")
    Response.AddHeader("content-
length",
theStream.Length.ToString())
    Dim theLen As Long =
theStream.Length;
    Dim theData() As Byte = New
Byte(Math.Min(32768, theLen))
    Do While theLen > 0
        theStream.Read(theData, 0,
theData.Length)
        Response.BinaryWrite(theData)
        theLen -= theData.Length
        If theLen < theData.Length
AndAlso theLen > 0 Then
            theData = New Byte(theLen)
        End If
    Loop
End Using
Response.End()
%>
```

Hello World

hello.asp

# GetText Function

---



Extract content from the current page in a specified format.

## [C#]

```
string GetText(string type)
string GetText(Page.TextType
type, bool includeAnnotations)
```

## [Visual Basic]

### Syntax

```
Function GetText(type As String)
As String
Function GetText(type As
Page.TextType, includeAnnotations
As Boolean) As String
```

### Params

Name	Description
type	The format in which to return the content.
includeAnnotations	Whether to include field and annotation text.
return	The returned content.

This function allows you to extract the content from a page.

This is a convenience function for easy access to the content of the current page. For full details of how text extraction works see the [Page.GetText](#) function.

The following formats are supported - "Text", "SVG", "SVG+", "SVG+2" and "RawText". These types can be specified as strings for backwards compatibility with older code. However in newer code you should prefer the function overload that takes a [Page.TextType](#) enumeration.

---

#### Notes

Text is in layout order which may not be the same as reading order. ABCpdf will make sensible assumptions on how items of text should be combined but some situations are ambiguous.

The current release of ABCpdf is much more sophisticated than previous ones when it comes to extracting text. However if you are relying on the ABCpdf 8 simplified model you can use the "RawText" format for backwards compatibility.

None.

---

#### Example



# MeasureText Function

---



Measure the length of a block of text without adding it to the page.

## [C#]

```
double MeasureText(string text)
double MeasureText(string text,
double fontSize, double
charSpacing, double wordSpacing,
bool italic, bool bold, double
outline)
double MeasureText(string text,
double fontSize, double
charSpacing, double wordSpacing,
bool italic, bool bold, double
outline, int defaultSize1000ths)
```

## [Visual Basic]

Syntax

```
Function MeasureText(text As
String) As Double
Function MeasureText(text As
String, fontSize As Double,
charSpacing As Double,
wordSpacing As Double, italic As
Boolean, bold As Boolean, outline
As Double) As Double
Function MeasureText(text As
String, fontSize As Double,
charSpacing As Double,
wordSpacing As Double, italic As
Boolean, bold As Boolean, outline
```

As Double, defaultSize1000ths As Integer) As Double

---

Params

Name	Description
text	The text to be measured.
fontSize	The size of the font.
charSpacing	The spacing to be applied between each character.
wordSpacing	The spacing to be applied between each word.
italic	Whether a synthetic italic style is to be applied.
bold	Whether a synthetic bold style is to be applied.
outline	The size of any outline to be applied to the font.
defaultSize1000ths	The default width for missing characters in 1000ths of an M.
return	The width of the text in the units provided.

This function is used to measure the length of a block of text using the current [Font](#).

---

Notes

This method is unit agnostic so you can use whatever units you like and the result will be returned in terms of those units. However typically you would provide point based measurements to provide a point based length.

---

**Example**

None.



# Read Function

---

Reads an existing document.

## [C#]

```
void Read(string path)
void Read(byte[] data)
void Read(Stream stream)
void Read(string path, string password)
void Read(byte[] data, string password)
void Read(Stream stream, string password)
void Read(string path, XReadOptions options)
void Read(byte[] data, XReadOptions options)
void Read(Stream stream, XReadOptions options)
```

## [Visual Basic]

```
Sub Read(path As String)
Sub Read(data() As Byte)
Sub Read(stream As Stream)
Sub Read(path As String, password As String)
Sub Read(data() As Byte, password As String)
Sub Read(stream As Stream, password As String)
Sub Read(path As String, options As
XReadOptions)
Sub Read(data() As Byte, options As
XReadOptions)
Sub Read(stream As Stream, options As
XReadOptions)
```

Syntax

- may throw Exception()

---

**Params**

Name	Description
path	The file path to the PDF, OpenOffice.org, SVG, XPS or other supported document type.
data	The source PDF data.
stream	The source PDF or document stream.
password	Any password needed to open the document.
options	The settings for the read.

Use this method to read a file into a document object. Any existing document content will be discarded. All properties are set back to their defaults.

You can specify a PDF as a file path or by passing in the raw data. Raw data must be held as an array of bytes. You can open encrypted PDF documents if you supply a valid password.

You may notice that colors in the PDF files are slightly different when you are reading non-PDF files. PDF handles alpha blending differently from other file formats. Refer to [SwfImportOperation.Import](#) for notes about alpha blending.



**Other File Types.** This method supports the import of a range of other file types as standard.

For the import of doc and docx formats we recommend use of [WordGlue](#) wherever possible. This eliminates the installation and configuration issues which can be associated with other doc import applications.

If [OpenOffice.org](#) is installed you can pass this method path to OpenOffice.org compatible documents. This means you can read file types like Microsoft Word (.doc, .docx), Microsoft Excel (.xls, .xlsx), Rich Text Format (.rtf), PowerPoint (.ppt, .pptx), WordPerfect (.wpd), Lotus 1-2-3 (.wk1) and AutoCAD (.dxf).

If both Microsoft Office and .NET 3.5 are installed you can pass this method a path to any Microsoft Office compatible document. By default the Microsoft Office import operation works direct but it can also work via the XPS printer driver if you explicitly specify that the XpsAny read module be used. ABCpdf works with Office 2007 or later.

Rich Text Format (.rtf) documents are automatically imported using the nativeABCpdf Rich Text Format reader module. This is generally the simplest, fastest and most reliable import method. However if you have specific needs you can also import them using the OpenOffice.org or Microsoft Office [XReadOptions.ReadModules](#).

You can pass this method a file path to an SVG or EPS document for conversion to PDF. ABCpdf supports a subset of SVG based around the SVG Tiny specification. For details see the [SVG Support](#) section of the documentation. You can also pass a file path to an XPS, OXPS or EPS document for conversion to PDF.

You can use a path to an image type such as JPEG or PNG. ABCpdf will import multi-page images as multi-page documents. The image types supported are, broadly, th

supported by the XImage object.

If you have a stream or data rather than a file path then will need to specify an options parameter. This is necessary because in the case of data ABCpdf does not have a file extension from which it can automatically decide the type module to use. As a result it regards all data as PDF unless told otherwise.

Full control over the import process can be implemented by specifying an options parameter. Reading a document without specifying an options parameter is functionally the same as reading a document using a default XReadOptions object.

After the read operation is complete the [Page](#) property will contain the ID of the first page in the document. The [Rect](#) and [Media](#) properties will reflect the size of the first page in the document.

ABCpdf .NET operates an intelligent just-in-time object loading scheme for PDFs which ensures that only those objects that are required are loaded into memory. This means that if you are modifying large documents then server load will be kept to a minimum. The original PDF document must be available for as long as the [Doc](#) object is being used. As a result you cannot modify or overwrite a PDF file while it is read into a [Doc](#) object. You will need to save your PDF to another location and then delete the two files around after the [Doc](#) object's use of the PDF is ended (with a call to [Clear](#), [Dispose](#), or [Read](#) with another file).

Object deletion requires that all references to an object are removed. There is no way of doing this without checking every object in the document. So object deletion requires that every object in the document is loaded and for large documents this may place a significant load on the server. Reading encrypted documents places a greater load on the server because - lll

object deletion - it requires that every object in the document be loaded.

Please note that you are legally bound to respect the permissions present in existing PDF documents. For details please see [Legal Requirement Section](#).

The Read method may be used to read eForm FDF documents as well as PDF documents. Most PDF operations will not work on FDF documents but you can query field values using the Get methods to return Unicode strings.

**Modifying Documents.** ABCpdf will allow you to open, modify and save PDF documents.

ABCpdf will allow you to draw on top of PDF documents: add or delete pages or modify document data. However because of the way that PDF documents are structured it is unlikely that you'll be able to edit existing content.

So if there are blank spaces which you can draw your entries into that will work. Indeed you might want to draw a white box over existing content and then draw on that.

However you shouldn't expect to be able to edit and re-text that is already in your PDF.

The following illustrates how one might add a large red number to a page of a PDF document.

**[C#]**

```
Doc theDoc = new Doc();
```

```

theDoc.Read(Server.MapPath("../mypics/sampl
theDoc.FontSize = 500;
theDoc.Color.String = "255 0 0";
theDoc.TextStyle.HPos = 0.5;
theDoc.TextStyle.VPos = 0.3;
int theCount = theDoc.PageCount;
for (int i = 1; i <= theCount; i++) {
    theDoc.PageNumber = i;
    theDoc.AddText(i.ToString());
}
theDoc.Save(Server.MapPath("docread.pdf"));
theDoc.Clear();

```

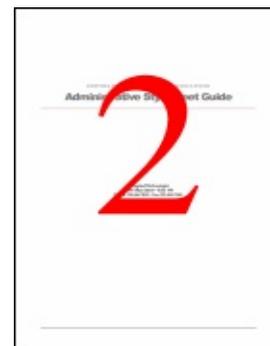
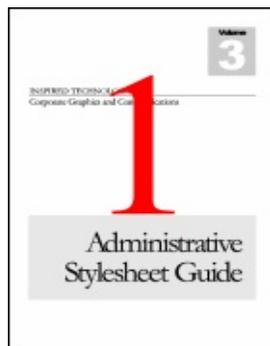
### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sampl
theDoc.FontSize = 500
theDoc.Color.String = "255 0 0"
theDoc.TextStyle.HPos = 0.5
theDoc.TextStyle.VPos = 0.3
Dim theCount As Integer = theDoc.PageCount
For i As Integer = 1 To theCount
    theDoc.PageNumber = i
    theDoc.AddText(i.ToString())
Next
theDoc.Save(Server.MapPath("docread.pdf"))
theDoc.Clear()

```

Example



docsave.pdf - [Page 1]



Table of Contents

Introduction	1
Chapter 1	1
Chapter 2	1
Chapter 3	1
Chapter 4	1
Chapter 5	1
Chapter 6	1
Chapter 7	1
Chapter 8	1
Chapter 9	1
Chapter 10	1
Chapter 11	1
Chapter 12	1
Chapter 13	1
Chapter 14	1
Chapter 15	1
Chapter 16	1
Chapter 17	1
Chapter 18	1
Chapter 19	1
Chapter 20	1
Chapter 21	1
Chapter 22	1
Chapter 23	1
Chapter 24	1
Chapter 25	1
Chapter 26	1
Chapter 27	1
Chapter 28	1
Chapter 29	1
Chapter 30	1
Chapter 31	1
Chapter 32	1
Chapter 33	1
Chapter 34	1
Chapter 35	1
Chapter 36	1
Chapter 37	1
Chapter 38	1
Chapter 39	1
Chapter 40	1
Chapter 41	1
Chapter 42	1
Chapter 43	1
Chapter 44	1
Chapter 45	1
Chapter 46	1
Chapter 47	1
Chapter 48	1
Chapter 49	1
Chapter 50	1
Chapter 51	1
Chapter 52	1
Chapter 53	1
Chapter 54	1
Chapter 55	1
Chapter 56	1
Chapter 57	1
Chapter 58	1
Chapter 59	1
Chapter 60	1
Chapter 61	1
Chapter 62	1
Chapter 63	1
Chapter 64	1
Chapter 65	1
Chapter 66	1
Chapter 67	1
Chapter 68	1
Chapter 69	1
Chapter 70	1
Chapter 71	1
Chapter 72	1
Chapter 73	1
Chapter 74	1
Chapter 75	1
Chapter 76	1
Chapter 77	1
Chapter 78	1
Chapter 79	1
Chapter 80	1
Chapter 81	1
Chapter 82	1
Chapter 83	1
Chapter 84	1
Chapter 85	1
Chapter 86	1
Chapter 87	1
Chapter 88	1
Chapter 89	1
Chapter 90	1
Chapter 91	1
Chapter 92	1
Chapter 93	1
Chapter 94	1
Chapter 95	1
Chapter 96	1
Chapter 97	1
Chapter 98	1
Chapter 99	1
Chapter 100	1

docsave.pdf - [Page 3]

docsave.pdf - [Page 2]



Chapter 1

## Seven Keys to Writing a Professional Essay

Introduction

The purpose of this chapter is to provide you with the tools and techniques you need to write a professional essay. This chapter will cover the following topics:

- Understanding the assignment
- Choosing a topic
- Researching your topic
- Organizing your ideas
- Writing your thesis statement
- Writing your introduction
- Writing your body paragraphs
- Writing your conclusion

Key Takeaways

- Read the assignment carefully and make sure you understand what is being asked of you.
- Choose a topic that interests you and that you are knowledgeable about.
- Do your research and gather evidence to support your thesis statement.
- Organize your ideas into a logical and coherent structure.
- Write a clear and concise thesis statement that states your main argument.
- Write an engaging introduction that draws the reader in and sets the stage for your essay.
- Write body paragraphs that provide evidence and analysis to support your thesis statement.
- Write a strong conclusion that summarizes your main points and leaves a lasting impression on the reader.

docsave.pdf - [Page 4]



# RemapPages Method

---

Remaps pages for reordering, copying and deletion.

## [C#]

```
void RemapPages(string pages)
void RemapPages(int[] pages)
void RemapPages(int[] pages, int index, int
count)
```

---

## Syntax

## [Visual Basic]

```
Sub RemapPages(pages As String)
Sub RemapPages(pages() As Integer)
Sub RemapPages(pages() As Integer, index As
Integer, count As Integer)
```

---

## Params

Name	Description
pages	The list of page numbers.
index	The index of the first page number into the array pa
count	The number of page numbers in the array pages to

This method provides a simple method for remapping the pages in a document. It can be used for reordering, copying or deleting pages.

It accepts a list of page numbers and reorders the pages in a document so that they match these page numbers. If a number is repeated more than once, the page is duplicated. If a number is omitted, the page is deleted.

---

#### Notes

Page numbers can be delimited using spaces, commas or semicolons. The first page in a document is page one. So to reverse a four page document, you might use "4 3 2 1".

If a page is duplicated and it contains [Form](#) fields, you may want to call [MakeFieldsUnique](#) to avoid sharing fields across pages.

The following code snippet illustrates how one might reverse pages in a document.

#### [C#]

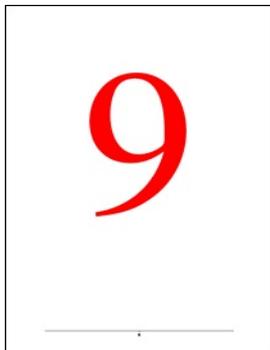
```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.doc"));
theDoc.FontSize = 500;
theDoc.Color.String = "255 0 0";
theDoc.TextStyle.HPos = 0.5;
theDoc.TextStyle.VPos = 0.3;
int theCount = theDoc.PageCount;
string thePages = "";
for (int i = 1; i <= theCount; i++) {
    theDoc.PageNumber = i;
    theDoc.AddText(i.ToString());
    thePages = thePages + (theCount - i + 1).ToString() + " ";
}
theDoc.RemapPages(thePages);
```

```
theDoc.Save(Server.MapPath("docremappages.p  
theDoc.Clear());
```

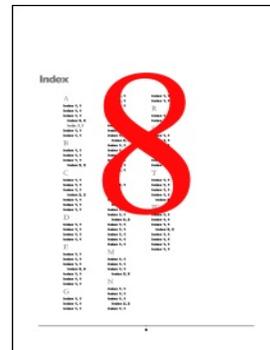
### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../mypics/sampl  
theDoc.FontSize = 500  
theDoc.Color.String = "255 0 0"  
theDoc.TextStyle.HPos = 0.5  
theDoc.TextStyle.VPos = 0.3  
Dim theCount As Integer = theDoc.PageCount  
Dim thePages As String = ""  
For i As Integer = 1 To theCount  
theDoc.PageNumber = i  
theDoc.AddText(i.ToString())  
thePages = thePages + (theCount - i +  
1).ToString() + " "  
Next  
theDoc.RemapPages(thePages)  
theDoc.Save(Server.MapPath("docremappages.p  
theDoc.Clear()
```

Example



docremappages.pdf [Page 1]



docremappages.pdf [P



docremappages.pdf [Page 3]



docremappages.pdf [P



## Save Function

---

Saves the document as PDF.

### [C#]

```
void Save(string path)
void Save(Stream stream)
```

### [Visual Basic]

Syntax

```
Sub Save(path As String)
Sub Save(stream As Stream)
```

- may throw Exception()

Params

Name	Description
path	The destination file path.
stream	The destination stream.

Use this method to export the current document as PDF, XPS, PostScript, HTML, DOCX or SWF.

To export as XPS, PostScript, DOCX or HTML you need to specify a file path with an appropriate extension - ".xps", ".ps", ".docx", ".htm", ".html" or

".swf". If the file extension is unrecognized then the default PDF format will be used.

When saving to a Stream the format can be indicated using a [Doc.SaveOptions.FileExtension](#) property such as ".htm" or ".xps". For HTML you must provide a sensible value for the [Doc.SaveOptions.Folder](#) property. For XPS streams must be both readable and writable - `FileAccess.ReadWrite` and not simply `FileAccess.Write`.

ABCpdf operates an intelligent just-in-time object loading scheme which ensures that only those objects that are required are loaded into memory. This means that if you are modifying large documents then server load will be kept to a minimum. The original PDF document must be available for as long as the [Doc](#) object is being used.

As a result you cannot modify or overwrite a PDF file while it is read into a [Doc](#) object. You will need to save your PDF to another location and then swap the two files around after the [Doc](#) object's use of the PDF is ended (with a call to [Clear](#), `Dispose`, or [Read](#) with another PDF file).

If you need to obtain a PDF as raw data you can use the [GetData](#) function.

The [Refactor](#) setting determines whether new/modified redundant objects are eliminated when the document is saved. You can use [SetInfo](#) to change the setting.

**Versions.** ABCpdf automatically determines the version depending on the features you use. If you use features from only the 1.1 specification

it will write a 1.1 PDF. If you use 1.3 features it will write a 1.3 PDF. If you use 1.4 features it will write a 1.4 PDF. Ditto 1.5 and 1.6.

If you're using a PDF template or drawing from another PDF the final output will be the minimum version used in these templates. In many real world applications this will be the factor which determines the version in the final output produced by ABCpdf.

There is no advantage in producing a 1.6 document if you're not using features from the 1.6 feature set. To do this will simply stop users of older versions of Acrobat from accessing a document which should be available to them.

When saving to SWF, if the `Doc.SaveOptions.Template` is null, the current page is exported with `Rect` as the bounds of the Flash movie using `Doc.SaveOptions.TemplateData.MeasureDpiX` and `Doc.SaveOptions.TemplateData.MeasureDpiY` if specified. Otherwise, `Doc.SaveOptions.Template` specifies the path to a SWF file. The saved SWF file starts with the template SWF files, and a frame is added for each page in the document. The script added is in ActionScript 2. If the template's version is Flash Player 7 or lower, the saved file's version will be Flash Player 8. For information on the interaction between the added frames and the script from the template, please refer to the example Flash file. Images are output in JPEG if (1) they are in DeviceGray or DeviceRGB and already in JPEG

(without any other compression on top), or (2) they are not in the indexed color space and both the width and the height are at least 8 pixels. For (1), the original JPEG data is used so you can control the quality by pre-compressing the images; for (2), the output will use 80% quality.

The following code illustrates how one might add text to a PDF and then save it out.

### **[C#]**

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.AddText("Hello World");
theDoc.Save(Server.MapPath("docsave.pdf"));
theDoc.Clear();
```

### **[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.AddText("Hello World")
theDoc.Save(Server.MapPath("docsave.pdf"))
theDoc.Clear()
```

---

**Example**

Hello World

docsave.pdf



# SetInfo Function

---

Sets information about an object.

## [C#]

```
void SetInfo(int id, string type, string info)
void SetInfo(int id, string type, int info)
void SetInfo(int id, string type, double info)
void SetInfo(int id, string type, DateTime info)
```

## [Visual Basic]

Syntax

```
Sub SetInfo(id As Integer, type As String, info As String)
Sub SetInfo(id As Integer, type As String, info As Integer)
Sub SetInfo(id As Integer, type As String, info As Double)
Sub SetInfo(id As Integer, type As String, info As DateTime)
```

Name	Description
id	The Object ID of the object to be modified.
type	The type of value to insert.
	The value to insert.

---

## Params

info

- The overloads taking integer/floating-point info convert this parameter to the string representation (number used in PDF) in the invariant culture without creating a managed string object.
- The overload taking DateTime info converts this parameter to the string representation (PDF date string) so it is mostly used with the `:Text` object type.

In the same way as you can get information about aspects of a document using the `GetInfo` method you can modify aspects of the document using the `SetInfo` method.

Different types of object support different types of properties; for more detailed information see the [Object Paths](#) section of this document.

---

## Notes

PDF objects are case sensitive so be sure you use the correct case.

See [Doc Properties Example](#) for an example of using the overload taking DateTime info.

The following shows how to modify the document catalog so that the PDF opens onto the second page rather than the first.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.pdf"));
int thePages = theDoc.GetInfoInt(theDoc.Root, "Pages");
int thePage2 = theDoc.GetInfoInt(thePages, "2");
```

```
string theAction = "[ " + thePage2.ToString  
R /Fit ]";  
theDoc.SetInfo(theDoc.Root, "/OpenAction",  
theAction);  
theDoc.Save(Server.MapPath("docsetinfo.pdf")  
theDoc.Clear());
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../mypics/sampl  
Dim thePages As Integer =  
theDoc.GetInfoInt(theDoc.Root, "Pages")  
Dim thePage2 As Integer =  
theDoc.GetInfoInt(thePages, "Page 2")  
Dim theAction As String = "[ " + thePage2.T  
+ " 0 R /Fit ]"  
theDoc.SetInfo(theDoc.Root, "/OpenAction",  
theAction)  
theDoc.Save(Server.MapPath("docsetinfo.pdf")  
theDoc.Clear()
```

---

#### Example

**Open Actions.** The way in which a PDF is displayed w is dependent on certain flags within the PDF itself. Here common combinations. For full details of how these wo should see the Adobe PDF Specification available from web site.

To show outlines:

```
theDoc.SetInfo(theDoc.Root, "/PageMode",  
"/UseOutlines")
```

Or for thumbnails:

```
theDoc.SetInfo theDoc.Root, "/PageMode",  
"/UseThumbs "
```

To display two pages side by side:

```
theDoc.SetInfo(theDoc.Root, "/PageLayout  
"/TwoColumnLeft")
```

To make the print dialog appear when the document is opened:

```
theDoc.SetInfo(theDoc.Root, "/OpenAction  
/Type /Action /S /Named /N /Print >>")
```

To open at 200% zoom onto the current page:

```
theDoc.SetInfo(theDoc.Root, "/OpenAction  
+ theDoc.Page.ToString() + " 0 R /XYZ nul  
2 ]")
```

To fit the document width onto the current page:

```
theDoc.SetInfo(theDoc.Root, "/OpenAction  
+ theDoc.Page.ToString() + " 0 R /FitH "  
theDoc.MediaBox.Height.ToString() + " ]")
```



# ToString Function

---

A string representation of the graphic style of the document

## [C#]

```
override string ToString()
```

---

## Syntax

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

## Params

Name	Description
none	The string representation of the object.

---

## Notes

This method returns the string value of the object. This is equivalent to reading the String property of the object.

---

## Example

None.



# Bookmark Property

---



Type	Default Value	Read Only	Description
<b>[C#]</b> <code>Objects.Bookmark</code>	n/a	No	The top level bookmark for the document.
<b>[Visual Basic]</b> <code>Objects.Bookmark</code>			

The top level bookmark associated with this document.

PDF documents typically provide a list of bookmarks for easy navigation between pages. In Acrobat this navigation structure is available under the Bookmarks tab. The PDF specification refers to this structure as the document outline.

---

## Notes

The document outline comprises of a hierarchy of bookmarks. The bookmark at the top of the hierarchy is available via this property.

See the [Objects.Bookmark](#) object for further details.

None.

---

## Example



# Color Property

---



Type	Default	Read Only	Description
[C#] XColor			The current drawing and filling color.
[Visual Basic] XColor	Black	No	

The color used for drawing.

---

## Notes

This property determines the color used for drawing lines, shapes, filled shapes and text.

The following code creates a PDF document with a red background.

### [C#]

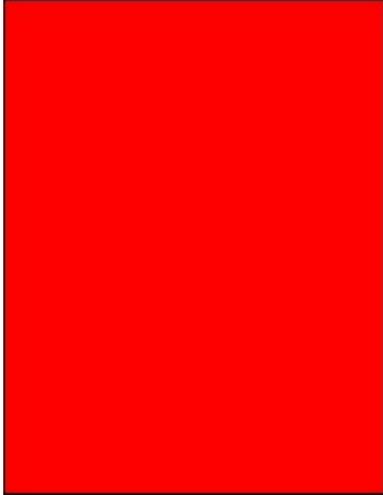
```
Doc theDoc = new Doc();
theDoc.Color.String = "255 0 0";
theDoc.FillRect();
theDoc.Save(Server.MapPath("doccolor.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Color.String = "255 0 0"
```

Example

```
theDoc.FillRect()  
theDoc.Save(Server.MapPath("doccolor.pdf"))  
theDoc.Clear()
```



doccolor.pdf

# ColorSpace Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>	0	No	The current ColorSpace ID.
[Visual Basic] <code>Integer</code>			

The current color space.

The ColorSpace is used when the [Color](#) is of a matching type. If color type does not match then the default - device - color space used.

---

## Notes

For example you add a CMYK color space and assign it to the ColorSpace property. All CMYK Colors you use will be defined in terms of this color space. However RGB and Grayscale colors will continue to be defined in terms of the default - device - color space.

To get a ColorSpace ID you need to add your color space to the current document using the [AddColorSpaceFile](#) or [AddColorSpaceSpot](#) method.

The following code shows how to colorize an image. It adds a color space to the current page and converts it to grayscale. Then it creates a new image in the color space and assigns the new color space to the image.

[C#]

---

```

Doc theDoc = new Doc();
theDoc.Rect.Inset(20, 20);

XImage theImg = new XImage();
theImg.SetFile(Server.MapPath("../mypics/mypic.
int theID = theDoc.AddImageObject(theImg, false

theID = theDoc.GetInfoInt(theID, "XObject");
theDoc.SetInfo(theID, "Grayscale", "");

int theCS = theDoc.AddColorSpaceSpot("MAGENTA",
0 0");
theDoc.SetInfo(theID, "/ColorSpace:Ref",
theCS.ToString());
theDoc.SetInfo(theID, "/Decode", "[1 0]");

theDoc.Save(Server.MapPath("doccolorspace.pdf"));
theDoc.Clear();

```

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(20, 20)

Dim theImg As XImage = New XImage()
theImg.SetFile(Server.MapPath("../mypics/mypic.
Dim theID As Integer = theDoc.AddImageObject(th
False)

theID = theDoc.GetInfoInt(theID, "XObject")
theDoc.SetInfo(theID, "Grayscale", "")

Dim theCS As Integer =
theDoc.AddColorSpaceSpot("MAGENTA", "0 100 0 0"
theDoc.SetInfo(theID, "/ColorSpace:Ref",
theCS.ToString())
theDoc.SetInfo(theID, "/Decode", "[1 0]")

```

**Example**

```
theDoc.Save(Server.MapPath("doccolorspace.pdf"))  
theDoc.Clear()
```



doccolorspace.pdf

# CropBox Property

---



Type	Default	Read Only	Description
[C#] XRect [Visual Basic] XRect	Letter Size [0, 0, 612, 792].	Yes	The current document visible page size.

The current document visible page size.

This property reflects the CropBox of the current page.

---

## Notes

For methods you can use to change the CropBox of the current page see the [MediaBox](#) property.

None.

---

## Example

# Encryption Property

---



Type	Default	Read Only	Description
[C#] XEncryption	No encryption.	No	The current encryption settings.
[Visual Basic] XEncryption			

This property determines the current encryption settings. When you save the document the encryption settings - if any - will be used to determine the method of encryption to be used.

By default documents are not encrypted when created. However if you read an existing PDF document the encryption property will be updated to reflect the encryption settings used to create the original document.

PDF encryption supports two different types of password - a user password and an owner password. You can use either password to open and view the document. However unless you supply the owner password, permissions will be applied and access to the document may be restricted.

---

## Notes

Typically PDF encryption is used to apply permissions to a document. The user password is left blank and only an owner password supplied. PDF readers will not prompt for a password if there is no user password. However any permissions will automatically be applied. In this way you

can allow anyone to open the document but restrict access to operations like copying text, printing the document or extracting images.

Please note that you are legally bound to respect the permissions present in existing PDF documents. For details please see the [Legal Requirement Section](#).

The following code saves a simple PDF document using a 128 bit encryption key. It applies a copy-protection permission to stop people copying text out of the document.

#### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.AddText("Hello World!");
theDoc.Encryption.Type = 2;
theDoc.Encryption.CanCopy = false;
theDoc.Encryption.OwnerPassword = "owner";
theDoc.Save(Server.MapPath("docencrypt.pdf"));
```

---

Example

#### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.AddText("Hello World!")
theDoc.Encryption.Type = 2
theDoc.Encryption.CanCopy = false
theDoc.Encryption.OwnerPassword = "owner"
theDoc.Save(Server.MapPath("docencrypt.pdf"))
```

# Font Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The current Font ID.

The font used for drawing text.

This property holds the current Font ID and determines the style of text that is added to the document using methods like [AddText](#).

---

## Notes

To get a Font ID you need to add your font to the current document using the [AddFont](#) method.

The Font property is an accessor for the the [XTextStyle.Font](#) property.

The following example adds two blocks of styled text to a document. The first block is in Helvetica and the second in Courier.

## [C#]

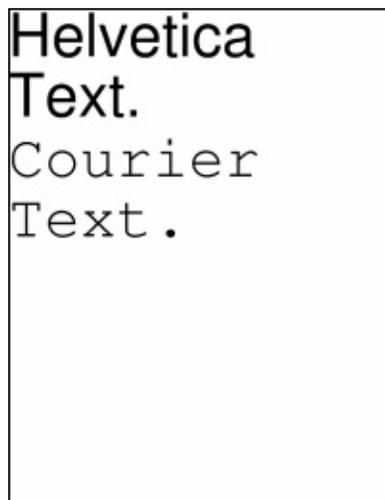
```
Doc theDoc = new Doc();  
theDoc.FontSize = 96; // big text
```

```
theDoc.Font = theDoc.AddFont("Helvetica");  
theDoc.AddText("Helvetica Text.");  
theDoc.Font = theDoc.AddFont("Courier");  
theDoc.AddText("Courier Text.");  
theDoc.Save(Server.MapPath("docfont.pdf"));  
theDoc.Clear();
```

### [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()  
theDoc.FontSize = 96 ' big text  
theDoc.Font = theDoc.AddFont("Helvetica")  
theDoc.AddText("Helvetica Text.")  
theDoc.Font = theDoc.AddFont("Courier")  
theDoc.AddText("Courier Text.")  
theDoc.Save(Server.MapPath("docfont.pdf"))  
theDoc.Clear()
```



Helvetica  
Text.  
Courier  
Text.

docfont.pdf

# FontSize Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	10	No	The current text size.

The line height for drawing text.

This property determines the size of text that is added to the document using methods like [AddText](#).

The font size is measured in the current [Units](#).

---

## Notes

You should prefer use of the [XTextStyle.Size](#) property, to which the `FontSize` property is simply an integer accessor. Assigning a value to the `FontSize` is exactly equivalent to assigning it to the [XTextStyle.Size](#). Getting a value from this property is exactly equivalent to the following.

```
int n = (int)(doc.TextStyle.Size + 0.5);
```

The following example adds two blocks of styled text to a document. The first block is in 96 point type and the second is in 192 point type.

## [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.AddText("Small ");
theDoc.FontSize = 192;
theDoc.AddText("Big");
theDoc.Save(Server.MapPath("docfontsize.pdf"));
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.AddText("Small ")
theDoc.FontSize = 192
theDoc.AddText("Big")
theDoc.Save(Server.MapPath("docfontsize.pdf"))
theDoc.Clear()
```

Example



docfontsize.pdf

# Form Property

---



Type	Default Value	Read Only	Description
[C#] XForm			
[Visual Basic] XForm	n/a	No	The document form and fields.

This property allows you to examine and manipulate the document Form and Fields.

Fields are generally obtained using a fully qualified name. A full name describes a path down through the field hierarchy - using periods as delimiters - to a specific field object.

---

## Notes

Once a field has been obtained you can query or change its values. If you wish to convert the fields into a permanent part of the document you can use the [Field.Stamp](#) method to permanently emboss them.

See the XForm object for full details.

---

## Example

None.



# HtmlOptions Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">XHtmlOptions</a>	n/a	No	The HTML and URL rendering options.
[Visual Basic] <a href="#">XHtmlOptions</a>			

This property allows you control over the way that HTML is rendered.

The properties of this object may be used to control the way that the [AddImageUrl](#) and [AddImageHtml](#) methods work.

---

## Notes

The methods of this object operate on objects added using the theAddImageUrl and AddImageHtml methods. Some operations change the content. Others provide information about the content which has been added.

See the [XHtmlOptions](#) object for further details.

---

## Example

None.



# Layer Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>	1	No	The insertion layer for new content.
[Visual Basic] <code>Integer</code>			

The insertion layer for new content.

The default layer is one which indicates insertion at the top. So if you want to insert at the bottom and your page has ten layers you need to insert at position eleven.

Note that insertion may not behave predictably with documents read from file. It is good practice for each layer to operate independently but not all documents are set up this way.

Optional Content Groups: The type of layer that this property describes is an ABCpdf construct that you cannot detect using Acrobat. Acrobat layers are something completely different and are more precisely known as Optional Content Groups (OCGs).

See the `Doc.Layer` property for details of how to construct this type of layer.

**Optional Content Groups.** The type of layer that this property describes is an ABCpdf construct that you cannot detect using Acrobat. Acrobat layers are something completely different and are more precisely known as Optional Content Groups (OCG items).

OCGs allow you to specify content that may be visible or invisible. As a user you can see the names of the OCGs that exist and you can turn them on or off to see or hide the respective content.

It is a mistake to think of OCGs as pure layers, as the rules associated with them can be quite sophisticated. Each content item on the page may be associated with one or more nested visibility groups and it is only if all these groups are visible that the content item is visible. While visibility groups are often OCGs they can also be Optional Content Membership Dictionaries (OCMD items) - a construct that determines visibility from a set of OCGs using a set of custom written rules. So visibility can be complex and items with visibility may be interleaved rather than conceptually part of a simple contiguous layer.

That said, it is quite common for simple OCG setups to mimic simple contiguous layers as this is what most people require.

The OCGLayers example project which comes with ABCpdf shows how to create content with visibility determined using OCGs. It also includes more complex examples detailing the use of nested OCGs and OCMDs. As well as

creating layers it also shows shows how to turn OCGs on and off, annotate the items on a page based on the OCGs that they belong to and how to redact and delete invisible items, removing any associated OCGs.

---

Example

None.

# LayerCount Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The number of layers on the current page.

This property holds the number of layers on the current page.

---

## Notes

You can add layers to pages using calls to methods such as [AddText](#) and [AddImage](#). See the [Layer](#) property for more details.

---

## Example

None.

# MediaBox Property

---



Type	Default	Read Only	Description
[C#] XRect	Letter Size	No	The current document page size.
[Visual Basic] XRect	[0, 0, 612, 792].		

This property reflects the size of the current page. It also determines the size of new pages created by the [AddPage](#) method. Note that changing the MediaBox does not change the current [Rect](#). So typically you'll want to write code like this:

## [C#]

```
theDoc.MediaBox.String = "0 0 200 300";  
theDoc.Rect.String =  
theDoc.MediaBox.String;
```

## [Visual Basic]

```
theDoc.MediaBox.String = "0 0 200 300"  
theDoc.Rect.String =  
theDoc.MediaBox.String
```

Changing this property will change the size of pages created by subsequent calls to `AddPage`. However it will not change the size of the pages that have already been created. To change the size of the pages that have already

been created you need to use the [SetInfo](#) method. For example:

### [C#]

Notes

```
theDoc.SetInfo(theDoc.Page,  
"/MediaBox:Rect", "0 0 200 300");
```

### [Visual Basic]

```
theDoc.SetInfo(theDoc.Page,  
"/MediaBox:Rect", "0 0 200 300")
```

Similar methods can be used to control other page size measures such as the CropBox, BleedBox, TrimBox and ArtBox. For example:

### [C#]

```
theDoc.SetInfo(theDoc.Page,  
"/CropBox:Rect", "20 20 180 280");
```

### [Visual Basic]

```
theDoc.SetInfo(theDoc.Page,  
"/CropBox:Rect", "20 20 180 280")
```

The default page size is often the one you'll want to use. However it may be that your PDFs are required to conform to a different page size. If this is the case you can simply specify the name of the size rather than the exact dimension. See the [Rect.String](#) property for details.

The following code creates a PDF document containing three different pages each with a different size.

### [C#]

---

```
Doc theDoc = new Doc();
theDoc.Page = theDoc.AddPage();
theDoc.MediaBox.String = "A4";
theDoc.Page = theDoc.AddPage();
theDoc.MediaBox.String = "B5";
theDoc.Page = theDoc.AddPage();
theDoc.Save(Server.MapPath("docmediabox.pdf"));
theDoc.Clear();
```

---

Example

### **[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
theDoc.Page = theDoc.AddPage()
theDoc.MediaBox.String = "A4"
theDoc.Page = theDoc.AddPage()
theDoc.MediaBox.String = "B5"
theDoc.Page = theDoc.AddPage()
theDoc.Save(Server.MapPath("docmediabox.pdf"))
theDoc.Clear()
```

# ObjectSoup Property

---



Type	Default	Read Only	Description
[C#] ObjectSoup	n/a	Yes	The collection of objects that make up the PDF.
[Visual Basic] ObjectSoup			

This property holds a reference to the ObjectSoup for the document. The soup is a non-traditional collection of indirect PDF objects which comprise the content of the PDF.

---

## Notes

Do not alter the soup or the contents of the soup unless you are sure of the changes you are making. If inappropriate changes are made the result will be a corrupt PDF output.

---

## Example

None.

# Options Property

---



Type	Default	Read Only	Description
[C#] <code>string</code>			The state options for low level drawing control.
[Visual Basic] <code>String</code>	""	No	

The options property is an advanced feature which allows low level control over PDF drawing. It is important to use it correctly as incorrect use can corrupt your documents.

The value of the options property is inserted into the PDF content stream after state has been established but before any drawing has taken place. You can use this property to insert your own state commands. This is most commonly used for drawing dashed lines.

---

## Notes

[FillRect](#), [FrameRect](#), [AddLine](#) and [AddArc](#) all insert the options parameter.

You can find details of graphics state parameters and how to use them in the [Adobe PDF Specification](#).

The following code adds an arc to a document. It uses the options parameter to make the line dashed rather than solid.

[C#]

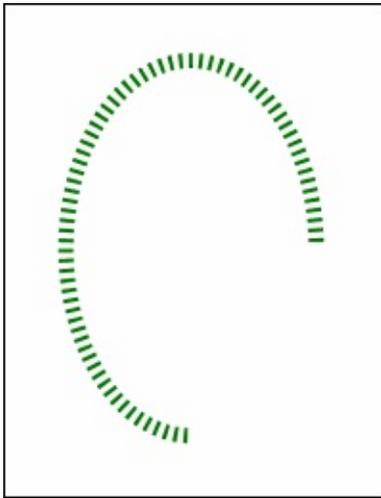
---

```
Doc theDoc = new Doc();
theDoc.Width = 24;
theDoc.Color.String = "0 120 0";
theDoc.Options = "[6 10] 6 d";
theDoc.AddArc(0, 270, 300, 400, 200, 300);
theDoc.Save(Server.MapPath("docoptions.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Width = 24
theDoc.Color.String = "0 120 0"
theDoc.Options = "[6 10] 6 d"
theDoc.AddArc(0, 270, 300, 400, 200, 300)
theDoc.Save(Server.MapPath("docoptions.pdf"))
theDoc.Clear()
```

Example



docoptions.pdf

# Page Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	0	No	The current Page ID.

This property holds the current Page ID. The current page is the one that receives new objects as they are added to the document.

For example the methods [AddText](#), [AddLine](#), [AddImage](#), [FrameRect](#) and [FillRect](#) all operate on the current page.

When you change the Page property the [Pos](#) property is reset to the top left of the current [Rect](#).

---

## Notes

Note that the [PageNumber](#) property is not the same as the Page property. The Page holds an Object ID typically returned from a call to [AddPage](#). The PageNumber indicates the page using an index ranging between one and the [PageCount](#).

Either the Page or the PageNumber property can be used to set the current page.

If no page is specified the current page is taken to be the first page in the document.

The following example creates a document with two pages and adds text to each of the pages in turn.

### [C#]

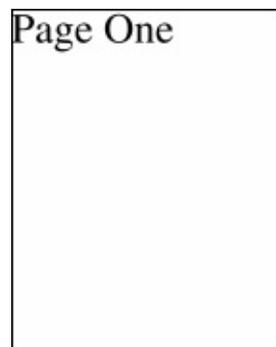
```
Doc theDoc = new Doc();
theDoc.FontSize = 96; // big text
theDoc.Page = theDoc.AddPage();
theDoc.AddText("Page One");
theDoc.Page = theDoc.AddPage();
theDoc.AddText("Page Two");
theDoc.Save(Server.MapPath("docpage.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96 ' big text
theDoc.Page = theDoc.AddPage()
theDoc.AddText("Page One")
theDoc.Page = theDoc.AddPage()
theDoc.AddText("Page Two")
theDoc.Save(Server.MapPath("docpage.pdf"))
theDoc.Clear()
```

---

Example



Page Two

docpage.pdf

# PageCount Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			The number of pages in the document.
[Visual Basic] <code>Integer</code>	0	Yes	

This property holds the number of pages in the document.

---

## Notes

You can add pages to the document using the [AddPage](#) method.

---

## Example

None.

# PageNumber Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The page number of the current page.

This property holds the current Page Number. The current page is the one that receives new objects as they are added to the document.

For example the methods [AddText](#), [AddLine](#), [AddImage](#), [FrameRect](#) and [FillRect](#) all operate on the current page.

When you change this property the [Pos](#) property is reset to the top left of the current [Rect](#).

Note that the PageNumber property is not the same as the [Page](#) property. The Page holds an Object ID typically returned from a call to [AddPage](#). The PageNumber indicates the page using an index ranging between one and the [PageCount](#).

Either the Page or the PageNumber property can be used to set the current page.

If no page is specified the current page is taken to be the first page in the document.

---

## Optimization Tips.

Well structured PDF documents are optimized for fast access to pages. So when you go to a particular page of a document by specifying a PageNumber this operation is quick.

However it is less obvious that getting the PageNumber property can be more expensive than one might think. ABCpdf cannot assume that the PDF is the same as it was last time you accessed the PageNumber so it cannot necessarily just give you the number it had last time. It has to assume the worst case which is that you have reorganized the pages in some way and at the very least it will have to validate the current PageNumber by a test retrieval of that page. At the very worst it will have to scan through the entire page tree looking to see where the current page has ended up.

In general getting the PageNumber is not too expensive. But if you write code which results in it being accessed thousands of times then it can become noticeable. So if you find yourself doing this then consider using the [Page.GetPageArrayAll](#) function instead. It's an easy optimization to make.



# Pos Property

---



Type	Default	Read Only	Description
[C#]XPoint	The top left of the current rectangle.	No	The current drawing position.
[Visual Basic]XPoint			

This property determines the current drawing position. This is used by the `AddText` method.

When you change the `Page` or `Rect` properties the `Pos` is automatically reset to the top left of the current `Rect`. After adding text using the `AddText` method the `Pos` is updated to point to the next text insertion position - just after the last drawn character.

When adding text with `AddText` or `AddHtml`, `Pos` specifies the top left corner of characters, and text is positioned in rows going from left to right and ordered from top to bottom. However, when `Font` is set to a vertical font (a font in the vertical writing mode/`FontObject.WritingMode`), text is positioned in columns going from top to bottom and ordered from right to left as in East Asian scripts. The value of `Pos` is not changed by a change in `Font`. This works because with vertical text flow, an internal transformation (which is only a manifestation of the text processing) is applied to `Pos` (only `Pos` and nothing else), and all horizontal positioning settings

are applied to vertical coordinates and vice versa. (Note that the effects of [XTextStyle.CharSpacing](#) and [XTextStyle.WordSpacing](#) depend on the writing mode.) The transformation rotates the coordinate system by 90 degrees clock-wise and maps the top-left corner of the [Rect](#) to the top-right corner of the [Rect](#). Hence, the default initial value of Pos specifies vertical text to start at the top-right corner of the [Rect](#). This is similar to Windows processing of text using an East Asian font whose name is prefixed with @ and applying the transformation only at the last step.

Please note that text in vertical fonts and text in horizontal fonts do not mix well without additional positioning so it is not recommended to use horizontal fonts with [AddHtml](#) when [Font](#) is set to a vertical font or vice versa.

The following code creates a PDF document with text positioned at a number of different points within it.

#### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 48;
for (int i = 1; i <= 8; i++) {
    theDoc.Pos.X = i * 40;
    theDoc.Pos.Y = i * 80;
    theDoc.AddText("Pos = " +
theDoc.Pos.String);
}
theDoc.Save(Server.MapPath("docpos.pdf"));
theDoc.Clear();
```

#### [Visual Basic]

---

Example

```
Dim theDoc As Doc = New Doc()  
theDoc.FontSize = 48  
For i As Integer = 1 To 8  
    theDoc.Pos.X = i * 40  
    theDoc.Pos.Y = i * 80  
    theDoc.AddText("Pos = " +  
theDoc.Pos.String)  
Next  
theDoc.Save(Server.MapPath("docpos.pdf"))  
theDoc.Clear()
```

```
Pos = 320 640  
Pos = 280 560  
Pos = 240 480  
Pos = 200 400  
Pos = 160 320  
Pos = 120 240  
Pos = 80 160  
Pos = 40 80
```

docpos.pdf

# Rect Property

---



Type	Default	Read Only	Description
[C#] <a href="#">XRect</a> [Visual Basic] <a href="#">XRect</a>	The dimensions of the current page.	No	The current rectangle used for drawing operations.

This property determines the current rectangle. This is used by a number of operations including [AddText](#), [AddImage](#), [FrameRect](#) and [FillRect](#).

The [XRect](#) object represents a rectangular area in two-dimensional space. The properties of the [XRect](#) object represent the bottom left and top right corners of the area.

[AddText](#) adds text within the current rectangle wrapping the text at the edges.

---

## Notes

The [AddImage](#) methods add an image scaled to fill the current rectangle.

[FrameRect](#) frames the current rectangle and [FillRect](#) fills the current rectangle.

When you change this property the [Pos](#) property is reset to point to the top left of the [Rect](#).

The following code creates a PDF document containing a number of concentric frames.

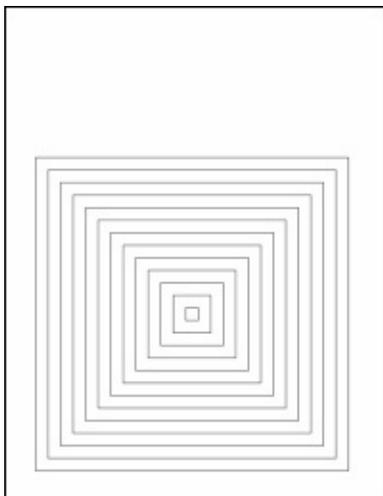
### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.String = "50 50 550 550";
for (int i = 1; i <= 20; i++) {
    theDoc.FrameRect();
    theDoc.Rect.Inset(20, 20);
}
theDoc.Save(Server.MapPath("docrect.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.String = "50 50 550 550"
For i As Integer = 1 To 20
    theDoc.FrameRect()
    theDoc.Rect.Inset(20, 20)
Next
theDoc.Save(Server.MapPath("docrect.pdf"))
theDoc.Clear()
```

**Example**



docrect.pdf

# Rendering Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">XRendering</a>	n/a	No	The rendering options and control.
[Visual Basic] <a href="#">XRendering</a>			

This property allows you control over the way that PDFs are rendered.

---

## Notes

Note that Rendering is only available under the ABCpdf Professional License.

See the [XRendering](#) object for further details.

---

## Example

None.

# Root Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>	0	Yes	The root catalog object.
[Visual Basic] <code>Integer</code>			

This property holds the ID of the catalog object.

The catalog is the root of the whole PDF document. It contains information on the root Pages object and the Outline object.

---

## Notes

For dynamically created documents the root of the document is always one. However documents read from existing PDF files may use different root IDs.

The following code snippet illustrates how one might find some information about a PDF document.

## [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/mydoc.pdf"));
string theVers, theNames, thePages, theOutlines;
theVers = theDoc.GetInfo(theDoc.Root, "Version");
theNames = theDoc.GetInfo(theDoc.Root, "/Names");
thePages = theDoc.GetInfo(theDoc.Root, "pages");
```

```
theOutlines = theDoc.GetInfo(theDoc.Root,
"outlines");
Response.Write("Version " + theVers + "<br>");
Response.Write("Names " + theNames + "<br>");
Response.Write("Pages ID " + thePages + "<br>");
Response.Write("Outlines ID " + theOutlines + '
<br>");
theDoc.Clear();
```

### [Visual Basic]

#### Example

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/mydoc.pdf")
Dim theVers As String,theNames As String,thePages
As String,theOutlines As String
theVers = theDoc.GetInfo(theDoc.Root, "Version")
theNames = theDoc.GetInfo(theDoc.Root, "/Names")
thePages = theDoc.GetInfo(theDoc.Root, "pages")
theOutlines = theDoc.GetInfo(theDoc.Root,
"outlines")
Response.Write("Version " + theVers + "<br>")
Response.Write("Names " + theNames + "<br>")
Response.Write("Pages ID " + thePages + "<br>")
Response.Write("Outlines ID " + theOutlines + '
<br>")
theDoc.Clear()
```

This might result in the following output.

```
Version /1.4
Names 12 0 R
Pages ID 618
Outlines ID 2169
```

# SaveOptions Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">XSaveOptions</a>	n/a	No	The save options and control.
[Visual Basic] <a href="#">XSaveOptions</a>			

This property allows you control over the way that PDF documents are saved or streamed.

---

## Notes

The properties of this object may be used to control the way that the [Save](#) and [GetData](#) methods work.

See the [XSaveOptions](#) object for further details.

---

## Example

None.

# String Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			A string representation of the graphic style of the document
[Visual Basic] <code>String</code>	Variable	No	

A string representation of the graphic style of the document.

---

## Notes

This covers the [Transform](#), [ColorSpace](#), [Color](#), [Font](#), [TextStyle](#), [Width](#) and [Options](#) properties. However it does not cover the [Page](#), [Layer](#), [Rect](#) or [Pos](#) properties.

In this example we show how to use the `String` property to implement a graphics state stack with `Push` and `Pop` operators.

### [C#]

```
Doc doc = new Doc();
doc.FontSize = 64;
doc.Rect.Inset(20, 20);
doc.Font = doc.AddFont("Helvetica");
Stack<string> state = new
```

```

Stack<string>();
state.Push(doc.String);
doc.AddText("Black
Helvetica\r\n\r\n");
doc.Color.SetRgb(255, 0, 0);
doc.Font = doc.AddFont("Helvetica-
Oblique");
doc.AddText("Red Helvetica-
Oblique\r\n\r\n");
doc.String = state.Pop();
doc.AddText("Black Helvetica
again\r\n\r\n");
doc.Save("savestate.pdf");

```

### [Visual Basic]

```

Dim doc As New Doc()
doc.FontSize = 64
doc.Rect.Inset(20, 20)
doc.Font = doc.AddFont("Helvetica")
Dim state As New Stack(Of String)()
state.Push(doc.[String])
doc.AddText("Black Helvetica" & vbCr
& vbLf & vbCr & vbLf)
doc.Color.SetRgb(255, 0, 0)
doc.Font = doc.AddFont("Helvetica-
Oblique")
doc.AddText("Red Helvetica-Oblique" &
vbCr & vbLf & vbCr & vbLf)
doc.[String] = state.Pop()
doc.AddText("Black Helvetica again" &
vbCr & vbLf & vbCr & vbLf)
doc.Save("savestate.pdf")

```

---

Example

Black Helvetica

*Red Helvetica-  
Oblique*

Black Helvetica  
again

savestate.pdf

# TextStyle Property

---



Type	Default	Read Only	Description
[C#] XTextStyle	Ten point text.	No	The current style for drawing text.
[Visual Basic] XTextStyle			

This property determines the current style settings used for adding text.

---

## Notes

The following code creates a PDF document and adds some text using a number of the text style properties to control formatting.

### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in
partes tres, quarum unam incolunt Belgae, alian
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes lingua,
institutis, legibus inter se differunt. Gallos
ab Aquitanis Garumna flumen, a Belgis Matrona e
Sequana dividit.";
theText = theText + "\r\n" + theText + "\r\n" +
theText + "\r\n";
```

```
theDoc.Rect.Inset(20, 20);
theDoc.TextStyle.Size = 32;
theDoc.TextStyle.Justification = 1;
theDoc.TextStyle.Indent = 64;
theDoc.TextStyle.ParaSpacing = 32;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("doctextstyle.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String = "Gallia est omnis divis
in partes tres, quarum unam incolunt Belgae,
aliam Aquitani, tertiam qui ipsorum lingua
Celtae, nostra Galli appellantur. Hi omnes
lingua, institutis, legibus inter se differunt.
Gallos ab Aquitanis Garumna flumen, a Belgis
Matrona et Sequana dividit."
theText = theText + vbCrLf + theText + vbCrLf +
theText + vbCrLf
theDoc.Rect.Inset(20, 20)
theDoc.TextStyle.Size = 32
theDoc.TextStyle.Justification = 1
theDoc.TextStyle.Indent = 64
theDoc.TextStyle.ParaSpacing = 32
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("doctextstyle.pdf"));
theDoc.Clear()
```

Example

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, institutis, legibus inter se differunt. Gallos ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, institutis, legibus inter se differunt. Gallos ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, institutis, legibus inter se differunt. Gallos ab Aquitanis Garumna flumen, a Belgis

doctextstyle.pdf

# TopDown Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	The current position of the origin.

PDF coordinates are measured upwards from the bottom of the document. For some types of layout it can be useful to measure coordinates downwards from the top of the document.

By setting this property to true coordinates are assumed to start at the top rather than the bottom of the document.

More precisely the origin is assumed to be at the top-left of the current [MediaBox](#).

---

## Notes

There are a [variety of methods](#) you can use to change coordinate systems.

Note that transforms operate on the underlying PDF coordinate space rather than any abstraction specified by the [Units](#) and [TopDown](#) properties. If you are using transforms you will find it easiest to work in the native PDF coordinate space.

The following code creates a PDF document and adds a grid measured in inches.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Units = UnitType.Inches;
theDoc.TopDown = true;
theDoc.Width = 1.0 / 8.0;
theDoc.FontSize = 1;
theDoc.Rect.Pin = XRect.Corner.TopLeft;
for (int i = 0; i <= 12; i += 2) {
    theDoc.AddLine(0, i, 12, i);
    theDoc.Rect.Position(0, i);
    theDoc.AddText(i.ToString());
    theDoc.AddLine(i, 0, i, 12);
    theDoc.Rect.Position(i, 0);
    theDoc.AddText(i.ToString());
}
theDoc.Save(Server.MapPath("doctopdown.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Units = UnitType.Inches
theDoc.TopDown = True
theDoc.Width = 1.0 / 8.0
theDoc.FontSize = 1
theDoc.Rect.Pin = XRect.Corner.TopLeft
For i As Integer = 0 To 12 Step 2
    theDoc.AddLine(0, i, 12, i)
    theDoc.Rect.Position(0, i)
    theDoc.AddText(i.ToString())
    theDoc.AddLine(i, 0, i, 12)
    theDoc.Rect.Position(i, 0)
    theDoc.AddText(i.ToString())
Next
```

**Example**

```
theDoc.Save(Server.MapPath("doctopdown.pdf"))  
theDoc.Clear()
```

0	2	4	6	8
2				
4				
6				
8				
10				

doctopdown.pdf

# Transform Property

---



Type	Default	Read Only	Description
[C#] XTransform	No transformation.	No	The current transformation for drawing.
[Visual Basic] XTransform			

This property determines the current world space transform. It affects any drawing using the [AddText](#), [AddImage](#), [AddLine](#), [FrameRect](#) and [FillRect](#) methods.

Transforms are general operations which encompass rotation, translation, magnification and skewing or a combination of these. Note that the order in which transforms are applied is significant: a rotation followed by a translation is not the same as a translation followed by a rotation.

---

## Notes

A world space transform is not same an object transform. You are changing the coordinate system - not the objects you're inserting.

Note that transforms operate on the underlying PDF coordinate space rather than any abstraction specified by the [Units](#) and [TopDown](#) properties. If you are using transforms you will find it easiest to work in the native PDF coordinate space.

The following code creates a PDF document and adds some text and a rectangle rotated at 45 degrees anti-clockwise around the middle of the document.

### [C#]

```
Doc theDoc = new Doc();
string theText;
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes lingua,
instituti, legibus inter se differunt. Gallos
ab Aquitanis Garumna flumen, a Belgis Matrona et
Sequana dividit.";
theText = theText + "\r\n" + theText + "\r\n" +
theText + "\r\n";
theDoc.Rect.Magnify(0.5, 0.5);
theDoc.Rect.Position(151, 198);
theDoc.FrameRect();
theDoc.Transform.Rotate(45, 302, 396);
theDoc.FrameRect();
theDoc.FontSize = 24;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("doctransform.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes lingua,
instituti, legibus inter se differunt. Gallos
```

---

Example

ab Aquitanis Garumna flumen, a Belgis Matrona et Sequana dividit."

```
theText = theText + vbCrLf + theText + vbCrLf +  
theText + vbCrLf
```

```
theDoc.Rect.Magnify(0.5, 0.5)
```

```
theDoc.Rect.Position(151, 198)
```

```
theDoc.FrameRect()
```

```
theDoc.Transform.Rotate(45, 302, 396)
```

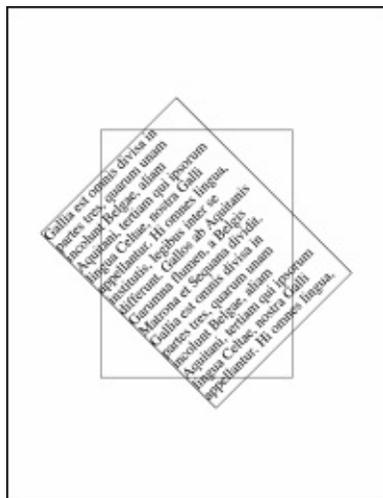
```
theDoc.FrameRect()
```

```
theDoc.FontSize = 24
```

```
theDoc.AddText(theText)
```

```
theDoc.Save(Server.MapPath("doctransform.pdf"))
```

```
theDoc.Clear()
```



doctransform.pdf

# Units Property

---



Type	Default	Read Only	Description
[C#] <code>UnitType</code>			The current measurement units.
[Visual Basic] <code>UnitType</code>	<code>UnitType.Points</code>	No	

This property holds the current measurement units.

The `UnitType` enumeration may take the following values:

- Points (PostScript Points) - 1/72 of an Inch
- Twips (Twentieths of a Point) - 1/20 of a PostScript Point
- Didots (Didot Points) - 1/72 of a French Royal Inch
- ATAPoints (ATA Points) - 1/72.272 of an Inch
- TeXPoints (TeX Points) - 1/72.27 of an Inch
- INPoints (l'Imprimerie Nationale Points) - 0.4 mm
- Picas - 12 PostScript Points
- Ciceros - 12 Didot Points
- ATACiceros - 12 ATA Points
- TeXCiceros - 12 TeX Points
- Microns - millionths of a metre
- Mm - thousandths of a metre
- Cm - hundredths of a metre
- M - metres
- Inches
- Feet

There are a [variety of methods](#) you can use to change

coordinate systems.

Notes

### Why are my Units a string?

In older versions of ABCpdf the Units property was a string. So you might find code of this form.

```
theDoc.Units = "mm"
```

In Version 8 the Units property has been changed to a true enumeration. This is a safer way of coding as it allows the compiler to ensure that the values you are using are valid. Your new code should look like this.

```
theDoc.Units = UnitType.Mm
```

The names of the items in the UnitType enumeration are the same as the values of the strings used in previous versions. So changing your code should be a simple search and replace operation.

Alternatively if you need to convert between enumerations and strings automatically you can do so. To convert from a string to an enumeration use the following code.

```
UnitType unitType =  
(UnitType)Enum.Parse(typeof(UnitType),  
unitString, true)
```

To convert from an enumeration to a string use the following code.

```
string unitString =  
unitType.ToString("G")
```



---

**Example**

See the [TopDown](#) property.

# Width Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	1.0	No	The current line width.

The width determines the width of lines drawn using methods like [AddLine](#) and [FrameRect](#). The width is measured in the current [Units](#).

The [AddLine](#) method creates lines centered on the points you provide. This means if you draw a horizontal line with a width of ten the line will extend five points above your start position and five points below it.

---

## Notes

The [FrameRect](#) method draws lines outside the rectangle you provide. So if you frame a rectangle using a width of ten, the drawn rectangle will extend ten points above, below, to the left and to the right of your specified rectangle. No drawing will fall within the rectangle.

The following code adds two lines to a document. The first line has a width of ten points and the second has a width of twenty points.

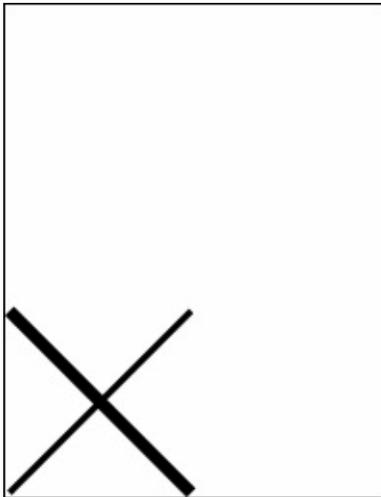
## [C#]

```
Doc theDoc = new Doc();  
theDoc.Width = 10;  
theDoc.AddLine(10, 10, 300, 300);  
theDoc.Width = 20;  
theDoc.AddLine(10, 300, 300, 10);  
theDoc.Save(Server.MapPath("docwidth.pdf"));  
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Width = 10  
theDoc.AddLine(10, 10, 300, 300)  
theDoc.Width = 20  
theDoc.AddLine(10, 300, 300, 10)  
theDoc.Save(Server.MapPath("docwidth.pdf"))  
theDoc.Clear()
```

Example



docwidth.pdf



# FromGray Function

---

Create an XColor from a grayscale value.

## [C#]

```
static XColor FromGray(int gray)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromGray(gray As Integer) As XColor
```

---

## Params

Name	Description
gray	The amount of black (0 to 255)
return	The resulting XColor.

Create an XColor from a grayscale value ranging between zero and 255.

---

## Notes

The value represents the amount of black ink so zero indicates white and 255 indicates black.

None.

---

## Example



# FromRgb Function

---



Create an XColor from a set of RGB component values.

## [C#]

```
static XColor FromRgb(int red,  
int green, int blue)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromRgb(red As  
Integer, green As Integer, blue  
As Integer) As XColor
```

---

## Params

Name	Description
red	The amount of red (0 to 255)
green	The amount of green (0 to 255)
blue	The amount of blue (0 to 255)
return	The resulting XColor.

---

## Notes

Create an XColor from a set of RGB component values ranging between zero and 255..

---

Example

None.

# FromCmyk Function

---



Create an XColor given a set of CMYK component values.

## [C#]

```
static XColor FromCmyk(int cyan,  
int magenta, int yellow, int  
black)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromCmyk(cyan As  
Integer, magenta As Integer,  
yellow As Integer, black As  
Integer) As XColor
```

---

## Params

Name	Description
cyan	The amount of cyan (0 to 100)
magenta	The amount of magenta (0 to 100)
yellow	The amount of yellow (0 to 100)
black	The amount of black (0 to 100)
return	The resulting XColor.

Create an XColor given a set of CMYK component values ranging between zero and

---

Notes 100.

---

Example None.

# FromComponents Function

---



Create an XColor from a set of PDF components in the generic ColorSpace color space.

## [C#]

```
static XColor  
FromComponents(double value1)  
static XColor  
FromComponents(double value1,  
double value2)  
static XColor  
FromComponents(double value1,  
double value2, double value3)  
static XColor  
FromComponents(double value1,  
double value2, double value3,  
double value4)
```

## [Visual Basic]

Syntax

```
Shared Function  
FromComponents(value1 As Double)  
As XColor  
Shared Function  
FromComponents(value1 As Double,  
value2 As Double) As XColor  
Shared Function  
FromComponents(value1 As Double,  
value2 As Double, value3 As  
Double) As XColor  
Shared Function
```

```
FromComponents(value1 As Double,  
value2 As Double, value3 As  
Double, value4 As Double) As  
XColor
```

---

**Params**

Name	Description
value1	The intensity of the first component (typically 0 to 1)
value2	The intensity of the second component (typically 0 to 1)
value3	The intensity of the third component (typically 0 to 1)
value4	The intensity of the fourth component (typically 0 to 1)
return	The resulting XColor.

Create an XColor from a set of PDF components in the generic [ColorSpace](#) color space.

---

**Notes**

PDF color components typically range between zero - no intensity - and one - 100% intensity. However this is not always the case. For color spaces such as Lab the components may take a wider range of values.

None.

---

## Example

# FromArrayAtom Function

---



Create an XColor from an ArrayAtom of NumAtoms containing PDF color values.

## [C#]

```
static XColor  
FromArrayAtom(ArrayAtom array)
```

## [Visual Basic]

Syntax

```
Shared Function  
FromArrayAtom(array As ArrayAtom)  
As XColor
```

- may throw Exception()

Params

Name	Description
array	The ArrayAtom containing the components of the color.
return	The resulting XColor.

Create an XColor from an [ArrayAtom](#) of [NumAtoms](#) containing PDF color values.

There may be only one, three or four items in

the ArrayAtom. The number of items is used to select between Grayscale, RGB or CMYK color spaces respectively.

---

**Notes**

The values expected are PDF color values so all the atoms in the ArrayAtom must be NumAtoms with a value between zero and one, each representing a component of the color.

If these conditions are not met then an exception will be raised.

None.

---

**Example**

# FromOperator Function

---



Create an XColor given a PDF color operator and a set of Atoms containing the arguments for that operator.

## [C#]

```
static XColor FromOperator(Atom[] arguments, string op)
```

## [Visual Basic]

Syntax

```
Shared Function  
FromOperator(arguments() As Atom,  
op As String) As XColor
```

- may throw Exception()

Params

Name	Description
arguments	The Atoms containing the parameters for the color operator.
op	The PDF color operator.
return	The resulting XColor.

Create an XColor given a PDF color operator and a set of [Atoms](#) containing the arguments for

that operator.

There are a variety of color operators available detailed in the PDF Specification. However the most common ones are "rg" and "RG" which are used to set the color to DeviceRGB. These operators take three NumAtom arguments for the red, green and blue components, each of which is a value ranging between zero and one. The upper case operator is used to set the stroking color and the lower case one is used to set the non-stroking color.

---

**Notes**

Other similar operators include "g" and "G" for grayscale; "k" and "K" for CMYK; "sc" and "SC" for generic color components; "scn" and "SCN" for generic components - possibly also including a pattern name.

If these conditions are not met then an exception will be raised.

None.

---

**Example**



# SetColor Function

---

Sets the color.

## [C#]

```
void SetColor(XColor color)
```

---

### Syntax

## [Visual Basic]

```
Sub SetColor(color As XColor)
```

---

### Params

Name	Description
color	The source color.

---

### Notes

This method copies the value from the parameter.

---

### Example

None.



# SetGray Function

---

Set the color to a grayscale value.

## [C#]

```
void SetGray(int gray)
void SetGray(int gray, int alpha)
```

---

## Syntax

## [Visual Basic]

```
Sub SetGray(gray As Integer)
Sub SetGray(gray As Integer,
alpha as Integer)
```

---

## Params

Name	Description
gray	The amount of black (0 to 255).
alpha	The level of opacity from transparent through to fully opaque (0 to 255).

Set the color to grayscale and provide a value for it.

---

## Notes

Optionally set the alpha value to specify a transparency level. If this parameter is omitted the color is set to fully opaque - no transparency.

---

**Example**

None.



# SetRgb Function

---

Set the color to an RGB value.

## [C#]

```
void SetRgb(int red, int green,  
int blue)  
void SetRgb(int red, int green,  
int blue, int alpha)
```

## [Visual Basic]

### Syntax

```
Sub SetRgb(red As Integer, green  
As Integer, blue As Integer)  
Sub SetRgb(red As Integer, green  
As Integer, blue As Integer,  
alpha as Integer)
```

### Params

Name	Description
red	The amount of red (0 to 255).
green	The amount of green (0 to 255).
blue	The amount of blue (0 to 255).
alpha	The level of opacity from transparent through to fully opaque (0 to 255).

Set the color to RGB and provide a value for it.

---

**Notes**

Optionally set the alpha value to specify a transparency level. If this parameter is omitted the color is set to fully opaque - no transparency.

---

**Example**

None.

# Equals Function

---



Test whether the two colors are effectively the same

## [C#]

```
bool Equals(XColor color)
override bool Equals(object
color)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(color As XColor)
As Boolean
Overrides Function Equals(other
As Object) As Boolean
```

---

## Params

Name	Description
color	The color to test against.
return	Whether the two colors are the same.

Test whether the two colors are effectively the same.

Colors are considered equal if they have the same [ColorSpace](#), the same number and values of color [Components](#) and the same

**Name.** This represents value equality for the colors in question.

---

**Notes**

The underlying components of a color are represented as floating point numbers. Floating point numbers are subject to rounding errors, so there has to be a degree of latitude when comparing color values. The degree of latitude is, by default, determined by the color space. CMYK values use a percentage based scale so the value used to determine the resolution of the comparison is 1%. Other color spaces use an eight bit scale so the resolution of comparison is 1/256.

---

**Example**

None.



# GetHashCode Function

---

A hash code for the XColor

## [C#]

```
override int GetHashCode()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function GetHashCode()  
As Integer
```

---

### Params

Name	Description
return	The returned hash code.

---

### Notes

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

---

### Example

None.



# SetCmyk Function

---

Set the color to an CMYK value.

## [C#]

```
void SetCmyk(int cyan, int  
magenta, int yellow, int black)  
void SetCmyk(int cyan, int  
magenta, int yellow, int black,  
int alpha)
```

## [Visual Basic]

### Syntax

```
Sub SetCmyk(cyan As Integer,  
magenta As Integer, yellow As  
Integer, black As Integer)  
Sub SetCmyk(cyan As Integer,  
magenta As Integer, yellow As  
Integer, black As Integer, alpha  
as Integer)
```

### Params

Name	Description
cyan	The amount of cyan (0 to 100).
magenta	The amount of magenta (0 to 100).
yellow	The amount of yellow (0 to 100).
black	The amount of black (0 to 100).
alpha	The level of opacity from transparent through to fully opaque (0 to 255).

Set the color to CMYK and provide a value for it.

---

**Notes**

Optionally set the alpha value to specify a transparency level. If this parameter is omitted the color is set to fully opaque - no transparency.

---

**Example**

None.

# SetComponents Function

---



Set the color to a set of ColorSpace PDF components

## [C#]

```
void SetComponents(double value1)
void SetComponents(double value1,
double value2)
void SetComponents(double value1,
double value2, double value3)
void SetComponents(double value1,
double value2, double value3,
double value4)
```

## [Visual Basic]

Syntax

```
Sub SetComponents(value1 As
Double)
Sub SetComponents(value1 As
Double, value2 As Double)
Sub SetComponents(value1 As
Double, value2 As Double, value3
As Double)
Sub SetComponents(value1 As
Double, value2 As Double, value3
As Double, value4 As Double)
```

Name	Description

---

**Params**

value1	The intensity of the first component (typically 0 to 1)
value2	The intensity of the second component (typically 0 to 1)
value3	The intensity of the third component (typically 0 to 1)
value4	The intensity of the fourth component (typically 0 to 1)

Sets the color to the generic [ColorSpace](#) color space and provide a set of components for it.

---

**Notes**

PDF color components typically range between zero - no intensity - and one - 100% intensity. However this is not always the case. For color spaces such as Lab the components may take a wider range of values.

---

**Example**

None.



## SetRandom Function

---

Set the color to a random opaque value in the current color space.

### [C#]

```
void SetRandom()
```

---

### Syntax

### [Visual Basic]

```
Sub SetRandom()
```

---

### Params

Name	Description
None	

---

### Notes

Set the color to a random opaque value in the current color space.

---

### Example

None.



# ToArrayAtom Function

---



An ArrayAtom representation of the components of the color

## [C#]

```
ArrayAtom ToArrayAtom()
```

---

### Syntax

## [Visual Basic]

```
ArrayAtom ToArrayAtom()
```

---

### Params

Name	Description
return	An ArrayAtom representing the color.

An ArrayAtom representation of the components of the color.

---

### Notes

The ArrayAtom that is created will contain one NumAtom for each component of the color, each with the appropriate value set.

None.

---

### Example





# ToString Function

---

Returns a string representation of the object.

## [C#]

```
override string ToString()
```

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Syntax

Name	Description
return	The string representation of the object.

---

### Params

This method returns the string value of the object. This is equivalent to reading the [String](#) property of the object.

---

### Notes

None.

---

### Example



# Alpha Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	255	No	The alpha opacity.

Allows you to get or set the alpha opacity of the color.

---

## Notes

Alpha values can range from 0 to 255. Zero indicates fully transparent and 255 indicates fully opaque.

Here we create a PDF document showing how different values of alpha result in different levels of transparency.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(20,20);
theDoc.FontSize = 300;
for (int i = 1; i <= 10; i++) {
    theDoc.Color.Alpha = 255 / i;
    theDoc.AddText(theDoc.Color.Alpha.ToString());
    theDoc.Rect.Move(25, -50);
}
theDoc.Save(Server.MapPath("coloralpha.pdf"));
theDoc.Clear();
```

## [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()  
theDoc.Rect.Inset(20,20)  
theDoc.FontSize = 300  
For i As Integer = 1 To 10  
    theDoc.Color.Alpha = 255 / i  
    theDoc.AddText(theDoc.Color.Alpha.ToString())  
    theDoc.Rect.Move(25, -50)  
Next  
theDoc.Save(Server.MapPath("coloralpha.pdf"))  
theDoc.Clear()
```



coloralpha.pdf

# Black Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The black component.

Allows you to get or set the black level.

CMYK color components can range from 0 to 100.

Querying this property does not change the [ColorSpace](#). This means you can obtain approximate CMYK values for RGB or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to CMYK.

None.

---

## Example

# Blue Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The blue component.

Allows you to get or set the blue component.

RGB color components can range from 0 to 255.

Querying this property does not change the `ColorSpace`. This means you can obtain approximate RGB values for CMYK or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to RGB.

None.

---

## Example

# Color Property

---



Type	Default	Read Only	Description
[C#] Color			
[Visual Basic] Color	n/a	No	The System.Drawing.Color.

---

## Notes

Allows you to access the color as a standard .NET Color from the System.Drawing namespace.

---

## Example

None.

# ColorSpace Property

---



Type	Default	Read Only	Description
[C#] ColorOperatorType	ColorOperatorType.Rgb	No	The color space for the color.
[Visual Basic] ColorOperatorType			

Allows you to get or set the current color space for the color.

Grayscale, RGB and CMYK colors are defined purely in terms of the values of the XColor object. However more complex colors need to be defined in the context of a separate color space.

The ColorOperatorType enumeration may take the following values:

- None
- DeviceGray (i.e. grayscale)
- DeviceRGB (i.e. RGB)
- DeviceCMYK (i.e. CMYK)
- ColorSpace (i.e. a generic set of color components)

The actual color is defined in terms of the [Components](#) of the color. For grayscale, RGB and CMYK there are one, three and four of these respectively. These components are most commonly accessed using properties such as [Gray](#), [Red](#), [Green](#) or [Blue](#) which express the values in terms of integers

(typically 0 to 255). However it is ultimately the **Components** ranging from 0.0 to 1.0 which are provided as parameters to the PDF color operator.

If you change the value of this property between DeviceGray, DeviceRGB and DeviceCMYK, the current color will automatically be converted to the indicated color space. However different color spaces have fundamentally different properties and color conversions can only ever be approximate. Changing to None or ColorSpace will not change the number or values of the **Components**.

The integer values for the enumeration values DeviceGray, DeviceRGB and DeviceCMYK are one, three and four respectively, so that they match up with the number of color components in each of these color spaces.

The ColorSpace enumeration value indicates a generic color with a generic set of components. This type of color only makes sense within the context of a specified color space such as a spot color space. In this situation the parameters that are provided to the PDF color operator are accessed directly using the **Components** of the color. In addition, for pattern color spaces, there may also be a **Name** associated with the color.

---

Notes

### **Why is my ColorSpace an integer?**

In older versions of ABCpdf the ColorSpace property was an integer. So you might find code of this form.

```
theColor.ColorSpace = 3
```

In Version 10 the ColorSpace property has been changed to a true enumeration. This is a safer way of coding as it allows the compiler to ensure that the values you are using are valid. Your new code should look like this.

```
theColor.ColorSpace =  
ColorOperatorType.DeviceRGB
```

The old values of the integers match up with the new enumeration values. So if you find code of this form:

```
theColor.ColorSpace = anInteger
```

You can just cast it to the correct type.

```
theColor.ColorSpace =  
(ColorOperatorType)anInteger
```

Or similarly:

```
int n = theColor.ColorSpace
```

You can again cast it to the correct type.

```
int n = (int)theColor.ColorSpace
```

None.

---

Example

# Components Property

---



Type	Default Value	Read Only	Description
[C#] IList<double>	n/a	No	The components of the color in native PDF format
[Visual Basic] IList<double>			

The components of the color in native PDF format.

PDF color components typically range between zero - no intensity - and one - 100% intensity. However this is not always the case. For color spaces such as Lab the components may take a wider range of values.

---

## Notes

In the following example we demonstrate how to use generic color components to draw in the Lab color space.

### [C#]

```
using (Doc doc = new Doc()) {  
    doc.Width = 80;  
    doc.Rect.Inset(50, 50);  
    ColorSpace cs = new  
    ColorSpace(doc.ObjectSoup,
```

```

ColorSpaceType.Lab);
    doc.ColorSpace = cs.ID;
    // This Lab color is a deep green
    doc.Color.ColorSpace =
ColorOperatorType.ColorSpace;
    doc.Color.Components[0] = 50; // L
range is 0 to +100
    doc.Color.Components[1] = -50; // a
range is -100 to +100
    doc.Color.Components[2] = +50; // B
range is -100 to +100
    doc.AddOval(true);
    doc.Save("examplelabcolorspace.pdf");
}

```

### [Visual Basic]

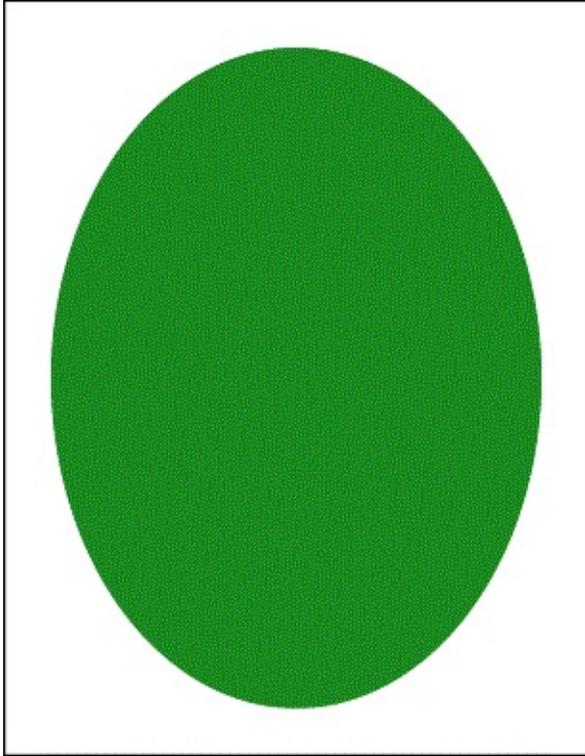
```

Using doc As New Doc()
    doc.Width = 80
    doc.Rect.Inset(50, 50)
    Dim cs As New
ColorSpace(doc.ObjectSoup,
ColorSpaceType.Lab)
    doc.ColorSpace = cs.ID
    ' This Lab color is a deep green
    doc.Color.ColorSpace =
ColorOperatorType.ColorSpace
    doc.Color.Components(0) = 50
    ' L range is 0 to +100
    doc.Color.Components(1) = -50
    ' a range is -100 to +100
    doc.Color.Components(2) = +50
    ' B range is -100 to +100
    doc.AddOval(True)
    doc.Save("examplelabcolorspace.pdf")
End Using
End Sub

```

---

Example



examplelabcolorspace.pdf

# Cyan Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	0	No	The cyan component.

Allows you to get or set the cyan level.

CMYK color components can range from 0 to 100.

Querying this property does not change the [ColorSpace](#). This means you can obtain approximate CMYK values for RGB or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to CMYK.

None.

---

## Example

# Gray Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The gray component.

Allows you to get or set the gray level.

Grayscale levels can range from 0 (black) to 255 (white).

Querying this property does not change the [ColorSpace](#). This means you can obtain approximate Grayscale values for RGB or CMYK colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to Grayscale.

This property can also be used to specify levels for spot colorants. Levels can range from 0 (0% intensity) to 255 (100% intensity). See the [AddColorSpaceSpot](#) method for an example.

None.

---

## Example



# Green Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The green component.

Allows you to get or set the green component.

RGB color components can range from 0 to 255.

Querying this property does not change the `ColorSpace`. This means you can obtain approximate RGB values for CMYK or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to RGB.

None.

---

## Example

# Magenta Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The magenta component.

Allows you to get or set the magenta level.

CMYK color components can range from 0 to 100.

Querying this property does not change the `ColorSpace`. This means you can obtain approximate CMYK values for RGB or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to CMYK.

None.

---

## Example

# Name Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	null	No	Any name that may be associated with this color

---

## Notes

Any name that may be associated with this color. Most commonly this is used for pattern names.

---

## Example

None.

# Red Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The red component.

Allows you to get or set the red component.

RGB color components can range from 0 to 255.

Querying this property does not change the `ColorSpace`. This means you can obtain approximate RGB values for CMYK or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to RGB.

None.

---

## Example

# String Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	"0 0 0"	No	The color as a string.

Allows you access to the color as a string.

If the color is in the RGB color space then the string contains three values representing the Red, Green and Blue levels. For example "100 150 200".

If the color is in the CMYK color space then the string contains four values representing the Cyan, Magenta, Yellow and Black levels. For example "30 60 90 10".

---

## Notes

If the color is in the Grayscale color space then the string contains one value representing the Gray level. For example "150".

You can use the ColorSpace property to find the current color space for the color.

Alpha values can be indicated by prepending an 'a' to an extra component. For example "30 60 90 a120" would indicate an RGB value with an alpha value of 120.

---

**Example**

None.

# Yellow Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The yellow component.

Allows you to get or set the yellow level.

CMYK color components can range from 0 to 100.

Querying this property does not change the `ColorSpace`. This means you can obtain approximate CMYK values for RGB or Grayscale colors.

---

## Notes

However if you change the value of this property the color will automatically be converted to CMYK.

None.

---

## Example



# SetCryptMethods Function

---

Sets the crypt methods for encryption levels of [type](#) 4 or ab

## [C#]

```
void SetCryptMethods(CryptMethodType method  
void SetCryptMethods(CryptMethodType string  
CryptMethodType streamMethod)
```

## [Visual Basic]

### Syntax

```
Sub SetCryptMethods(method As CryptMethodTy  
Sub SetCryptMethods(stringMethod As CryptMe  
streamMethod As CryptMethodType)
```

- may throw `Exception()`

### Params

Name	Description
method	The crypt method for strings and stream
stringMethod	The crypt method for strings.
streamMethod	The crypt method for streams.

The default crypt method for [Type](#) 4 is V2, which uses RC4  
Crypt method settings are in effect only when [Type](#) is 4 or a

The `CryptMethodType` enumeration can take any of the foll

- None – (Not supported.) An exception is thrown when i
- Identity – No encryption.
- V2 – Uses the RC4 algorithm.
- AESV2 – Uses the AES algorithm with 128-bit encrypti
- AESV3 – Uses the AES algorithm with 256-bit encrypti

---

#### Notes

Because Adobe Reader does not support using more than one encryption method per document (i.e. `stringMethod≠streamMethod`), an exception is thrown if you try to use multiple crypt methods. However, you can use the `Identity` method to degenerate and can be used with other crypt methods.

Here we use 128-bit AES encryption.

#### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.AddText("Hello World!");
theDoc.Encryption.Type = 4;
theDoc.Encryption.SetCryptMethods(CryptMethod.AESV2);
theDoc.Save(Server.MapPath("docencrypt.pdf"))
```

---

#### Example

#### [Visual Basic]

```
Dim theDoc As New Doc()
theDoc.FontSize = 96
theDoc.AddText("Hello World!")
theDoc.Encryption.Type = 4
theDoc.Encryption.SetCryptMethods(CryptMethod.AESV2)
theDoc.Save(Server.MapPath("docencrypt.pdf"))
```



# ToString Function

---

Returns a string representation of the object.

## [C#]

```
override string ToString()
```

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Syntax

Name	Description
return	The string representation of the object.

---

### Params

This method returns the string value of the object. This is equivalent to reading the [String](#) property of the object.

---

### Notes

None.

---

### Example



# CanAssemble Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can assemble the document.

This property determines if people who supply the user password can assemble the document.

Assembling is defined as inserting, rotating or deleting pages; creating bookmarks or thumbnail images. This is allowed even if the [CanChange](#) property is set to false.

---

## Notes

This property is only applied if the [Type](#) is set to 2 or higher.

See the [Doc.Encryption](#) property.

---

## Example

# CanChange Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can modify the document.

This property determines if people who supply the user password can change the document.

Changing is defined as modifying the document in any way other than those controlled by the [CanEdit](#), [CanFillForms](#) or [CanAssemble](#) properties.

---

## Notes

This property is only applied if the [Type](#) is set to 1 or higher.

See the [Doc.Encryption](#) property.

---

## Example

# CanCopy Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can copy from the document.

This property determines if people who supply the user password can copy from the document.

Copying is defined as copying, or otherwise extracting text and graphics from the document. If the [Type](#) is set to 3 and the [CanExtract](#) property is set to true then extracting text and graphics in support of accessibility to disabled users is allowed.

---

## Notes

This property is only applied if the [Type](#) is set to 1 or higher.

See the [Doc.Encryption](#) property.

---

## Example

# CanEdit Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can edit the document.

This property determines if people who supply the user password can edit the document.

Editing is defined as adding or modifying text annotations, filling in interactive form fields and if the [CanChange](#) property is set, creating or modifying interactive form fields including signature fields.

---

## Notes

This property is only applied if the [Type](#) is set to 1 or higher.

See the [Doc.Encryption](#) property.

---

## Example

# CanExtract Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can extract from the document.

This property determines if people who supply the user password can extract from the document.

Extracting is defined as extracting text or graphics in support of accessibility to disabled users or for other purposes.

---

## Notes

This property is only applied if the [Type](#) is set to 2 or higher.

See the [Doc.Encryption](#) property.

---

## Example

# CanFillForms Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can fill forms in the document.

This property determines if people who supply the user password can fill forms in the document.

Filling forms is defined as filling in existing interactive form fields including signature fields. This is allowed even if the [CanEdit](#) property is set to false.

---

## Notes

This property is only applied if the [Type](#) is set to 2 or higher.

See the [Doc.Encryption](#) property.

---

## Example

# CanPrint Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can print the document.

This property determines if people who supply the user password can print the document.

Printing may not be available at the highest quality level if the [CanPrintHi](#) property is set to false.

---

#### Notes

This property is only applied if the [Type](#) is set to 1 or higher.

See the [Doc.Encryption](#) property.

---

#### Example

# CanPrintHi Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether a user can print a high resolution copy of the document.

This property determines if people who supply the user password can print a high resolution copy of the document.

---

## Notes

When this property is set to false and the [CanPrint](#) property is set to true printing is limited to a lower quality representation of the document.

This property is only applied if the [Type](#) is set to 2 or higher.

---

## Example

See the [Doc.Encryption](#) property.

# EncryptMetadata Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to encrypt the document metadata for encryption levels of <b>type</b> 4 or above.

This property determines whether the document metadata is encrypted. It is in effect only when **Type** is 4 or above.

---

## Notes

The document metadata can be saved unencrypted so that tools that do not support PDF encryption can still index/process the document using the metadata.

---

## Example

None.

# OwnerPassword Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			The owner password.
[Visual Basic] <code>String</code>	""	No	

This property determines or reflects the owner password for the document.

You can open and view an encrypted document with either the user or the owner password. However only the owner has full control over the document. The user may have a restricted set of permissions.

---

## Notes

For more information about the user and owner passwords see the [Doc.Encryption](#) property.

See the [Doc.Encryption](#) property.

---

## Example

# Password Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	""	No	The user password.

This property determines or reflects the user password for the document.

You can open and view an encrypted document with either the user or the owner password. However only the owner has full control over the document. The user may have a restricted set of permissions.

---

## Notes

Typically the user password is left blank to allow all users to view the document.

For more information about the user and owner passwords see the [Doc.Encryption](#) property.

---

## Example

See the [Doc.Encryption](#) property.

# StreamCryptonMethod Property

---



Type	Default Value	Read Only	Description
[C#] CryptMethodType	V2	Yes	The crypt method for streams for encryption levels of <a href="#">type 4</a> or above.
[Visual Basic] CryptMethodType			

This property returns the crypt methods for streams. Crypt method settings are in effect only when [Type](#) is 4 or above.

---

## Notes

To change this value, use [SetCryptMethods](#).

None.

---

## Example

# String Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	Variable	No	The encryption settings as a string.

A string representation of the encryption settings.

---

## Notes

This covers all the properties of this class and can be used for a save and restore stack.

---

## Example

None

# StringCryptonMethod Property

---



Type	Default Value	Read Only	Description
[C#] CryptMethodType	V2	Yes	The crypt method for strings for encryption levels of <a href="#">type 4</a> or above.
[Visual Basic] CryptMethodType			

This property returns the crypt methods for strings. Crypt method settings are in effect only when [Type](#) is 4 or above.

---

## Notes

To change this value, use [SetCryptMethods](#).

None.

---

## Example

# Type Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	0	No	The level of encryption to use.

This property determines the level of encryption used when saving the document. The default value of zero indicates that no encryption will be used.

Higher levels of encryption provide higher levels of security and more flexibility in applying permissions but also require more recent versions of Adobe Acrobat to view. The table below details valid combinations.

Type Value	0	1	2	4	5
<b>Acrobat Version Required</b>	Any	2.0	5.0	7.0	9.0
<b>Encryption Key Length (bits)</b>	None	40	128	128	256
<b>Encryption Algorithm</b>	Identity	RC4	RC4	Identity/RC4/AES	Identity/AES
<b>Can Assemble</b>			●	●	●

Notes

Can Change		●	●	●	●
Can Copy		●	●	●	●
Can Edit		●	●	●	●
Can Extract			●	●	●
Can Fill Forms			●	●	●
Can Print		●	●	●	●
Can Print Hi			●	●	●

See the [Doc.Encryption](#) property and the [SetCryptMethods](#) method.

---

**Example**



## FindAll Function

---

Find all the fonts currently installed on the system.

### [C#]

```
static XFont[] FindAll()
```

### [Visual Basic]

```
Shared Function FindAll() As  
XFont()
```

---

#### Syntax

Name	Description
return	The set of matching fonts.

---

#### Params

This function finds all the fonts currently installed on the system.

---

#### Notes

None.

---

#### Example





# FindFamily Function

---

Find all the fonts belonging to a particular family.

## [C#]

```
static XFont[] FindFamily(string family)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FindFamily(family As String) As XFont()
```

---

## Params

Name	Description
family	The name of the font family.
return	The set of matching fonts.

---

## Notes

This function finds all the fonts belonging to a particular font family.

---

## Example

None.





# FindFamilyNames Function

---

Find the names of all font families.

## [C#]

```
static string[] FindFamilyNames()
```

---

## Syntax

## [Visual Basic]

```
Shared Function FindFamilyNames()  
As String()
```

---

## Params

Name	Description
return	The set of family names.

---

## Notes

This function finds all the font families currently available.

---

## Example

None.





## FindByName Function

---

Find all the fonts with a given name.

### [C#]

```
static XFont[] FindByName(string  
name)
```

---

### Syntax

### [Visual Basic]

```
Shared Function FindByName(name  
As String) As XFont()
```

---

### Params

Name	Description
name	The name of the font.
return	The set of matching fonts.

This function finds all the fonts with a given name.

---

### Notes

Note that each font may contain many different names. Some names are more unique than others. So it is good to be as specific as possible when searching by name.

---

**Example**

None.

# FindByStyle Function

---



Find a font from a specific family, with a given style.

## [C#]

```
static XFont FindByStyle(string
family, FontWeight weight, bool
italic)
static XFont FindByStyle(string
family, int weight, bool italic)
static XFont
FindByStyle(IEnumerable<XFont>
fonts, int weight, int
weightTolerance, bool italic)
```

## [Visual Basic]

```
Shared Function
FindByStyle(family As String,
weight As FontWeight, italic As
Boolean) As XFont
Shared Function
FindByStyle(family As String,
weight As Integer, italic As
Boolean) As XFont
Shared Function FindByStyle(fonts
As IEnumerable(Of XFont), weight
As Integer, weightTolerance as
Integer, italic As Boolean) As
XFont
```

Syntax

---

**Params**

Name	Description
family	The name of the font family.
fonts	The fonts.
weight	The weight (or the midpoint of the permitted weight range) of the required font.
weightTolerance	The maximum permitted difference between the weight parameter and the weight of the required font. The default value is 0.
italic	Whether an italic font is required.
return	A matching font. Null if no matching font could be found.

This function finds a font from a specific family or among the given fonts, with a given style.

If no matching font can be found, then the function will return null.

---

**Notes**

The desired weight can be passed in either as an integer or as a value from the [FontWeight](#) enumeration.

---

None.

## Example



# TextWidth Function

---

Calculate the width of a string of text.

## [C#]

```
int TextWidth(string text)
```

## [Visual Basic]

```
Function TextWidth(text As  
String) As Integer
```

---

### Syntax

---

### Params

Name	Description
text	The text
return	The width of the text in 1000ths.

This function calculates the width for a given text string.

---

### Notes

The value returned is measured in thousandths of a unit. So to calculate the physical size on the page multiply the returned value by the font size and divide by one thousand.

None.

---

**Example**



## Unload Function

---

Unloads a font so that it is no longer available.

### [C#]

```
void Unload()
```

---

### Syntax

### [Visual Basic]

```
Sub Unload()
```

---

### Params

Name	Description
none	

Unloads a font so that it is no longer available.

It is important that you ensure that the fonts is not being used by ABCpdf at the point that it is unloaded. This means you need to consider if other threads might be constructing PDF documents that are making use of the font. Unloading a font which is in use may result in unpredictable behavior or output.

---

### Notes

---

Example

None.

# FamilyName Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The name of the family to which the font belongs.

The name of the family to which the font belongs.

Fonts often come in collections providing variations on a particular style. Although the font name for each style may be different the family name will be the same.

---

## Notes

None.

---

## Example

# FixedPitch Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the font is in a fixed pitch style.

The value of this property is true if the font is in a fixed pitch style.

---

## Notes

Fixed pitch fonts have characters which are all the same width. Courier is an example of such a font.

---

## Example

None.

# Italic Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the font is in an italic style.

---

## Notes

The value of this property is true if the font is in an italic style.

---

## Example

None.

# Name Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The complete human readable name of the font.

The complete human readable name of the font.

---

## Notes

Font names are not guaranteed to be unique. However it is unusual to find duplicates.

None.

---

## Example

# Names Property

---



Type
<b>[C#]</b> <code>System.Collections.Specialized.StringCollection</code>
<b>[Visual Basic]</b> <code>System.Collections.Specialized.StringCollection</code>

The complete set of names by which this font is known.

---

## Notes

Fonts can hold multiple names. These can be short names, long names, names for a particular purpose or names in different languages.

None.

---

## Example

# PostScriptName Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The name the font will be known by on a PostScript printer.

The name the font will be known by on a PostScript printer.

---

## Notes

PDF documents generally refer to fonts by their PostScript name.

None.

---

## Example

# Script Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the font is in a script style.

The value of this property is true if the font is in a script style.

---

## Notes

Script styles are designed to look like cursive handwriting. Comic Sans MS is an example of such a font.

---

## Example

None.

# Serif Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the font is in a serif style.

The value of this property is true if the font is in a serif style.

Fonts come in serif and sans-serif styles. Serif styles have structural elements on the end of strokes. Sans-serif styles do not.

---

## Notes

Times is an example of a font with serifs. Arial is an example of one without.

None.

---

## Example

# Weight Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The weight of the font.

The weight of the font determines whether the font is light, normal or bold.

The value of this property can be compared against the `FontWeight` enumeration.

The `FontWeight` enumeration contains the following values:

- `DontCare` = 0
- `Thin` = 100
- `ExtraLight` = 200
- `Light` = 300
- `Normal` = 400
- `Medium` = 500
- `SemiBold` = 600
- `Bold` = 700
- `ExtraBold` = 800
- `Heavy` = 900

---

Notes

---

**Example**

None.

# Baseline Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The baseline of the font in thousandths of a unit.

The baseline of the font in thousandths of a unit.

To find the base of the font use this metric together with the font size to find the distance from the top of the text area to the baseline of your font.

---

## Notes

For example if the baseline is 800 and the font size is 100 points, then the base of the font will be 80 points from the top of the current rectangle.

None.

---

## Example

# Widths Property

---



Type	Default	Read Only	Description
[C#] <code>int[]</code>			
[Visual Basic] <code>Integer()</code>	n/a	Yes	The widths of the characters in the font.

The character widths for all the characters in the font.

The array is indexable by Unicode value. For example to find the width of a space (ASCII 32) you would simply reference item 32 in the array.

---

## Notes

The values are measured in in 1000ths of a PDF unit. Characters with no representation in the font are assigned a width of -1.

None.

---

## Example



## GetFieldNames Function

---

Gets the full names of all the fields in the document.

### [C#]

```
string[] GetFieldNames()
```

---

### Syntax

### [Visual Basic]

```
Function GetFieldNames() As  
String()
```

---

### Params

Name	Description
return	The names of all the eForm fields in the document.

---

### Notes

This function scans the field tree and extracts the full name of every terminal field in the document.

---

### Example

None.





## MakeFieldsUnique Function

---

Makes shared XForm fields unique.

### [C#]

```
void MakeFieldsUnique()  
void MakeFieldsUnique(string  
token)
```

---

### Syntax

### [Visual Basic]

```
Sub MakeFieldsUnique()  
Sub MakeFieldsUnique(token As  
String)
```

---

### Params

Name	Description
token	A token appended to the existing field name, together with the page number. This makes the field name unique. Default value is "_page".

Makes shared XForm fields unique.

After duplicating pages via calls such as [RemapPages](#) and [Append](#), fields may end up shared by pages. This means that updating a

field in a page will affect the same field in a different page. Very often this is not the desired behaviour. Fields should then be made unique. Changing fields in a page will not affect other pages when fields are unique. Call this method to make the fields unique. For example: `RemapPages("1 1")` creates two copies of the first page, which means that any field owned by the first page is now shared by two pages. Calling `MakeFieldsUnique` after `RemapPages`, will create separate copies of the fields, each one with a unique name.

To make names unique we use the token parameter. This is a string that is appended to the field name. It is also followed by the page number. For example, you can set the token to something like `"_page"`, the default value. The field names will then be changed to `ExistingName_page1`, where `ExistingName` is the current name of the field and the field is located on page 1. Fields must have unique names, this is required by the PDF specifications.

---

Notes

If the token already exists in the field name - and is located at the end except for digits - only the page number is changed. The token will not be appended twice. Therefore an existing field name that ends with digits, and has the exact token before such digits, will have the digits replaced by the page number and nothing else changed.

The token may be empty but be aware that any digits at the end of the field name will be replaced by the page number. If the token is null it will be replaced by the default token, `_page`.

If a field is shared by several widget annotations either on the same or different pages, we do our best not to break this sharing when the fields are made unique. When duplicating a shared field, the page number in the field name will be the number of the last page the field appeared on. For example, take a field called "title" and shared between pages 1 and 2. The following remap is performed: "1 1 2 2". Then MakeFieldsUnique is called. The first and third pages will then share a field called title\_page3 and the second and forth pages will share a field called title\_page4. If remapping with "1 2 1 2", then the first two pages share title\_page2 and the last two pages share title\_page4.

---

**Example**

None.



# AddResource Function

---

Add a particular type of resource to the form

## [C#]

```
string AddResource(IndirectObject  
resource, string type, string  
name)
```

---

## Syntax

## [Visual Basic]

```
Function AddResource(resource As  
IndirectObject, type As String,  
name As String) As String
```

---

## Params

Name	Description
resource	The resource to be added.
type	The type of resource.
name	The format of the name that should be used.

Add a particular type of resource to the form.

Forms may contain default resources usable by any of the fields and annotations in the form. The most common resource types are "Font", "XObject" and "ColorSpace". For further details

see the PDF Specification.

---

**Notes**

This method allows you to add new resources to the form. You may supply your own name for the resource but if the name is already in use, it may need to be modified. For this reason the function returns the value which was actually used for the addition.

---

**Example**

None.



## Item Function

---

Returns a particular field referenced by full name.

### [C#]

```
Field this[string name]
```

---

### Syntax

### [Visual Basic]

```
Default Property Item(name As String) As Field
```

---

### Params

Name	Description
name	The fully qualified name of the field.
return	The matching field.

Use this method to retrieve a field referenced by fully qualified name.

If no matching field can be found then a null value will be returned.

---

### Notes

In C# this property is the indexer for the class.

See the [XForm](#) object for details of how fully qualified names are constructed.

---

**Example**

None.



## Refresh Method

---

Refresh and reload the document fields.

### [C#]

```
void Refresh()
```

---

### Syntax

### [Visual Basic]

```
Sub Refresh()
```

Name	Description
none	

---

### Params

Use this method to refresh and reload the document fields.

When the form is first requested field data is cached. This allows a level of optimization which would not otherwise be possible.

---

### Notes

However if you are using the low level functionality to modify the field structure the cache will not reflect your changes. In this situation you can force the fields to be reloaded by calling Refresh.

---

**Example**

None.



## Stamp Method

---

Stamp all fields into the document.

### [C#]

```
void Stamp()
```

---

### Syntax

### [Visual Basic]

```
Sub Stamp()
```

Name	Description
none	

---

### Params

Use this method to permanently stamp all fields into the document.

When this method is called all field appearances are stamped permanently into the document and the fields are deleted.

---

### Notes

Each field becomes a new layer on the page (see [Doc.LayerCount](#)) so you may wish to call [Doc.Flatten](#) on any affected pages.

You can use the [Field.Stamp](#) method to stamp individual fields into the document.

---

**Example**

None.

# DateTimeFormat Property

---



Type	Default Value	Read Only	Description
[C#] IFormatProvider	null	No	The format provider for formatting dates and times.
[Visual Basic] IFormatProvider			

This property specifies the formats of dates and times for the appearances of fields that use javascript functions such as AFDate\_FormatEx, AFTIME\_FormatEx.

A null value indicates the use of the locale-independent default formats.

---

## Notes

You can see the appearances when the PDF is rendered to an image.

If the fields are not stamped, Adobe Reader may ignore the appearances and generate new appearances so you may not see the dates and times in the specified formats when viewing in Adobe Reader.

---

None.

**Example**

# Fields Property

---



Type	Default Value	Read Only	Description
[C#] <b>Fields</b>	See description.	Yes	All top level fields in the form.
[Visual Basic] <b>Fields</b>			

The collection of all top level fields in the form.

Items in this collection can be referenced by partial field name or by zero based index.

---

## Notes

As with all collections you can use the Count property to determine the number of items contained and you can iterate through the collection using the standard methods appropriate to the language you are coding in.

---

## Example

None.

# FormatFields Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	true	No	Whether values should be formatted before insertion into fields.

This determines if values should be formatted before insertion.

Acrobat uses JavaScript to implement number and date formats. Although this is something which is part of Acrobat rather than of the PDF specification you can ask ABCpdf to adhere to these formatting scripts.

---

## Notes

When this property is set to true ABCpdf will detect these standard scripts and generate appearances conforming with them.

It is unlikely you will want to change the value of this property .

None.

---

## Example



# GenerateAppearances Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether field appearances should be pre-generated.

This property determines if field appearances are pre-generated or not.

Many PDF viewing applications are unable to generate field appearances dynamically. This means that they are unable to display fields given the field and the value of that field. Instead they require the appearance of the field to be pre-generated.

---

## Notes

A new appearance will only be generated when it is required. Typically this is when the value of a field changes.

It is unlikely you will want to change the value of this property.

None.

---

## Example



# NeedAppearances Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether the viewer should automatically regenerate field appearances.

This property determines if the NeedAppearances flag in the PDF is set when field values are changed.

The NeedAppearances flag signals to the viewing application that field appearances should be automatically generated rather than using the appearance embedded in the PDF.

---

## Notes

It is generally best to allow the viewing application to generate its own appearances and use the pre-generated ones as a fallback.

It is unlikely you will want to change the value of this property.

None.

---

## Example



# EndTasks Method

---



Ends any HTML Engine worker threads or processes.

## [C#]

```
int EndTasks(TaskState condition)
```

---

## Syntax

## [Visual Basic]

```
Function EndTasks(condition As TaskState) As Integer
```

Name	Description
condition	<p>The way in which to select threads or processes that should be ended. The TaskState enumeration may take the following values:</p> <ul style="list-style-type: none"><li>• None — no thread or process is selected.</li><li>• Idle — idle threads or processes are selected.</li><li>• AllWhenBecomeldle — all threads or processes are selected, and to busy threads or processes, the operation is applied only when they become idle.</li></ul>

---

## Params

	<ul style="list-style-type: none"><li>• All — all threads or processes are selected.</li></ul>
return	The number of selected threads or processes.

Use this method to end worker threads or processes for the [HTML Engine](#). HTML Engine may be executed in separate threads or processes for isolation.

---

**Notes**

If condition specifies All, busy threads or processes will be ended immediately, and this may cause unexpected behavior in other threads of your application that depend on the busy threads or processes.

---

**Example**

None.



## GetHttpStatusCode Method

---

Retrieves the HTTP status code.

### [C#]

```
int GetHttpStatusCode(int id)
```

---

### Syntax

### [Visual Basic]

```
Function GetHttpStatusCode(id As Integer) As Integer
```

---

### Params

Name	Description
id	The Object ID of the web page to be accessed.
return	The HTTP status code or zero if not available.

Use this method to retrieve the HTTP status code if the URL uses the HTTP protocol.

The ID should be obtained from a call to [Doc.AddImageUrl](#).

---

### Notes

If the [PageLoadMethod](#) property is [WebBrowserNavigate](#), only error status codes are available. Non-error status codes, such as

200, are not available.

Example

None.



## GetScriptReturn Method

---

Retrieves the client side onload script return value.

### [C#]

```
string GetScriptReturn(int id)
```

---

### Syntax

### [Visual Basic]

```
Function GetScriptReturn(id As Integer) As String
```

---

### Params

Name	Description
id	The Object ID of the web page to be accessed.
return	The return value.

Use this method to retrieve the client side onload script return value.

The ID should be obtained from a call to [Doc.AddImageUrl](#) or [Doc.AddImageHtml](#).

---

### Notes

See the [OnLoadScript](#) property for further details.

---

**Example**

See the [UseScript](#) property.

# GetTagIDs Function

---



Gets an array of the HTML IDs of tagged visible items.

## [C#]

```
string[] GetTagIDs(int id)
```

---

## Syntax

## [Visual Basic]

```
Function GetTagIDs(id As Integer)  
As String()
```

---

## Params

Name	Description
id	The Object ID of the object.
return	The IDs of tagged visible HTML objects.

Use this method to retrieve the HTML IDs of tagged visible items.

To use this method you need to enable the tagging functionality. See the [AddTags](#) property for details.

This function takes an ID obtained from a call to [Doc.AddImageUrl](#), [Doc.AddImageHtml](#) or

[Doc.AddImageToChain](#) and returns the IDs of any items which are visible on the PDF page as a result of that call.

---

**Notes**

For example the ID associated with the following paragraph is "p1".

```
<p id="p1" style="abcpdf-tag-visible: true">Gallia est omnis  
divisa in partes tres.</p>
```

The IDs may be repeated if the objects are split over more than one area.

The IDs match up directly on a one-to-one basis with the [XRects](#) returned by the [GetTagRects](#) or the [GetTagUntransformedRects](#) function.

---

**Example**

See the [GetTagRects](#) method.



## GetTagRects Function

---

Gets an array of the locations of tagged visible items.

### [C#]

```
XRect[] GetTagRects(int id)
```

---

### Syntax

### [Visual Basic]

```
Function GetTagRects(id As Integer) As XRect
```

---

### Params

Name	Description
id	The Object ID of the object.
return	The location of tagged visible HTML objects.

Use this method to retrieve the locations of tagged visible items. The locations are to be used with [Doc.Transform](#) being identity.

To use this method you need to enable the tagging function [AddTags](#) property for details.

---

### Notes

This function takes an ID obtained from a call to [Doc.AddImage](#), [Doc.AddImageHtml](#) or [Doc.AddImageToChain](#) and returns an array of [XRect](#) objects for any items which are visible on the PDF page as a result of the call.

The locations match up directly on a one-to-one basis with the IDs returned by the [GetTagIDs](#) function.

The following example shows the effect that this parameter rendering.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(100, 100);
theDoc.Rect.Top = 700;
// Tag elements with style 'abcpdf-tag-visible'
theDoc.HtmlOptions.AddTags = true;
int id = theDoc.AddImageHtml("<FONT id=\"p1
style=\"abcpdf-tag-visible: true; font-size
est omnis divisa in partes tres.</FONT>");
// Frame location of the tagged element
XRect[] tagRects = theDoc.HtmlOptions.GetTagRects();
foreach (XRect theRect in tagRects) {
    theDoc.Rect.String = theRect.ToString();
    theDoc.FrameRect();
}
// Output tag ID
string[] tagIds = theDoc.HtmlOptions.GetTagIds();
theDoc.Rect.String = theDoc.MediaBox.String;
theDoc.Rect.Inset(20, 20);
theDoc.FontSize = 64;
theDoc.Color.String = "255 0 0";
theDoc.AddText("Tag ID \" + tagIds[0] + "\"");
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsGetTagRects.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(100, 100)
theDoc.Rect.Top = 700
```

---

**Example**

```
' Tag elements with style 'abcpdf-tag-visible'
theDoc.HtmlOptions.AddTags = true
Dim id As Integer
id = theDoc.AddImageHtml("<FONT id=""p1"" style=""tag-visible: true; font-size: 72pt"">Gallia
divisa in partes tres.</FONT>")
' Frame location of the tagged element
Dim tagRects As XRect()
tagRects = theDoc.HtmlOptions.GetTagRects(id)
Dim theRect As XRect
For Each theRect in tagRects
    theDoc.Rect.String = theRect.ToString()
    theDoc.FrameRect()
Next
' Output tag ID
Dim tagIds As String()
tagIds = theDoc.HtmlOptions.GetTagIDs(id)
theDoc.Rect.String = theDoc.MediaBox.String
theDoc.Rect.Inset(20, 20)
theDoc.FontSize = 64
theDoc.Color.String = "255 0 0"
theDoc.AddText("Tag ID "" + tagIds(0) + """)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsGetT
theDoc.Clear()
```

Tag ID "p1":

Gallia est omnis  
divisa in partes tres.

HtmlOptionsGetTagRects.pdf

# GetTagUntransformedRects Function

---



Gets an array of the locations of tagged visible items before [Doc.Transform](#) is applied.

## [C#]

```
XRect[]  
GetTagUntransformedRects(int id)
```

---

## Syntax

## [Visual Basic]

```
Function  
GetTagUntransformedRects(id As  
Integer) As XRect()
```

---

## Params

Name	Description
id	The Object ID of the object.
return	The location (before Doc.Transform is applied) of tagged visible HTML objects.

Use this method to retrieve the locations of tagged visible items. The locations are to be used with the value of [Doc.Transform](#) the same as when the ID is obtained.

To use this method you need to enable the tagging functionality. See the [AddTags](#) property for details.

---

**Notes**

This function takes an ID obtained from a call to [Doc.AddImageUrl](#), [Doc.AddImageHtml](#) or [Doc.AddImageToChain](#) and returns the locations of any items which are visible on the PDF page as a result of that call.

The locations match up directly on a one-to-one basis with the IDs returned by the [GetTagIDs](#) function.

---

**Example**

See the [GetTagRects](#) method.



## LinkDestinations Method

---

Convert a restricted selection of external links to internal links.

### [C#]

```
int LinkDestinations(IEnumerable<int> ids)
int LinkDestinations(IEnumerable<int> linkIDs,
IEnumerable<int> destIDs, bool linkPages)
```

---

### Syntax

### [Visual Basic]

```
Sub LinkDestinations(ids As IEnumerable(Of Integer))
Sub LinkDestinations(linkIDs As IEnumerable(Of Integer), destIDs As IEnumerable(Of Integer), linkPages As Boolean)
```

---

### Params

Name	Description
ids	Specifies both linkIDs and destIDs.
linkIDs	The list of IDs of view objects containing links (a href attributes).
destIDs	The list of IDs of view objects containing destination tags with name attributes).
linkPages	Whether links pointing to the URLs of HTML page fragments) are converted to internal links. The default is false.
return	The number of links converted.

This method scans the view objects specified in linkIDs and converts them to internal links where the destinations are found in the specified in destIDs. It is similar to the [LinkPages](#) method but restricts the conversion to lists of view objects.

By default, links in rendered HTML are preserved as is. This means that in a web page link to external URLs. When you click on the link, a browser window will be launched and the original target of the link displayed.

In some situations, you may wish to resolve links within the PDF rather than launch a browser window. This method allows you to do this by converting the links so that they take you between pages in the PDF rather than launch a browser window.

---

#### Notes

For example, you might add a number of web pages which link to each other. Rather than linking to the pages on the original document, you might like to resolve the links so that they point at the pages which appear in the PDF.

Similarly, if you use named destinations (HTML fragments) in your document, you may wish to use this method to convert external links to internal ones.

This example shows how to import an HTML page which uses named destinations.

We first create a [Doc](#) object and inset the edges a little so that the page is in the middle of the page. We assign the appropriate HTML options so that the HTML can be rendered live.

#### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(18, 18);
theDoc.HtmlOptions.AddLinks = true;
```

#### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Rect.Inset(18, 18)  
theDoc.HtmlOptions.AddLinks = True
```

We add the pages to the document.

### [C#]

```
List<int> theList = new List<int>();  
int theID =  
theDoc.AddImageUrl("http://www.websupergoo.  
while (true) {  
    theList.Add(theID);  
    if (!theDoc.Chainable(theID))  
        break;  
    theDoc.Page = theDoc.AddPage();  
    theID = theDoc.AddImageToChain(theID);  
}
```

### [Visual Basic]

```
Dim theList As List(Of Integer) = New List(  
Dim theID As Integer  
theID =  
theDoc.AddImageUrl("http://www.websupergoo.  
While True  
    theList.Add(theID)  
    If Not theDoc.Chainable(theID) Then  
        Exit While  
    End If  
    theDoc.Page = theDoc.AddPage()  
    theID = theDoc.AddImageToChain(theID)  
End While
```

The URL we've referenced makes extensive use of named these named destination links to take us between pages on taking us to the original URL.

---

**Example**

After adding the pages, we can flatten them. We can't do that if the pages have been added because flattening will invalidate our previous chain.

**[C#]**

```
theDoc.HtmlOptions.LinkDestinations(theList
for (int i = 1; i <= theDoc.PageCount; i++)
    theDoc.PageNumber = i;
    theDoc.Flatten();
}
```

**[Visual Basic]**

```
theDoc.HtmlOptions.LinkDestinations(theList
For i As Integer = 1 To theDoc.PageCount
    theDoc.PageNumber = i
    theDoc.Flatten()
Next
```

Finally, we save.

**[C#]**

```
theDoc.Save(Server.MapPath("linkpages.pdf"))
theDoc.Clear();
```

**[Visual Basic]**

```
theDoc.Save(Server.MapPath("linkpages.pdf"))
theDoc.Clear()
```

We get the following output. The links work within the PDF.



linkpages.pdf [Page 1]



linkpages.pdf [F



## LinkPages Method

---

Convert external links to internal links wherever possible.

### [C#]

```
void LinkPages()
```

---

### Syntax

### [Visual Basic]

```
Sub LinkPages()
```

Name	Description
return	n/a.

---

### Params

This method scans the entire document converting external links wherever possible. As an alternative you can restrict the conversion by using the [LinkDestinations](#) method.

By default, links in rendered HTML are preserved as is. This means that in a web page link to external URLs. When you click on the link, a browser window will be launched and the original target of the link displayed.

In some situations, you may wish to resolve links within the document so that they take you between pages in the PDF rather than launch a browser window.

---

### Notes

For example, you might add a number of web pages which link to each other. Rather than linking to the pages on the original

might like to resolve the links so that they point at the pages appear in the PDF.

Similarly, if you use named destinations (HTML fragments) document, you will may wish to use this method to convert links to internal ones.

This example shows how to import an HTML page which us

We first create a [Doc](#) object and inset the edges a little so t in the middle of the page. We assign the appropriate HTML be rendered live.

#### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(18, 18);
theDoc.HtmlOptions.AddLinks = true;
```

#### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(18, 18)
theDoc.HtmlOptions.AddLinks = True
```

We add the pages to the document.

#### [C#]

```
int theID =
theDoc.AddImageUrl("http://www.websupergoo.
while (true) {
    if (!theDoc.Chainable(theID))
        break;
    theDoc.Page = theDoc.AddPage();
    theID = theDoc.AddImageToChain(theID);
}
```

---

## [Visual Basic]

```
Dim theID As Integer
theID =
theDoc.AddImageUrl("http://www.websupergoo.
While True
    If Not theDoc.Chainable(theID) Then
        Exit While
    End If
    theDoc.Page = theDoc.AddPage()
    theID = theDoc.AddImageToChain(theID)
End While
```

The URL we've referenced makes extensive use of named these named destination links to take us between pages on taking us to the original URL.

---

### Example

After adding the pages, we can flatten them. We can't do th have been added because flattening will invalidate our prev chain.

## [C#]

```
theDoc.HtmlOptions.LinkPages();
for (int i = 1; i <= theDoc.PageCount; i++)
    theDoc.PageNumber = i;
    theDoc.Flatten();
}
```

## [Visual Basic]

```
theDoc.HtmlOptions.LinkPages()
For i As Integer = 1 To theDoc.PageCount
    theDoc.PageNumber = i
    theDoc.Flatten()
Next
```

Finally, we save.

[C#]

```
theDoc.Save(Server.MapPath("linkpages.pdf"))  
theDoc.Clear();
```

[Visual Basic]

```
theDoc.Save(Server.MapPath("linkpages.pdf"))  
theDoc.Clear()
```

We get the following output. The links work – where possible



linkpages.pdf [Page 1]



linkpages.pdf [Page 2]



## PageCacheClear Method

---

Clears the HTML page cache.

### [C#]

```
void PageCacheClear()
```

---

### Syntax

### [Visual Basic]

```
Sub PageCacheClear()
```

---

### Params

Name	Description
return	n/a.

ABCpdf holds a cache of recently requested URLs and it's only after five minutes or so that these pages expire from the cache.

---

### Notes

This results in a considerable degree of optimization for many common operations.

You can clear the cache of all pages by calling this method.

---

**Example**

None.



## PageCachePurge Method

---

Purges the HTML page cache.

### [C#]

```
void PageCachePurge()
```

---

### Syntax

### [Visual Basic]

```
Sub PageCachePurge()
```

---

### Params

Name	Description
return	n/a.

ABCpdf holds a cache of recently requested URLs and it's only after five minutes or so that these pages expire from the cache.

---

### Notes

This results in a considerable degree of optimization for many common operations.

You can clear the cache of all expired pages by calling this method.

---

Example

None.



## SetTheme Function

---

Specify whether to use Windows themes or not.

### [C#]

```
void SetTheme(bool useTheme)
void SetTheme(bool useTheme, bool
noTheme)
```

---

### Syntax

### [Visual Basic]

```
Sub SetTheme(useTheme As Boolean)
Sub SetTheme(useTheme As Boolean,
noTheme As Boolean)
```

---

### Params

Name	Description
useTheme	Whether to use the current theme.
noTheme	Whether not to use the current theme. The default value is the negation of useTheme.

Sets the values of [UseTheme](#) and [NoTheme](#). You will generally want to use the function which accepts only one parameter.

The theme is only relevant if you are using the MSHTML engine for IE9 or earlier. As such it is

likely that this function will be deprecated in future releases of ABCpdf.

The theme is the user interface design for text boxes, buttons, combo boxes and other interactive features which may be found on web pages. Different versions of Windows have different themes. If you do not use the current theme (typically Windows XP/Aero Glass) you will get the default Windows 95 appearances.

---

**Notes**

The fact that you can set two different values is a reflection of the fact that these two inputs correspond to the Windows flags `DOCHOSTUIFLAG_THEME` and `DOCHOSTUIFLAG_NOTHEME` respectively. While there seems little point in setting both to the same boolean value, the fact that Windows allow it is why it is reflected in this function call. If `UseTheme` and `NoTheme` are the same no theme is applied.

None.

---

**Example**

# Engine Property

---



Type	Default Value	Read Only	Description
[C#]EngineType			
[Visual Basic]EngineType	MSHtml	No	The engine to use for rendering operations.

This property specifies the HTML engine to use when importing HTML documents.

The EngineType enumeration can take any of the following values:

- **MSHtml. Characteristics:**
  - The Screen style sheets are used during the rendering.
  - Page heights can vary across different pages. Segment images are dynamically composed to fit the target rectangle as accurately as possible. This means the target rectangle is scaled to fit the contents for larger target rectangles instead of scaled to fit the contents.
  - For definite control of HTTP requests, you should set the `HttpTracing` value.

## Internet Explorer 9 support

ABCpdf automatically detects and accommodates the changes introduced in Microsoft Internet Explorer 9. For many scenarios, this is transparent and requires no code change.

The Fusion Log Viewer is known to interfere with the initialization of the engine. If any problem arises, try disabling it or specifying a different engine.

Currently, there are some limitations when using the MS process (See [MSHtmlBootstrap](#)):

- The [HtmlCallback](#) property is not supported.
- The [HtmlEmbedCallback](#) property is not supported.

- **Gecko. Characteristics:**

- The Gecko engine is a component independent of other System. It will not be affected even if you upgrade your local Firefox installation.
- It supports a wide range of the HTML5 and CSS3 standards.
- It supports the majority of the SVG Full profile. SVG graphics convert natively to PDF primitives. Alternatively, you can provide a location in [Doc.AddImageUrl](#) or provide a string of SVG in [Doc.AddImageHtml](#) to have it converted to PDF using the [Doc.AddImageToCanvas](#) property.
- It supports XML pages with XSLT and MathML.
- You can specify which set of style sheets to use (Print/Screen) using the [Media](#) property.
- In large tables that span across pages, thead and tfoot are rendered on different pages properly.
- The rendered HTML will always fill the entire current rendering area. Using [Doc.AddImageUrl](#) or [Doc.AddImageHtml](#), the page height is determined by the content. Subsequent pages rendered using [Doc.AddImageToCanvas](#) (instead of having the contents reflowed) to fit in the current rendering area as far as possible.
- ActiveX components are not supported.
- The [Paged](#) property is always assumed to be true.
- The Gecko engine does not use Page caching. This means that pages are never stored in a memory cache.
- On the other hand, Web caching is used to store content from the Internet and is configurable using the [UseNoCache](#) property.
- JavaScript in [OnLoadScript](#) cannot directly modify the document. `document.write("hello world");` would not work. Most people get around to get your desired effect by indirectly modifying the document as `document.body.innerHTML = "hello world";`
- The use of [OnLoadScript](#) does not require [UseScript](#). You can use `document.write("hello world");` in [OnLoadScript](#).

customize the page without allowing JavaScript executi

---

## Notes

### Gecko configuration

The Gecko engine contains a number of preferences that are a set of .js files in the "XULRunner??\_?\defaults\pref" folder in alphabetical order when it starts. Each file contains a set of preferences. Adding your own .js file to this folder, you can add in your own preferences.

If you are not familiar with the format of these files, you can view the configuration file from Firefox:

- View the URI "about:config" in Firefox.
- Modify the configuration as usual.
- Close Firefox. The configuration is saved to the file prefs.js when completely closed.
- See the Path value in %APPDATA\Mozilla\Firefox\Profiles\ProfileName\prefs.js. The value may be Profiles/sx4fgjw2.default.
- Copy prefs.js in the folder pointed to by the Path value in profile.ini) to:
  - XULRunner??\_?\defaults\pref. The preference: the **software developer**. For example, copy C:\<user>\AppData\Roaming\Mozilla\Firefox\Profiles\ProfileName\prefs.js to XULRunner??\_?\defaults\pref.
  - some place outside of XULRunner??\_? and set the registry value (file name) to the [GeckoPrefFile](#) registry value. This is considered to be set by the **end user**. See below for the difference between preferences set by the end user and the software developer.
- Do not delete or change the first line in prefs.js. You can delete other lines for your configuration change. You can delete other

Preferences set by the **end user** are in the JavaScript file prefs.js in the [GeckoPrefFile](#) registry value. (Registry values can be set in [web.config](#) or [app.config](#).) The file is loaded after all JavaScript files in "XULRunner??\_?\defaults\pref", so user-set preferences take precedence.

effect only when set by the end user. Here is an incompl

Preference in effect only if it is set by the end user	Default Value	Description
intl.accept_languages	"en-US, en"	The order of languages the specified font does not support. The Gecko engine appends missing in the following: en, zh-TW, HK. For example, if the preference is set to "en-US, en" (the default preference, a piece of sans-serif Latin font) may be imposed because some characters are not supported by Japanese fonts and some

Additional supported settings:

Preference	Default Value
print.print_shrink_to_fit	true
print.print_scaling	"1"

Each HTML engine supports a different subset of HtmlOptions methods. To use one of the filter objects: [ForMSHtml](#) and [ForGecko](#), to filter the HTML in the IDE.

# ForMSHtml Property

---



Type	Default Value	Read Only	Description
[C#] IHtmlMShtmlOptions	n/a	Yes	An object that provides access to only the HTML options supported by the MSHTML engine.
[Visual Basic] IHtmlMShtmlOptions			

The HTML options supported by the MSHTML engine.

## Supported methods

- [GetHttpStatusCode](#)
- [GetScriptReturn](#)
- [GetTagIDs](#)
- [GetTagRects](#)
- [GetTagUntransformedRects](#)
- [LinkDestinations](#)
- [LinkPages](#)
- [PageCacheClear](#)
- [PageCachePurge](#)
- [SetTheme](#)

## Supported properties

- [AddForms](#)
- [AddLinks](#)

---

Notes

- AddMovies
- AddTags
- AdjustLayout
- AutoTruncate
- BreakMethod
- BreakZoneSize
- BrowserWidth
- CoerceVector
- ContentCount
- DeactivateWebBrowser
- DisableVectorCoercion
- DoMarkup
- FontEmbed
- FontProtection
- FontSubset
- FontSubstitute
- HideBackground
- HostWebBrowser
- HtmlCallback
- HtmlEmbedCallback
- HttpAdditionalHeaders
- ImageQuality
- InitialWidth
- LogonName
- LogonPassword
- MakeFieldNamesUnique
- MaxAtomicImageSize
- NoCookie
- NoTheme
- OnLoadScript
- PageCacheEnabled
- PageCacheExpiry
- PageCacheSize
- Paged
- PageLoadMethod
- ProcessOptions (See MSHtmlBootstrap)
- ReloadPage

- RequestMethod
- RetryCount
- TargetLinks
- Timeout
- TransferModule
- UseActiveX
- UseJava
- UseNoCache
- UseResync
- UseScript
- UseTheme
- UseVideo

### [C#]

```
Doc doc = new Doc();
doc.HtmlOptions.Engine = EngineType.MShtml;
doc.HtmlOptions.ForMShtml.AddLinks = true;

// You can store a reference to the filter to
reduce code repetition
IHtmlMShtmlOptions options =
doc.HtmlOptions.ForMShtml;

options.UseActiveX = true;
options.AutoTruncate = true;

doc.AddImageUrl("http://www.websupergoo.com");
doc.Save(Server.MapPath("wsg.pdf"));
```

---

### Example

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.HtmlOptions.Engine = EngineType.MShtml
theDoc.HtmlOptions.ForMShtml.AddLinks = True

' You can store a reference to the filter to
```

```
reduce code repetition
Dim theOptions As IHTMLMSHTMLOptions =
theDoc.HTMLOptions.ForMSHTML

theOptions.UseActiveX = True
theOptions.AutoTruncate = True

theDoc.AddImageUrl("http://www.websupergoo.com'
theDoc.Save(Server.MapPath("wsg.pdf"))
```

# ForGecko Property

---

---



Type	Default Value	Read Only	Description
[C#] IHtmlGeckoOptions	n/a	Yes	An object that provides access to only the HTML options supported by the Gecko HTML engine.
[Visual Basic] IHtmlGeckoOptions			

The HTML options supported by the Gecko HTML engine.

## Supported methods

- [GetGeckoVersion](#) — also initializes the Gecko engine.
- [EndTasks](#)
- [GetHttpStatusCode](#)
- [GetScriptReturn](#)
- [GetTagIDs](#)
- [GetTagRects](#)
- [GetTagUntransformedRects](#)
- [LinkDestinations](#)
- [LinkPages](#)

## Supported properties

- [AddForms](#)
- [AddLinks](#)

---

**Notes**

- [AddMovies](#) (Flash only)
- [AddTags](#)
- [BrowserWidth](#)
- [FontEmbed](#)
- [FontSubset](#)
- [FontSubstitute](#)
- [FontProtection](#)
- [HideBackground](#)
- [ImageQuality](#)
- [ImprovePageBreakAvoid](#)
- [ImprovePageBreakInTable](#)
- [InitialWidth](#)
- [LogonName](#)
- [LogonPassword](#)
- [MakeFieldNamesUnique](#)
- [Media](#)
- [NoDefaultBackground](#)
- [NoSnapRounding](#)
- [OnLoadScript](#)
- [ProcessOptions](#)
- [RequestMethod](#)
- [RetryCount](#)
- [Timeout](#)
- [UseNoCache](#)
- [UseScript](#)

The [HttpAdditionalHeaders](#) property is supported under some circumstances but because these are unusual it is not included in this interface. For details see the documentation for this property.

**[C#]**

```
Doc doc = new Doc();  
doc.HtmlOptions.Engine = EngineType.Gecko;
```

```
doc.HtmlOptions.ForGecko.AddLinks = true;

// You can store a reference to the filter to
reduce code repetition
IHtmlGeckoOptions options =
doc.HtmlOptions.ForGecko;

options.AddLinks = true;

doc.AddImageUrl("http://www.websupergoo.com");
doc.Save(Server.MapPath("wsg.pdf"));
```

---

Example

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.HtmlOptions.Engine = EngineType.Gecko
theDoc.HtmlOptions.ForGecko.AddLinks = True

' You can store a reference to the filter to
reduce code repetition
Dim theOptions As IHtmlGeckoOptions =
theDoc.HtmlOptions.ForGecko

theOptions.AddLinks = True

theDoc.AddImageUrl("http://www.websupergoo.com")
theDoc.Save(Server.MapPath("wsg.pdf"))
```

# AddForms Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether form fields live.

This property determines whether form fields in HTML are reproduced in the final PDF output.

---

## Notes

Form fields in PDF do not work exactly the same as form fields in HTML while the results will be similar they may not be identical.

The following example shows the effect that this parameter has on the output.

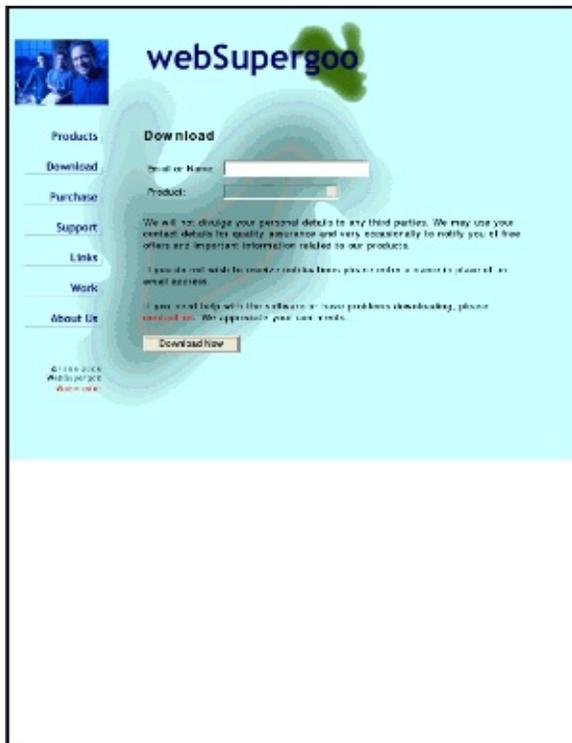
### [C#]

```
Doc theDoc = new Doc();
// Covert html form fields to the pdf form field
output file
theDoc.HtmlOptions.AddForms = true;
int id =
theDoc.AddImageUrl("http://www.websupergoo.com/");
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsAddForms.pdf"));
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
' Covert html form fields to the pdf form fields  
file  
theDoc.HtmlOptions.AddForms = True  
Dim id As Integer =  
theDoc.AddImageUrl("http://www.websupergoo.com/  
' Save the document  
theDoc.Save(Server.MapPath("HtmlOptionsAddForms  
theDoc.Clear()
```

Example



HtmlOptionsAddForms.pdf

# AddLinks Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether hyperlinks should be live.

This property determines whether links in HTML are reproduced as links in the final PDF output.

Links which are not live look exactly like links but do not link through to a destination when you click on them.

---

## Notes

Live links are reproduced exactly as specified on the page and link through to the web pages specified. These web pages will open in a new browser window.

To change external links to internal PDF links see the [Doc.LinkPages](#) method.

---

## Example

See the [Doc.LinkPages](#) method.



# AddMovies Property

---



Type	Default Value	Read Only	Description
[C#]bool			Whether active content such as movies should be added.
[Visual Basic] Boolean	false	No	

Enables the embedding of active content such as Flash (SWF), AVI, MPEG and WMV.

**Flash / SWF Previews.** ABCpdf will automatically create a still preview of Flash (SWF) movies.

The preview is what you will see if you don't have Flash installed. It is what is used when printing PDFs. In general it is what you will see if you open your PDF using a viewer other than Acrobat.

The preview needs to be created at a certain resolution and using content from a particular point in the movie. By default the values are taken from the FlashPreviewTime and FlashPreviewDPI [Registry Keys](#). However you can specify ABCpdf\_PreviewTime and ABCpdf\_PreviewDPI attributes within your EMBED tag to override these defaults.

---

## Notes

Some movies take time to draw themselves because they use script to determine how they should appear. You can use the `ABCpdf_PreviewWaitTime` attribute to determine the time (in milliseconds) that the movie will be given to initialize itself before the preview is made.

For example to take a preview 2000 ms into the movie at 300 dpi you might use the following HTML.

```
<EMBED src="frogger.swf"  
WIDTH="700" HEIGHT="500"  
ABCpdf_PreviewWaitTime="2000"  
ABCpdf_PreviewDPI="300"></EMBED>
```

These attributes are useful if your settings are dependent on the content within the movie rather than on the PDF.

None.

---

## Example

## AddTags Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether location of certain tags should be noted.

This property determines whether certain tags in HTML are noted so that their location can be determined at a later date.

After adding your web pages you can retrieve information about the items on the PDF page using the [GetTagIDs](#) and the [GetTagRects](#) methods.

Note that only HTML elements which have an actual visual representation in the final output will be tagged. For example, a simple `<form>` element may not be tagged because it does not have a visual representation.

All elements of style 'abcpdf-tag-visible: true' will be tagged. For example:

```
<p id="p1" style="abcpdf-tag-visible: true">Gallia est omnis divisa in partes tres.</p>
```

For the **Gecko engine**, having a visual

---

## Notes

representation is not enough. The element must be a display item, which means that there is something to display for the element. Inner contents, such as text and inner elements, are children of the element and are not the element itself. This is important for layout-only elements such as `<p>`, `<div>`, and `<span>`; and style-only elements such as `<em>`, `<strong>`, and `<font>`. Border, outline, etc. are some of the things that make a layout/style-only element a display item:

```
<p id="p1" style="abcpdf-tag-visible:
true; border: 1px solid
transparent">Gallia est omnis divisa
in partes tres.</p>
```

For the **Gecko engine**, you can use the CSS class 'abcpdf-tag-visible' instead of the CSS style 'abcpdf-tag-visible: true' (even if the CSS class is not defined), but it is **deprecated**. See `UseCSSClassToAddTags` in [Registry Keys](#).

```
<p id="p1" class="abcpdf-tag-visible"
style="border: 1px solid
transparent">Gallia est omnis divisa
in partes tres.</p>
```

---

## Example

See the [GetTagRects](#) and the [GetTagUntransformedRects](#) methods.

# AdjustLayout Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether HTML layout is checked and adjusted for optimal output to PDF.

This property determines whether HTML layout is checked and adjusted for optimal output to PDF.

HTML layout is optimized for screen based output on a scrolling display. Certain types of percentage based sizing can give rise to situations in which the proportions of the different elements do not add up to 100%. On screen this type of confused layout is less noticeable than it is on a paged medium like PDF.

---

## Notes

When set the document is scanned for typical layout based errors and these are fixed.

However if you are not using these features then skipping the adjustment stage can save time with no noticeable change in output quality.

None.

---

**Example**

# AutoTruncate Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to automatically clip redundant content at the end of the page.

This property determines whether redundant content at the end of an HTML page is shown or not.

---

## Notes

Redundancy is determined using a number of heuristics. However most commonly it covers repeating background images which don't need to be shown.

---

## Example

None.

# BreakMethod Property

---



Type	Default Value	Read Only	Description
[C#]HtmlBreakMethodType	CumulativeCohesion or GlobalOptimum	No	The break for
[Visual Basic]HtmlBreakMethodType			

This property specifies the page breaking algorithm.

The HtmlBreakMethodType enumeration can take any of the following values:

- CumulativeCohesion  $\tilde{c}^{1/2}$  uses the cumulative cohesion of a objects occupying each horizontal line/vertical position.
- MaximumCohesion  $\tilde{c}^{1/2}$  uses the maximum cohesion among objects occupying each horizontal line/vertical position.

in combination with any of the following values:

- GlobalOptimum  $\tilde{c}^{1/2}$  optimizes the break position globally around a candidate position.
- LocalOptimum  $\tilde{c}^{1/2}$  optimizes the break position locally around a candidate position.
- NoOptimization  $\tilde{c}^{1/2}$  no optimization.

---

## Notes

Each rendered objects in an HTML page has associated cohesion. Text, images, and form controls converted to form fields have high cohesion whereas simple lines and shapes have low cohesion.

Page breaks are selected at positions with low cohesion.

Use MaximumCohesion when you want to break at positions where there are a lot of low-cohesion objects and no high-cohesion objects.

The break position can be optimized according to the cohesion. If it is optimized locally, the cohesion values of positions between the final position and the candidate position is a monotone function.

---

**Example**

None.

# BreakZoneSize Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	75	No	The percentage of the current drawing area in which HTML breaks can occur.

The size of drawing area in which page breaks are allowed.

When HTML is added to a PDF page, it is placed within the current [Doc.Rect](#).

---

## Notes

In general, you do not want page breaks to occur too high up in this area. So ABCpdf only scans for good break locations towards the bottom.

This property determines how far up (in terms of a percentage) from the bottom that ABCpdf will scan for a good break location.

None.

---

## Example



# BrowserWidth Property

---



Type	Default Value	Read Only	Description
[C#] int	0	No	The width of the
[Visual Basic] Integer			

## For MSHTML engine

This property determines or reflects the virtual browser width when rendering HTML documents.

HTML documents do not have a predefined width and height. The width of the document varies as the client resizes the browser. How far the content of the page is dependent on the width of the browser.

The Width parameter is used to control this aspect of HTML rendering. If you were displaying your HTML in a browser window the same width as the value. Typical values might be 640 or 800.

---

### Notes

If the Width is zero then the web page view will default to a size that fits the available content without needing to scroll from left to right. You should ensure that all your HTML content is visible.

## For Gecko engine

This value determines the *paper width*, in *pixels*, to use while doing layout.

Since the Gecko engine always render in a paginated context any contents that horizontally exceed this value will often be *reflowed*. A sidebar done using the CSS `float` property may appear below the main content because the paper is not wide enough to accommodate both items. The paper width will be determined from the current Rect's dimension.

The following example shows the effect that this parameter has on rendering.

### [C#]

```
Doc theDoc = new Doc();
string theURL =
    "http://www.nasa.gov/multimedia/imagegallery/index.html";
// Render html page with default browser width
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsBrowserWidth"));
theDoc.Clear();
// Render html page with browser width = 300
theDoc.HtmlOptions.BrowserWidth = 300;
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsBrowserWidth"));
theDoc.Clear();
```

### [Visual Basic]

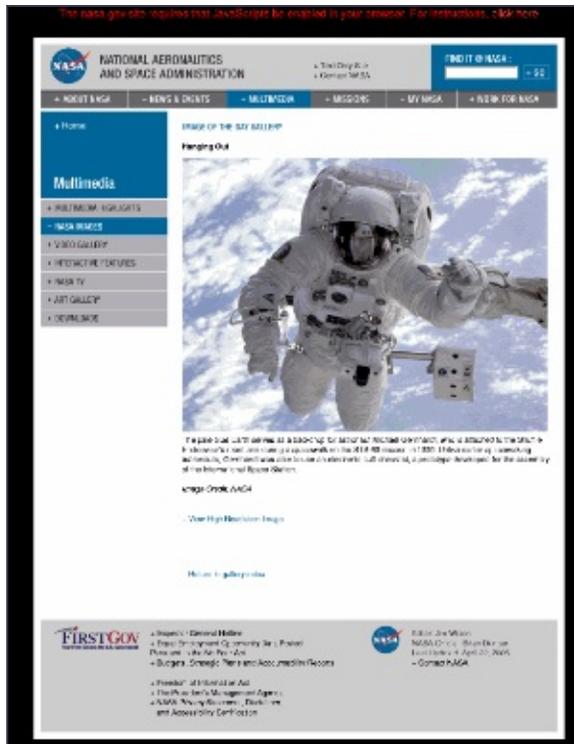
```
Dim theDoc As Doc = New Doc()
Dim theURL As String =
    "http://www.nasa.gov/multimedia/imagegallery/index.html"
' Render html page with default browser width
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsBrowserWidth"))
theDoc.Clear()
```

```

' Render html page with browser width = 300
theDoc.HtmlOptions.BrowserWidth = 300
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsBrowserV
theDoc.Clear()

```

Example



HtmlOptionsBrowserWidth0.pdf

The nasa.gov site requires that JavaScripts be enabled in your browser. For instructions click here



HtmlOptionsBrowserWidth300.pdf

# CoerceVector Property

---



Type	Default Value	Read Only	Description
[C#] HtmlRenderConditions	Default	No	The conditions under which to coerce a vector output.
[Visual Basic] HtmlRenderConditions			

The HtmlRenderConditions enumeration can take a combination of the following values:

- Never – never coerce a vector output.
- Always – always coerce a vector output.
- Default – coerce a vector output under the default conditions. It currently behaves the same as OnFiltersDisabled.
- OnFiltersDisabled – coerce a vector output when some DirectX filter is disabled.
- OnPartialContent – ignored.
- OnVectorCoercionFailed – ignored.

This property determines when to coerce vector outputs if this is not disabled by [DisableVectorCoercion](#). This requires the Microsoft XPS Document Writer printer and hosting a WebBrowser. When the printer is not available or [HostWebBrowser](#) is false, this property is ignored and assumed Never.

When certain constructs, such as DirectX filters or ActiveX controls, are used in the HTML page, the output may become rasterized. When vector-output coercion is activated, a vector output is produced, but the objects that cause the rasterization of the output may not appear in the vector output.

When vector-output coercion is activated, [DeactivateWebBrowser](#) is ignored and assumed false.

---

**Example**

None.

# ContentCount Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	36	No	The minimum number of content items required for a page to be valid.

The minimum number of items a page of HTML should contain.

---

## Notes

If the number is less than this value then the page will be assumed to be invalid. Depending on the [RetryCount](#) settings the page may be re-requested or an error may be returned.

---

## Example

See the [RetryCount](#) property.

# DeactivateWebBrowser Property

---



Type	Default Value	Read Only	Description
[C#]bool			
[Visual Basic] Boolean	true	No	Whether to deactivate the WebBrowser.

This property determines whether the WebBrowser is deactivated before rendering.

By default, if [vector-output coercion](#) is not activated, the WebBrowser is deactivated because an active WebBrowser may produce a rasterized output.

---

## Notes

In some cases, you may want to set this to false and possibly disable vector-output coercion in order to render pages, such as Google Maps, that require the WebBrowser to be active.

None.

---

## Example

# DisableVectorCoercion Property

---



Type	Default Value	Read Only	Description
[C#] HtmlRenderConditions	Default	No	The conditions under which to disable vector-output coercion.
[Visual Basic] HtmlRenderConditions			

- may throw Exception()

The HtmlRenderConditions enumeration can take a combination of the following values:

- Never – never disable vector-output coercion.
- Always – always disable vector-output coercion.
- Default – disable vector-output coercion under the default conditions. It currently behaves the same as OnPartialContent Or OnVectorCoercionFailed.
- OnFiltersDisabled – disable vector-output coercion when some DirectX filter is disabled.
- OnPartialContent – disable vector-output coercion when the HTML page size is bigger than the size supported by vector-output coercion.
- OnVectorCoercionFailed – continue rendering

---

**Notes**

without vector-output coercion when vector-output coercion has failed.

This property determines when to disable vector-output coercion if the coercion is otherwise activated as specified by [CoerceVector](#). There is a size limit to the coerced vector output. If the HTML page size is bigger than the limit, the vector output is clipped so the default behavior is to disable the coercion in this case.

Sometimes, vector coercion fails with unexpected errors. If `OnVectorCoercionFailed` is specified, the page is rendered without vector coercion. Otherwise, an exception is thrown. If the output becomes rasterized, you may want to try rendering the page with [HostWebBrowser](#) false.

---

**Example**

None.

# DoMarkup Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether HTML pages are marked up before conversion to PDF.

This property determines whether HTML pages are marked up before conversion to PDF.

Markup allows common HTML problems to be fixed automatically. It is also necessary for enabling page break CSS tags, for adding fields and for tagged content to be identified.

---

## Notes

However if you are not using these features then skipping markup can save time for some documents.

None.

---

## Example

# FontEmbed Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether fonts should be embedded rather than referenced.

This property determines whether fonts are embedded.

HTML rendering may require that fonts are added to the PDF document.

---

## Notes

By default these fonts are - where possible - referenced. You can use this setting to ensure that fonts are embedded. This will increase output fidelity at a cost of processing time and output file size.

None.

---

## Example

# FontProtection Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether fonts should be protected.

This property determines whether fonts are protected.

HTML rendering may require that fonts are embedded into the PDF document.

---

## Notes

Fonts often contain licensing information. By default ABCpdf will prevent you from embedding fonts which indicate that embedding is not permitted.

You can disable this protection by changing the default for this value.

---

## Example

None.

# FontSubset Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether embedded fonts should be subsetted or not.

This property determines whether embedded fonts are subset.

HTML rendering may require that fonts are added to the PDF document.

By default these fonts are - where possible - referenced. You can use `FontEmbed` to ensure that fonts are embedded. When `FontEmbed` is set to true, you can then use this property to specify if fonts should be subset (only required glyphs are embedded) or not (entire font is embedded). This will considerably increase output size and processing time.

---

## Notes

None.

---

## Example



# FontSubstitute Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether font substitution should be used to reduce file size.

This property determines whether font substitution is enabled.

In many situations default built-in fonts can be substituted in place of the specified fonts.

---

## Notes

This can result in a substantial increase in speed and a considerable reduction in file size with little or no loss in output quality.

None.

---

## Example

# HideBackground Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to hide the background color of a page.

Whether to to hide the background color of an HTML page.

This can be useful for making HTML pages with transparent back

---

## Notes

Note that the background color is not the same as a background image. This property operates on background colors only.

The following example shows the effect that this parameter has on

## [C#]

```
Doc theDoc = new Doc();  
// Please note that the URL below is included for  
demonstration purposes only.  
// In your code you should use your own URL. The  
URL below changes frequently  
// and may include other opaque elements which  
our blue rectangle.  
string theURL = "http://www.usa.gov/";  
// Add some content
```

```
theDoc.Color.String = "0 255 255"; // light blue
theDoc.FillRect(200, 200);
// Hide the background of the HTML page so content
through
theDoc.HtmlOptions.HideBackground = true;
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsHideBack
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
' Please note that the URL below is included for
demonstration purposes only.
' In your code you should use your own URL. The
URL below changes frequently
' and may include other opaque elements which not
blue rectangle.
Dim theURL As String = "http://www.usa.gov/"
' Add some content
theDoc.Color.String = "0 255 255" ' light blue
theDoc.FillRect(200, 200)
' Hide the background of the HTML page so content
through
theDoc.HtmlOptions.HideBackground = True
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsHideBack
theDoc.Clear()
```

---

Example



# HostWebBrowser Property

---

---



Type	Default Value	Read Only	Description
[C#] bool	true (overridable by <a href="#">registry</a> value)	No	Whether to host a WebBrowser control.
[Visual Basic] Boolean			

This property determines whether a WebBrowser control is hosted.

By default, a WebBrowser control is hosted. Script-accessible properties (left, top, width, height, offsetLeft, offsetTop, offsetWidth, offsetHeight, clientLeft, clientTop, clientWidth, clientHeight, pixelLeft, pixelTop, pixelWidth, pixelHeight, posLeft, posTop, posWidth, and posHeight) hold their proper values. The source document can be in any format as long as it is HTML-compatible. For example, XML (with or without XSL applied) can be used because the WebBrowser control transforms it to HTML.

---

## Notes

When no WebBrowser control is hosted, script-accessible dynamic, geometric properties are always zero.

This property affects the [CoerceVector](#) property.

---

Example

None.

# HtmlCallback Property

---



Type	Default Value	Read Only	Description
[C#]HtmlCallback			
[Visual Basic]HtmlCallback	null	No	The multicast delegate HTML rendering is taking place

This delegate is called during HTML rendering.

During the time that HTML rendering is taking place delegates are called. This gives a mechanism for tracking progress and for intercepting queries for pages on the fly.

If the page is obtained from the [cache](#) then the delegate will not be called.

Note that the amount of work done during a callback should be kept to a minimum.

The definition of the HtmlCallback multicast delegate is as follows:

**[C#]**

```
delegate void HtmlCallback(string stage, object page)
```

---

## Notes

**[Visual Basic]**

```
Delegate Sub HtmlCallback(stage As String, page As Page)
```

Current values for the stage variable are 'get width', 'get height' and 'get natural width'. The 'get width' event occurs prior to finding the natural width of the HTML (the width it would have if it were rendered in a browser).

scrolling). The second occurs prior to finding the natural height. 7  
translation into PDF format.

The page is provided via a `mshtml.HTMLDocumentClass` object  
documentation for details. However note that while the interface  
may not be identical. For example element positions may not be

The following example shows the effect that this parameter has (

**[C#]**

```
Doc theDoc = new Doc();
string theURL =
"http://www.nasa.gov/multimedia/imagegallery/in
// Setup the callback
theDoc.HtmlOptions.HtmlCallback = MyHtmlCallback
// Render html page
theDoc.AddImageUrl(theURL);
// Add log over the top of the content
theDoc.Rect.Inset(100, 100);
theDoc.Color.String = "255 0 0";
theDoc.FontSize = 96;
theDoc.AddText(theLog);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsCallback
theDoc.Clear();
```

**[C#]**

```
private static string theLog = "";
public static void MyHtmlCallback(string stage,
{
    theLog += stage + "\n";
}
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theURL As String =
"http://www.nasa.gov/multimedia/imagegallery/in
' Setup the callback
theDoc.HtmlOptions.HtmlCallback = AddressOf MyH
' Render html page
theDoc.AddImageUrl(theURL)
' Add log over the top of the content
theDoc.Rect.Inset(100, 100)
theDoc.Color.String = "255 0 0"
theDoc.FontSize = 96
theDoc.AddText(theLog)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsCallback
theDoc.Clear()
```

---

Example

## [Visual Basic]

```
Private Shared theLog As String = ""
Public Shared Sub MyHtmlCallback(stage As Strin
    theLog = theLog + stage + "\n"
End Sub
```

The nasa.gov site requires that JavaScript be enabled in your browser. For instructions, click here.



**NATIONAL AERONAUTICS  
AND SPACE ADMINISTRATION**

The Day After Tomorrow  
Covers NASA

INDICATOR:

GO

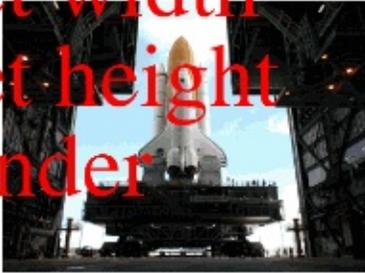
HOME
NEWS & EVENTS
MULTIMEDIA
MISSIONS
BY NASA
FOR NASA

Home

Special of the Day Bulletin

Multimedia

- ▶ MULTIMEDIA HIGHLIGHTS
- ▶ IMAGES
- ▶ VIDEO GALLERY
- ▶ INTERACTIVE
- ▶ NASA TV
- ▶ ART GALLERY
- ▶ DOCUMENTS



The Shuttle Columbia, along with its External Tank and Solid Rocket Boosters, begins the 28th mission since 1981 as it is moved toward the orbiter building at the Kennedy Space Center. The shuttle's flight is scheduled to begin on a mission to the International Space Station, with the shuttle's 28th mission, STS-121, set to launch on Friday, Feb. 27, 2003.

[Return to Top](#)  
[Image Credit: NASA/ASAC](#)  
[View High Resolution Image](#)  
[Home](#) | [Gallery Index](#)

**FIRST GOV**

- ▶ Request General Inquiry
- ▶ Request Information or Inquiry for Product
- ▶ Request for Information on the NASA
- ▶ Request for Information on NASA's

[Privacy Policy](#)  
[Terms of Use](#)  
[Accessibility](#)

**FOIA**

- ▶ Request for Information
- ▶ Request for Information on the NASA
- ▶ Request for Information on NASA's

[Privacy Policy](#)  
[Terms of Use](#)  
[Accessibility](#)

HtmlOptionsCallback.pdf

# HtmlEmbedCallback Property

---



Type	Default Value	Read Only	Description
[C#]HtmlEmbedCallback			
[Visual Basic]HtmlEmbedCallback	null	No	The multicast delegate called while HTML rendering.

This delegate is called when embedding an object.

Objects embedded in HTML, such as those with the embed HTML content types. This callback provides a way to change the parameters.

The definition of the HtmlEmbedCallback multicast delegate is as follows:

## [C#]

```
delegate void HtmlEmbedCallback(Doc doc, HtmlEmbedInfo info);
```

## [Visual Basic]

```
Delegate Sub HtmlEmbedCallback(doc As Doc, info As HtmlEmbedInfo)
```

doc is the target Doc for the added HTML. The properties of HtmlEmbedInfo are:

Property of HtmlEmbedInfo	Type	Default Value
	[C#] string	

Url	[Visual Basic] String	n/a
BaseUrl	[C#] string  [Visual Basic] String	n/a
EmbedType	HtmlEmbedType	See des
ParameterNamesAreCaseSensitive	[C#] bool  [Visual Basic] Boolean	n/a
Parameters	HtmlParameterDictionary	n/a

---

Notes

Parameters depends on EmbedType as follows:

EmbedType	EmbedType Description	ParameterNamesAreCaseSensitive	P
None	No content. This value is not used.		
Swf	SWF files	true if the version of the SWF file is 7 or above.	P a

The values in HtmlParameterDictionary are of type HtmlParamet

Property of HtmlParameter	Type	Default Value	Read Only	D
Value	[C#] string  [Visual Basic] String	n/a	No	T
Conversion	HtmlParameterConversionType	None	No	T

The following example shows the effect that this parameter has on

**[C#]**

```
Doc theDoc = new Doc();
string theURL = "file:///C:/myChart.html";
// Setup the callback
theDoc.HtmlOptions.HtmlEmbedCallback = MyHtmlEmbedCallback;
// Render html page
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsEmbedCallback.html"));
theDoc.Clear();
```

**[C#]**

```
public static void MyHtmlEmbedCallback(Doc doc,
{
    if (info.EmbedType==HtmlEmbedType.Swf) {
        HtmlParameter param;
```

```

        if (info.Parameters.TryGetValue("dataURL",
            info.Parameters.Remove("dataURL"));
            param.Conversion = HtmlParameterConversionType.Xml;
            info.Parameters["dataXML"] = param;
        }
    }
}

```

**Example**

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
Dim theURL As String = "file:///C:/myChart.html"
' Setup the callback
theDoc.HtmlOptions.HtmlEmbedCallback = AddressOf MyHtmlEmbedCallback
' Render html page
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsEmbedCallback.html"))
theDoc.Clear()

```

### [Visual Basic]

```

Public Shared Sub MyHtmlEmbedCallback(doc As Doc, info As DocInfo)
    If info.EmbedType = HtmlEmbedType.Swf Then
        Dim param As HtmlParameter
        If info.Parameters.TryGetValue("dataURL", param) Then
            info.Parameters.Remove("dataURL")
            param.Conversion = HtmlParameterConversionType.Xml
            info.Parameters("dataXML") = param
        End If
    End If
End Sub

```

# HttpAdditionalHeaders Property

---



Type	Default Value	Read Only	Description
[C#]string			Additional HTTP headers to send in the request.
[Visual Basic]String	""	No	

This property specifies HTTP headers added to the default headers. It must follow the syntax of HTTP headers, typically delimited using a cr+lf combination.

For example, you may add the cookies from a response to a request under some authentication (e.g. ASP.NET Forms) where a session ID is returned for subsequent requests so re-authentication is not needed. See the [NoCookie](#) property for further information if you specify cookies in the HTTP headers.

---

## Notes

The headers are sent in only the request for the URL; they are not sent in the requests for the linked resources (e.g. CSS and images) in the obtained HTML page.

The additional headers are always used by the MSHTML [Engine](#). Under Gecko, these headers are only sent if you are using the Screen [Media](#) or if you are using the POST [RequestMethod](#).

The following example shows how this property may be used.

### [C#]

```
Doc doc = new Doc();
string url = ...;
HttpWebRequest request =
(HttpWebRequest)WebRequest.Create(url);
request.CookieContainer = new
CookieContainer(); // required for
HttpWebResponse.Cookies
request.Credentials = ...;
using(WebResponse resp =
request.GetResponse()) {
    // cookieless Forms Authentication adds
authentication ticket to the URL
    url = resp.ResponseUri.AbsoluteUri;
    HttpWebResponse response =
(HttpWebResponse)resp;
    if(response.Cookies.Count>0) { // includes
ASP.NET_SessionId
        bool needsCookie2 = false;
        StringBuilder builder = new
StringBuilder("Cookie: ");
        for(int i = 0; i<response.Cookies.Count;
++i) {
            Cookie cookie = response.Cookies[i];
            if(!needsCookie2 && cookie.Version!=1)
                needsCookie2 = true;
            if(i>0)
                builder.Append("; ");
            builder.Append(cookie.ToString());
        }
        builder.Append(!needsCookie2? "\r\n":
"\r\nCookie2: $Version=1\r\n");
        doc.HtmlOptions.HttpAdditionalHeaders =
builder.ToString();
    }
}
```

```

    }
}
doc.HtmlOptions.NoCookie = true;
doc.HtmlOptions.PageLoadMethod =
PageLoadMethodType.MonikerForHtml;
int id = doc.AddImageUrl(url);
doc.Save(Server.MapPath("HttpHeaders.pdf"));
doc.Clear();

```

### [Visual Basic]

Example

```

Dim doc As Doc = New Doc()
Dim url As String = ...
Dim request As HttpWebRequest =
CType(WebRequest.Create(url),
HttpWebRequest)
request.CookieContainer = New
CookieContainer() ' required for
HttpWebResponse.Cookies
request.Credentials = ...
Using resp As WebResponse =
request.GetResponse()
    ' cookieless Forms Authentication adds
authentication ticket to the URL
    url = resp.ResponseUri.AbsoluteUri
    Dim response As HttpWebResponse =
CType(resp, HttpWebResponse)
    If response.Cookies.Count > 0 Then '
includes ASP.NET_SessionId
        Dim needsCookie2 As Boolean = False
        Dim builder As StringBuilder = New
StringBuilder("Cookie: ")
        For i As Integer = 0 To
response.Cookies.Count - 1
            Dim cookie As Cookie =
response.Cookies(i)
            If Not needsCookie2 AndAlso

```

```

cookie.Version <> 1 Then
    needsCookie2 = True
End If
If i > 0 Then builder.Append("; ")
builder.Append(cookie.ToString())
Next
If Not needsCookie2 Then
    builder.Append(ControlChars.CrLf)
Else
    builder.Append(ControlChars.CrLf &
"Cookie2: $Version=1" & ControlChars.CrLf)
End If
doc.HtmlOptions.HttpAdditionalHeaders =
builder.ToString()
End If
End Using
doc.HtmlOptions.NoCookie = True
doc.HtmlOptions.PageLoadMethod =
PageLoadMethodType.MonikerForHtml
Dim id As Integer = doc.AddImageUrl(url)
doc.Save(Server.MapPath("HttpHeaders.pdf"))
doc.Clear()

```

**ASP.NET\_SessionId.** If your application runs in ASP.NET, you cannot use the cookie from the application's originating request as it identifies the session. If the site containing the target HTML page is different from your site, the session ID is invalid. If it is the same, this will cause a re-entrance in the same session, and ASP.NET may not allow re-entrance for the same session.



# ImageQuality Property

---



Type	Default Value	Read Only	Description
[C#]int [Visual Basic] Integer	101	No	The quality of compression continuous tone images such as JPEGs.

This property determines the image quality acceptable when rendering HTML. ABCpdf uses a high quality lossless compression method for image rendering HTML.

Using this setting you can indicate the quality of compression when rendering continuous tone images such as JPEGs.

This can result in a considerable reduction in file size with little or no loss of quality.

---

## Notes

Note that the MSHTML engine contains optimizations to allow data to be rendered from the web through to the PDF without recompression. However, if you do not indicate that you need this data recompressed, as such, it is not as effective in terms of output size and output quality, to leave the default.

Qualities should range between 0 and 100 (75 is a reasonable value). A value of 100 will result in lossless compression being used in all situations.

The following example shows the effect that this parameter has on the output.

## [C#]

```
Doc theDoc = new Doc();
string theURL =
"http://www.nasa.gov/multimedia/imagegallery/in
// Set low image quality for HTML rendering
theDoc.HtmlOptions.ImageQuality = 5;
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsImageQua
theDoc.Clear());
// Set lossless image quality for HTML renderin
theDoc.HtmlOptions.ImageQuality = 101;
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsImageQua
theDoc.Clear());
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theURL As String =
"http://www.nasa.gov/multimedia/imagegallery/in
' Set low image quality for HTML rendering
theDoc.HtmlOptions.ImageQuality = 5
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsImageQua
theDoc.Clear()
' Set lossless image quality for HTML rendering
theDoc.HtmlOptions.ImageQuality = 101
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsImageQua
theDoc.Clear()
```





# ImprovePageBreakAvoid Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to improve the support for page-break-inside of avoid.

---

## Notes

The Gecko engine does not support some complex uses of CSS style page-break-inside:avoid. In particular, the style applied inside `<table>` does not work well. Set this property to true to improve the layout of elements with the style.

---

## Example

None.

# ImprovePageBreakInTable Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	true	No	Whether to improve the support for page break in table to prevent missing rows.

The Gecko engine has the problem of not laying out the remaining rows in a table if a row reaches exactly the bottom of a page. Set this property to true to resolve the problem.

---

## Notes

None.

---

## Example

# InitialWidth Property

---

---



Type	Default Value	Read Only	Description
[C#] int	800	No	The minimum width to be used for auto-sized pages.
[Visual Basic] Integer			

When the [BrowserWidth](#) is zero ABCpdf will attempt to automatically determine an appropriate size for the page.

To do this it determines the scroll width and scroll height for the page. These are the dimensions which would need to be applied to the page to ensure that all content was displayed without the need to scroll.

---

## Notes

Before these measurements are taken the page needs to be set to a default size. This value determines the width that is used.

This effectively means that the output will always be a minimum of this initial width. If you are working with narrow pages you may wish to reduce this value.

---

None.

**Example**

# LogonName Property

---



Type	Default Value	Read Only	Description
[C#]string			
[Visual Basic]String	""	No	A user name to be used for authentication.

This property determines the authentication user name to be used when accessing secured web sites.

For example you might set this property to "MyServer\Steve" to authenticate as Steve when accessing a particular web site.

This property needs to be used in conjunction with the [LogonPassword](#) property.

ABCpdf is a user like any other user. When it is logged in it stays logged in until the session times out or until you explicitly log it out. So if you wish ABCpdf to log on as a different user you must ensure that it is logged out first.

**ASP.NET Forms Authentication.** The LogonName and LogonPassword properties are related to authentication methods built into HTTP. Methods like Basic (forms) Authentication and Windows Integrated Authentication.

Something like ASP.NET forms based authentication requires a different kind of method because it is under your

programmatic control. Because it is under your programmatic control - ABCpdf cannot authenticate itself. Because Authentication is under your programmatic control - you have to Authenticate ABCpdf.

You will need to allow ABCpdf to pass in a user name and password to your page (probably in encoded form via the URL) and then have that page call the FormsAuthentication.Authenticate code using that username and password.

Some of our clients have encoded this information via a time expiring certificate or token to add extra security to the process.

As an alternative you can obtain the HTML of the current page using the HttpResponse.Filter property or by overriding the Render method of the page. You can then present this HTML to ABCpdf using AddImageHtml. If your HTML references resources using relative references you may wish to insert a <BASE> tag into the HTML before presentation to ABCpdf. Of course any resources you reference would need to be available outside your Authentication scheme.

The following example shows this property may be used.

**[C#]**

```
Doc theDoc = new Doc();
string theURL = "https://www.top-secret-site.co
// Assign name and password
theDoc.HtmlOptions.LogonName = "Steve";
theDoc.HtmlOptions.LogonPassword = "stevepasswo
```

```
// Add HTML page
theDoc.AddImageUrl(theURL);
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsLogon.pc
theDoc.Clear());
```

---

**Example**

**[Visual Basic]**

```
Dim theDoc As Doc = New Doc()
Dim theURL As String = "https://www.top-secret-
site.com"
' Assign name and password
theDoc.HtmlOptions.LogonName = "Steve"
theDoc.HtmlOptions.LogonPassword = "stevepasswo
' Add HTML page
theDoc.AddImageUrl(theURL)
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsLogon.pc
theDoc.Clear())
```

# LogonPassword Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	""	No	A password to be used for authentication.

This property determines the authentication password to be used when accessing secured web sites.

---

## Notes

For example you might set this property to "stevespassword" to authenticate as Steve when accessing a particular web site.

This property needs to be used in conjunction with the [LogonName](#) property.

---

## Example

See the [LogonName](#) property.

# MakeFieldNamesUnique Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether field names should be changed to make them unique.

HTML forms can contain fields with the same name.

If the names of two fields in a PDF are the same, then the fields take the same value.

---

## Notes

So if multiple HTML fields with the same name are added to a PDF, these fields will all appear to contain the same content. This is true even if the original HTML fields contained different content.

Setting this property will result in duplicate fields being renamed to allow the content to be different.

---

## Example

None.

# MaxAtomicImageSize Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	480	No	The maximum size at which an image may be regarded as unbreakable.

ABCpdf generally regards images as atomic - unbreakable over more than one page.

However when an image reaches a certain size it's impossible to stop the image being broken. To try do so would result in unacceptable gaps in the content. So the assumption that the image is atomic has to be disregarded.

---

## Notes

This property reflects the height (in pixels) at which images will be redefined as non-atomic.

None.

---

## Example

# Media Property

---



Type	Default Value	Read Only	Description
[C#] MediaType			
[Visual Basic] MediaType	Print	No	The CSS media type to use.

The MediaType enumeration can take any of the following values:

- Print
- Screen

For the Gecko engine, the Screen media must be used with [UseScript](#) true. An invalid combination (i.e. Screen with UseScript false) yields an exception when you call `AddImageUrl` or `AddImageHtml`.

The same HTML document can achieve different visual representations when rendered on different media by specifying several sets of CSS styles for each medium type. The Gecko engine supports the rendering of Web pages using the print stylesheets and the screen stylesheets.

---

## Notes

When the Screen media is used in conjunction with the Gecko engine, ABCpdf obtains the web page before invoking the engine. Since ABCpdf does not obtain additional headers from the Gecko engine, they are not sent in the request. The kind of headers

that may not be sent include Accept, Accept-Language, Accept-Encoding, Keep-Alive, and Accept-Charset. The User-Agent may be different. In most situations, this is unimportant, but if you are relying on these headers, you can use the [HtmlOptions.HttpAdditionalHeaders](#) property to add your own headers to the request.

# NoCookie Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to disable automatic cookies.

This property determines whether automatic cookies are disabled. If it is true, [PageLoadMethod](#) must be effectively `MonikerForHtml`.

The HTML engine manages cookies as it receives HTTP/HTTPS responses and sends HTTP/HTTPS requests. If this property is set to true, cookies stored in the local cookie database will not be sent in requests, and cookies received in responses will not be stored in the local cookie database.

---

## Notes

Cookies specified in [HttpAdditionalHeaders](#) are not sent in requests if automatic cookies are enabled. Set this property to true if cookies are specified in [HttpAdditionalHeaders](#).

See the [HttpAdditionalHeaders](#) property.

---

## Example



# NoDefaultBackground Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to disable the default background color.

When the HTML page does not specify a background color, the Gecko engine applies the default background color specified by preference `browser.display.background_color` or overridden by preference `browser.display.use_system_colors`. This default color is always opaque.

---

## Notes

Because the default preference specifies white, you get a white background (which you likely do not notice unless there is already some objects on the PDF page) if this property is set to false.

None.

---

## Example

# NoSnapRounding Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to disable the snap rounding of coordinates and lengths.

The Gecko engine snap-rounds measurements at the output resolution. This does not make sense when the output format, such as PDF, has no inherent resolution. The units of measurement of PDF is points, which is too coarse for vector graphics.

---

## Notes

Set this property to true to use the source measurements without snap rounding.

None.

---

## Example

# NoTheme Property

---



Type	Default Value	Read Only	Description
[C#] bool			Whether themes should be disabled.
[Visual Basic] Boolean	false	Yes	

This property determines whether UI theme is not applied to controls.

The theme is only relevant if you are using the MSHTML engine for IE9 or earlier. As such it is likely that this function will be deprecated in future releases of ABCpdf.

---

## Notes

To change the value, use [SetTheme](#).

None.

---

## Example

# OnLoadScript Property

---

---



Type	Default Value	Read Only	Description
[C#] string	""	No	A script to be run after the page is loaded.
[Visual Basic] String			

A client side JavaScript to be applied to a web page before the page is rendered to PDF.

This kind of script can be used for a variety of purposes. For example you might wish to hide background images or get pre-render information from the document.

You can provide a return value by setting an "abcpdf" property in the documentElement. For example:

```
document.documentElement.abcpdf = "my  
return value";
```

The return value can be accessed using the [GetScriptReturn](#) method with the ID returned from [Doc.AddImageUrl](#) or [Doc.AddImageHtml](#).

For the MSHtml engine, you must have the [UseScript](#) key set to true or you will get an "access denied"

---

#### Notes

error; the return value can only be set inside [OnLoadScript](#). The Gecko engine does not have these restrictions.

For the MSHTML engine, the script in this property is executed after the page load is complete, so it may be executed before or after any script in the web page that is executed after the completion of page load.

For the Gecko engine, the script in this property may be executed before or after any event-triggered script in the web page. For example, if you set `window.ABCpdf_go` to false in `OnLoadScript`, but the event listener that sets `window.ABCpdf_go` to true is in the script in the web page, you should check the value of `window.ABCpdf_go` (initially undefined) before setting it to false in `OnLoadScript`.

```
doc.HtmlOptions.OnLoadScript =  
"if(!window.ABCpdf_go)  
window.ABCpdf_go = false;";
```

See the [UseScript](#) property.

---

#### Example

# PageCacheEnabled Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether the page cache should be searched before rendering the page.

ABCpdf holds a cache of recently requested URLs, and it's only after five minutes or so that these pages expire from the cache.

Searching the page cache before rendering a URL results in a considerable degree of optimization for many common operations.

---

## Notes

However, if you wish to bypass the cache, you can do so by setting this property to false. When the cache is disabled, the registry value [MakeURLsUnique](#) takes effect, and the actual URL may be modified.

None.

---

## Example



# PageCacheExpiry Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	300,000	No	The length of time that pages can be held in the cache (ms).

ABCpdf holds a cache of recently requested URLs and it's only after five minutes or so that these pages expire from the cache.

This results in a considerable degree of optimization for many common operations.

---

## Notes

You can modify the length of time that pages are held in the cache using this parameter. This value cannot be reduced to zero.

This value is measured in milliseconds.

None.

---

## Example

# PageCacheSize Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	100	No	The number of pages that can be held in the cache.

ABCpdf holds a cache of recently requested URLs and it's only after five minutes or so that these pages expire from the cache.

This results in a considerable degree of optimization for many common operations.

---

## Notes

You can modify the number of items which can be held in the URL cache using this parameter. This value cannot be reduced to zero.

None.

---

## Example

# Paged Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether content should be rendered in multipaged format.

This property determines whether HTML should be prepared in a way that allows it to be paged over multiple PDF pages.

If paged mode is used then only the first page of the document is drawn. Subsequent pages can be drawn using the [Doc.AddImageToChain](#) method.

---

## Notes

There are subtle differences between paged and non-paged HTML. In general you will want to set this parameter to true even if you only wish to draw one page.

None.

---

## Example

# PageLoadMethod Property

---



Type	Default Value	Read Only	Description
[C#] PageLoadMethodType	Default	No	The method for loading URI/HTML.
[Visual Basic] PageLoadMethodType			

This property specifies the method to load the URI/HTML page.

The PageLoadMethodType enumeration can take any of the following values:

- Default — this is currently the same as MonikerForHtml.
- MonikerForHtml — Moniker is used to load the page.
- WebBrowserNavigate — IWebBrowser2::Navigate is used to load the page. [HostWebBrowser](#) must be true to use this value.

---

## Notes

This property affects the [NoCookie](#) property and the [GetHttpStatusCode](#) method.

---

**Example**

See the [HttpAdditionalHeaders](#) property.

# ProcessOptions Property

---



Type	Default Value	Read Only	Description
[C#] <code>XHtmlProcessOptions</code>	The default <code>XHtmlProcessOptions</code>	Yes	The options for new worker processes.
[Visual Basic] <code>XHtmlProcessOptions</code>			

For each [HTML engine](#) that uses worker processes, you can set the properties in the `XHtmlProcessOptions` object to specify the options for new worker processes.

The worker processes are in a process pool for each HTML engine, which is shared by different instances of `Doc`. However, the options used are not reflected back to the (writable) `XHtmlProcessOptions` properties for different instances of `Doc`.

---

## Notes

The options are used only at the start of the process pool and must not be disposed of before the process pool is stopped. If the process pool has started, the options are ignored.

If the process pool has started, to use a different set of options for subsequent HTML rendering, you will have to stop the process pool by calling `EndTasks` so that all worker processes are terminated. `XHtmlProcessOptions.PoolHasStarted` is false when the process pool is stopped.

---

Example

None.

# ReloadPage Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to reload page.

This property determines whether the page is reloaded from the server. If it is true, [PageLoadMethod](#) must be effectively `MonikerForHtml`.

When a page is to be reloaded, all proxy servers will request the page anew, the page cache is not searched, and if the page cache is enabled, the reloaded page is stored in the page cache.

---

## Notes

Setting this property to true sets `INTERNET_FLAG_RELOAD`. WinINET will bypass the cache (redownloading all entries), (will not send an `If-Modified-Since` or `If-None-Match` request header on these requests (Unconditional request; server cannot return a `HTTP/304`),) and will add a request header, as [UseNoCache](#) does, to help ensure that an intermediary/proxy does not return a previously cached result.

---

Example

None.

# RequestMethod Property

---



Type	Default Value	Read Only	Description
[C#]UrlRequestMethodType			The request method for URL.
[Visual Basic]UrlRequestMethodType	Get	No	

The UrlRequestMethodType enumeration can take any of the following values:

- Get
- Post

This property specifies which method [Doc.AddImageUrl](#) uses to get the HTML page. Normally, Get is the preferred method when requesting the page is an idempotent operation. However, the Get method supports at most 2,048 characters in the URL. If the URL is longer than 2,048 characters and contains parameters, it is necessary to use the Post method. However, the URL minus the parameters (and the question mark) is still limited to 2,048 characters.

---

## Notes

The Post method is used only when the the URL contains a question mark (and this property is set to Post). If the URL does not contain any parameter, the Post method can still be used by appending a question mark to the URL.

When the Post method is used in conjunction with the

Gecko engine, ABCpdf obtains the web page before invoking the engine. Since ABCpdf does not obtain additional headers from the Gecko engine, they are not sent in the request. The kind of headers that may not be sent include Accept, Accept-Language, Accept-Encoding, Keep-Alive, and Accept-Charset. The User-Agent may be different. In most situations, this is unimportant, but if you are relying on these headers, you can use the [HtmlOptions.HttpAdditionalHeaders](#) property to add your own headers to the request.

None.

---

**Example**

# RetryCount Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	5	No	The number of times a page is unavailable or invalid.

This property controls how many times ABCpdf will attempt to obtain HTML rendering may fail one time but succeed the next. This is under the control of ABCpdf.

---

## Notes

So ABCpdf may attempt to re-request a page if it is not immediately available, analogous to clicking on the refresh button of a web browser if the page is unavailable.

See the [ContentCount](#) and the [Timeout](#) properties for how ABCpdf handles a page that is unavailable or invalid.

The following example shows the effect that this parameter has on the rendering of a page.

### [C#]

```
Doc theDoc = new Doc();
string theURL =
"http://www.nasa.gov/multimedia/imagegallery/in
// Set minimum number of items a page of HTML s
// Otherwise the page will be assumed to be inv
```

```

theDoc.HtmlOptions.ContentCount = 20;
// Try to obtain html page 10 times
theDoc.HtmlOptions.RetryCount = 10;
// The page must be obtained in less then 10 se
theDoc.HtmlOptions.Timeout = 10000;
try
{
    theDoc.AddImageUrl(theURL);
}
catch
{
    // Page couldn't be loaded
}
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsRetryCou
theDoc.Clear();

```

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
Dim theURL As String =
"http://www.nasa.gov/multimedia/imagegallery/in
' Set minimum number of items a page of HTML sh
' Otherwise the page will be assumed to be inva
theDoc.HtmlOptions.ContentCount = 20
' Try to obtain html page 10 times
theDoc.HtmlOptions.RetryCount = 10
' The page must be obtained in less then 10 sec
theDoc.HtmlOptions.Timeout = 10000
Try
    theDoc.AddImageUrl(theURL)
Catch
    ' Page couldn't be loaded
End Try
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsRetryCou
theDoc.Clear()

```

Example

The page you are viewing requires that JavaScript be enabled in your browser. For instructions, click here.



**NATIONAL AERONAUTICS  
AND SPACE ADMINISTRATION**

• The Official  
• Official NASA

WELCOME TO NASA

---

- HOME
- NEWS & EVENTS
- MULTIMEDIA
- ISSUES
- MY NASA
- NEWS FOR NASA

---

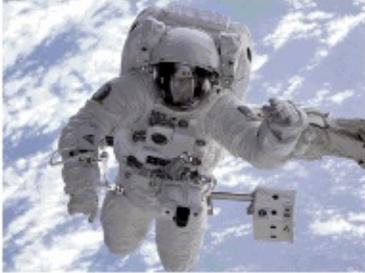
Home

**Multimedia**

- MULTIMEDIA HIGHLIGHTS
- IMAGES
- VIDEO GALLERY
- PHOTOGRAPHY/IMAGES
- NASA TV
- ART GALLERY
- DOCUMENTS

**IMAGE OF THE DAY GALLERY**

**Hanging Out**



The photo shows a **background of a bright, hazy horizon** with a **satellite in the foreground**. The astronaut is **floating in space**, and the **satellite is attached to the station**. The **satellite is a large, rectangular object** with a **small antenna** extending from it. The **satellite is a large, rectangular object** with a **small antenna** extending from it.

**Image Credit: NASA**

[View High Resolution Image](#)

[More Images](#)

---

**FIRST GOV**

- President General Muller
- President General Muller
- President General Muller
- President General Muller

**NASA**

- NASA
- NASA
- NASA
- NASA

**CONTACT**

- Contact
- Contact
- Contact
- Contact

HtmlOptionsRetryCount.pdf

# TargetLinks Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether hyperlinks should be allowed to open new windows.

Whether hyperlinks with targets should be allowed to open in a new browser window.

In HTML, hyperlinks whose targets are neither empty, "\_self", "\_parent", nor "\_top" will be opened in new browser window.

---

## Notes

Setting this property to true allows you to mimic this behavior when the PDF document is viewed inside a browser.

None.

---

## Example

# Timeout Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	15,500	No	The maximum amount of time allowed for obtaining a page (ms).

HTML rendering can take some time.

If the time taken exceeds the Timeout then the page is assumed to be unavailable. Depending on the [RetryCount](#) settings the page may be re-requested or an error may be returned.

---

Notes

This value is measured in milliseconds.

See the [RetryCount](#) property.

---

Example

# TransferModule Property

---



Type	Default Value	Read Only	Description
[C#] TransferModuleType	Default	No	The module for obtaining URI/HTML data.
[Visual Basic] TransferModuleType			

This property specifies the module to obtain URI/HTML data.

The TransferModuleType enumeration can take any of the following values:

- Default — this is currently the same as RetryNetWebRequestOnEngineAccessDenied.
- EngineTransfer — the engine (i.e. MSHTML/WinINet) obtain the page.
- NetWebRequest — System.Net.WebRequest is used to obtain the page.
- RetryNetWebRequestOnEngineAccessDenied — the engine obtains the page. If the engine is denied access to local storage, System.Net.WebRequest is used to obtain the page (Possibly more than one request sent.)

RetryNetWebRequestOnEngineAccessDenied is useful in restricted environment. When MSHTML downloads web pages, it needs to access the following shell folders (specified in "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders" in the registry):

---

**Notes**

- History — the default value is  
%LOCALAPPDATA%\Microsoft\Windows\History
- Cache — the default value is  
"%LOCALAPPDATA%\Microsoft\Windows\Temporary Internet Files"
- Cookies — the default value is  
%APPDATA%\Microsoft\Windows\Cookies

If the folders are not accessible, the fallback folders History, "Temporary Internet Files", and Cookies in %TEMP% are used.

For example, your application uses the "ASP.NET v4.0" application pool and the identity is ApplicationPoolIdentity, you need to grant "IIS APPPOOL\ASP.NET v4.0" access to the folders with

%LOCALAPPDATA% =

%SystemRoot%\System32\config\systemprofile\AppData\Local  
and %APPDATA% =

%SystemRoot%\System32\config\systemprofile\AppData\Roaming  
or to the fallback folders with %TEMP% = %SystemRoot%\Temp

If MSHTML cannot access those folders, ABCpdf with RetryNetWebRequestOnEngineAccessDenied specified will obtain the HTML page using System.Net.WebRequest. In such a case, more than one request may be sent.

None.

---

**Example**

# UseActiveX Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to enable ActiveX.

This property determines whether ActiveX is enabled.

By default ActiveX is disabled when rendering HTML documents.

This is done for good security reasons and we strongly recommend that you do not change this setting.

However if you are sure that your source documents do not pose a security risk you can enable ActiveX using this setting.

---

## Notes

**Compatibility.** Not all ActiveX controls are compatible with ABCpdf. In particular some rely on being able to draw direct to screen which stops ABCpdf being able to use them.

---

**Example**

None.

# UseJava Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to enable Java.

This property determines whether Java is enabled.

By default Java is disabled when rendering HTML documents.

---

## Notes

This is done for good security reasons and we strongly recommend that you do not change this setting.

However if you are sure that your source documents do not pose a security risk you can enable Java using this setting.

---

## Example

None.

# UseNoCache Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether any proxy servers should re-request items of content.

This property determines whether no-cache is enabled.

Setting this property to true forces any intervening proxy server to re-request the page.

For the [MSHTML engine](#), setting this property to true sets `INTERNET_FLAG_PRAGMA_NOCACHE`. WinINET will add request header **Pragma: no-cache** if going through a HTTP/1.0 proxy and request header **Cache-Control: no-cache** if going through a HTTP/1.1 proxy.

For the [Gecko engine](#), setting this property to true sets the following flags.

---

## Notes

- `LOAD_FLAGS_IS_REFRESH` — the load should have the semantics of an HTML Meta-refresh tag (i.e., that the cache should be bypassed).
- `LOAD_FLAGS_BYPASS_HISTORY` — history should not be updated.

- `LOAD_FLAGS_BYPASS_CACHE` — the local web cache should be bypassed (but an intermediate proxy cache could still be used to satisfy the load).
- `LOAD_FLAGS_BYPASS_PROXY` — any intermediate proxy caches should be bypassed (i.e., that the content should be loaded from the origin server).

---

Example

None.

# UseResync Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to resynchronize pages.

This property determines whether resync is enabled.

When a page is resynchronized, the server is requested to send the new version of the page on the condition that the page in the engine's cache is stale.

---

## Notes

Setting this property to true sets INTERNET\_FLAG\_RESYNCHRONIZE. WinINET will send an If-Modified-Since or If-None-Match request header to allow a 304 response, and it *may* add a request header, as [UseNoCache](#) does, to help ensure that an intermediary (proxy) does not return a previously cached result.

---

## Example

None.



# UseScript Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to enable JavaScript and VBScript.

This property determines whether JavaScript and VBScript are enabled in client-side documents.

By default, client-side script such as JavaScript is disabled when rendering HTML documents.

This is done for good security reasons, and we strongly recommend that you do not change this setting.

However, if you are sure that your source documents do not pose a security risk, you can enable Script using this setting.

If you have a server edition of Windows (e.g. Windows Server 2008), you may need to also disable Enhanced Security Configuration for user running the program/application pool to allow script execution.

**Script-Accessible Functions and Properties.** These are functions and properties that script inside HTML can access. The function names are in C#-like syntax. You'll need to pass in the correct number of arguments.

[MSHTML](#) `window.external.ABCpdf_RenderWait` – Delays

only rendering of the HTML page.

---

**Syntax**

```
bool ABCpdf_RenderWait()
```

Name	Description
------	-------------

---

**Params**

return	True if ABCpdf waits (i.e. the function succeeds), otherwise false.
--------	---

---

**Notes**

When this function is called before the page load is complete, the page load is not considered complete until `ABCpdf_RenderComplete` is called. This function returns true if `ABCpdf_RenderComplete` has been called. This is useful if the page relies on timeout/asynchronous events (AJAX).

MSHTML  
only

**window.external.ABCpdf\_RenderComplete** – Resumes rendering of the HTML page.

---

**Syntax**

```
bool ABCpdf_RenderComplete()  
bool ABCpdf_RenderComplete(bool force)
```

Name	Description
------	-------------

---

**Params**

force	Whether ABCpdf ignores normal indications of page load completion. The default value is false.
return	True if ABCpdf resumes rendering as requested (i.e. the function succeeds), otherwise false.

---

**Notes**

This function can be called whether `ABCpdf_RenderComplete` has or has not been called. If `force` is false, ABCpdf resumes rendering only if other indications of page load completion are present. If `force` is true, ABCpdf starts rendering immediately, ignoring other indications of page load completion. This function returns true if `force` is false and the function has been previously called and returned true. This is useful if the page relies on timeout/asynchronous events (AJAX).

---

Notes

Gecko

**window.ABCpdf\_go** – Specifies whether ABCpdf

only

render the HTML page.

---

**Syntax**

```
bool ABCpdf_go
```

---

**Params**

Value	Description
undefined (initial value), true	ABCpdf proceeds to render the HTML page.
false	ABCpdf waits.

---

**Notes**

UseScript has to be true and **OnLoadScript** has to be set to true for this property to be effectual. ABCpdf will wait for the OnLoadScript property to be either undefined or true before rendering the HTML page. HTML rendering operation is still subject to the **Timeout** value. Usually, this property is set to false in OnLoadScript, but can also be set to true in an event listener added in OnLoadScript. Sample assignments to this property are provided in both C# and Visual Basic. For the script in the web page, please refer to the note for the order of execution.

### [C#]

```
Doc doc = new Doc();
doc.HtmlOptions.Engine = EngineType.Chromium;
doc.HtmlOptions.UseScript = true;

// Render after 3 seconds
doc.HtmlOptions.OnLoadScript = "(function() {
window.ABCpdf_go = false; setTimeout(function() {
window.ABCpdf_go = true; }, 3000); })";

doc.AddImageUrl("http://www.websupermarket.com/images/logo.png");
doc.Save(Server.MapPath("wsg.pdf"));
```

---

**Example**

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()

theDoc.HtmlOptions.Engine = EngineType.Chromium
theDoc.HtmlOptions.UseScript = True
```

```

' Render after 3 seconds
theDoc.HtmlOptions.OnLoadScript = "(
window.ABCpdf_go = false; setTimeout
window.ABCpdf_go = true; }, 3000); }

theDoc.AddImageUrl("http://www.websu
theDoc.Save(Server.MapPath("wsg.pdf"

```

The following example shows one method of crawling and transf site to PDF. Here, we use JavaScript to determine the links prese page. However, you could equally well use the [HtmlCallback](#) to c thing.

### [C#]

```

Doc theDoc = new Doc();
string theURL = "http://www.fbi.gov/";
// Set HTML options
theDoc.HtmlOptions.AddLinks = true;
theDoc.HtmlOptions.UseScript = true;
theDoc.HtmlOptions.PageCacheEnabled = false;
// JavaScript is used to extract all links from
theDoc.HtmlOptions.OnLoadScript = "var hrefColl
document.all.tags(\"a\");" +
    "var allLinks = \"\";" +
    "for(i = 0; i < hrefCollection.length; ++i) {"
    "if (i > 0)" +
    "    allLinks += \",\";" +
    "allLinks += hrefCollection.item(i).href;" +
    "};"+
    "document.documentElement.abcpdf = allLinks;"
// Array of links - start with base URL
ArrayList links = new ArrayList();
links.Add(theURL);

```

```

for (int i = 0; i < links.Count; i++) {
    // Stop if we render more than 20 pages
    if (theDoc.PageCount > 20)
        break;
    // Add page
    theDoc.Page = theDoc.AddPage();
    int theID = theDoc.AddImageUrl(links[i] as st
    // Links from the rendered page
    string allLinks =
theDoc.HtmlOptions.GetScriptReturn(theID);
    string[] newLinks = allLinks.Split(new char[]
    foreach (string link in newLinks) {
        // Check to see if we allready rendered thi
        if (links.BinarySearch(link) < 0) {
            // Skip links inside the page
            int pos = link.IndexOf("#");
            if (! ( pos > 0 &&
links.BinarySearch(link.Substring(0, pos)) >= (
                if (link.StartsWith(theURL)) {
                    links.Add(link);
                }
            }
        }
    }
}
// Add other pages
while (true) {
    theDoc.FrameRect();
    if (!theDoc.Chainable(theID))
        break;
    theDoc.Page = theDoc.AddPage();
    theID = theDoc.AddImageToChain(theID);
}
}
// Link pages together
theDoc.HtmlOptions.LinkPages();
// Flatten all pages
for (int i = 1; i <= theDoc.PageCount; i++) {

```

```

    theDoc.PageNumber = i;
    theDoc.Flatten();
}
// Save the document
theDoc.Save(Server.MapPath("HtmlOptionsJavaScript"));
theDoc.Clear();

```

## [Visual Basic]

```

Dim theDoc As Doc = New Doc()
Dim theURL As String = "http://www.fbi.gov/"
' Set HTML options
theDoc.HtmlOptions.AddLinks = True
theDoc.HtmlOptions.UseScript = True
theDoc.HtmlOptions.PageCacheEnabled = False
' JavaScript is used to extract all links from
theDoc.HtmlOptions.OnLoadScript = "var hrefColl
document.all.tags(""a"");" + _
    "var allLinks = """";" + _
    "for(i = 0; i < hrefCollection.length; ++i) {"
    "if (i > 0)" +
    "    allLinks += """, """";" + _
    "allLinks += hrefCollection.item(i).href;" +
    "};" + _
    "document.documentElement.abcpdf = allLinks;"
' Array of links - start with base URL
Dim links As ArrayList = New ArrayList()
links.Add(theURL)
Dim theID As Integer
For i As Integer = 0 To links.Count - 1
    ' Stop if we render more than 20 pages
    If theDoc.PageCount > 20 Then Exit For
    ' Add page
    theDoc.Page = theDoc.AddPage()
    theID = theDoc.AddImageUrl(links(i))
    ' Links from the rendered page
    Dim allLinks As String

```

Example

```

allLinks = theDoc.HtmlOptions.GetScriptReturn
Dim newLinks() As String
newLinks = allLinks.Split(New Char() {","})
Dim link As String
For Each link in newLinks
    ' Check to see if we allready rendered this
    If links.BinarySearch(link) < 0 Then
        ' Skip links inside the page
        Dim pos As Integer
        pos = link.IndexOf("#")
        If Not ( pos > 0 And
links.BinarySearch(link.Substring(0, pos)) >= 0
            If link.StartsWith(theURL) Then links.Add(link)
        End If
    End If
Next
' Add other pages
Do
    theDoc.FrameRect()
    If Not theDoc.Chainable(theID) Then Exit Do
    theDoc.Page = theDoc.AddPage()
    theID = theDoc.AddImageToChain(theID)
Loop
Next
' Link pages together
theDoc.HtmlOptions.LinkPages()
' Flatten all pages
For i As Integer = 1 To theDoc.PageCount
    theDoc.PageNumber = i
    theDoc.Flatten()
Next
' Save the document
theDoc.Save(Server.MapPath("HtmlOptionsJavaScript.js"))
theDoc.Clear()

```



HtmlOptionsJavaScript.pdf - [Page 1]



HtmlOptionsJavaScript.pdf - [Page 2]



HtmlOptionsJavaScript.pdf - [Page 3]



HtmlOptionsJavaScript.pdf - [Page 4]

# UseTheme Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	Yes	Whether to use themes.

This property determines whether UI theme is applied to controls.

The theme is only relevant if you are using the MSHTML engine for IE9 or earlier. As such it is likely that this function will be deprecated in future releases of ABCpdf.

---

## Notes

To change the value, use [SetTheme](#).

None.

---

## Example

# UseVideo Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to enable Video.

This property determines whether Video is enabled.

By default Video is disabled when rendering HTML documents.

---

## Notes

This is done for good security reasons and we strongly recommend that you do not change this setting.

However if you are sure that your source documents do not pose a security risk you can enable Video using this setting.

---

## Example

None.



## StartPool Method

---

Start the process pool for the HTML Engine.

### [C#]

```
bool StartPool()
```

---

### Syntax

### [Visual Basic]

```
Function StartPool() As Boolean
```

---

### Params

Name	Description
return	Whether the process pool is started anew.

This starts the process pool for the HTML Engine (without rendering an HTML page).

---

### Notes

This method returns false if the process pool has been started before the method is called.

---

### Example

None.



# PoolHasStarted Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the process pool for the HTML Engine has started.

This gets whether the process pool for the HTML Engine has started.

The process options are used only at the start of the process pool. If the process pool has started, the options are ignored.

---

## Notes

You can stop the process pool by calling [XHtmlOptions.EndTasks](#) so that all worker processes are terminated.

None.

---

## Example

# ProcessCount Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			The number of existing processes for the HTML Engine.
[Visual Basic] <code>Integer</code>	n/a	Yes	

This gets the number of existing processes for the HTML Engine.

---

## Notes

None.

---

## Example

# Domain Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	null	No	The domain or server containing the user.

This specifies the domain or server containing the user for new worker processes.

---

## Notes

None.

---

## Example

# LoadUserProfile Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to load the user profile.

This specifies whether to load the user profile for new worker processes.

---

## Notes

None.

---

## Example

# Password Property

---



Type	Default Value	Read Only	Description
[C#] SecureString	null	No	The password for the user.
[Visual Basic] SecureString			

This specifies the password for the user for new worker processes.

---

## Notes

The SecureString must not be disposed of while the process pool is not stopped.

---

## Example

None.

# UserName Property

---



Type	Default Value	Read Only	Description
[C#] string	null	No	The user name.
[Visual Basic] String			

This specifies the user name for new worker processes.

---

## Notes

For applications running in a restricted environment, you may need to install [TaskGardener](#) to specify the user.

---

## Example

None.



## FromFile Function

---

Creates an XImage from a file path.

### [C#]

```
static XImage FromFile(string path, XReadOptions options)
```

### [Visual Basic]

```
Shared Function FromFile(path As String, options As XReadOptions) As XImage
```

Syntax

- may throw Exception()

Params

Name	Description
path	The path to the graphic file.
options	The settings for the read. The XImage takes ownership of this parameter. Relevant property for Default ReadModule: <a href="#">PreserveTransparency</a> .
return	The resulting IndirectObject.

The object takes the ownership of the XReadOptions, which is disposed of when the

object is disposed of. You can make the object release the ownership without disposing of the XReadOptions using the parameterized overloads of [Dispose](#) and [Clear](#). The XReadOptions must not be modified as long as the object has the ownership.

For Default [ReadModule](#), this method provides the same functionality as the [SetFile](#) method except that the image holds additional information ([PreserveTransparency](#)). In addition this method allows you to read any document format and treat it as an image. See the [Doc.Read](#) method for details of supported document formats.

For modules that use an [Operation](#) (whether [Operation](#) is actually null or not), if [NeedsFile](#) is true, the file specified must exist as long as the XImage object is not cleared, disposed of, or garbage-collected. The object does not delete the file when it is disposed of.

---

**Notes**

None.

---

**Example**



## FromData Function

---

Creates an XImage from an array of bytes.

### [C#]

```
static XImage FromData(byte[]  
data, XReadOptions options)
```

### [Visual Basic]

```
Shared Function FromData(data()  
As Byte, options As XReadOptions)  
As XImage
```

Syntax

- may throw Exception()

Params

Name	Description
data	The data containing the graphic.
options	The settings for the read. The XImage takes ownership of this parameter. Relevant property for Default <a href="#">ReadModule: PreserveTransparency</a> .
return	The resulting IndirectObject.

The object takes the ownership of the XReadOptions, which is disposed of when the

object is disposed of. You can make the object release the ownership without disposing of the XReadOptions using the parameterized overloads of [Dispose](#) and [Clear](#). The XReadOptions must not be modified as long as the object has the ownership.

---

**Notes**

For Default [ReadModule](#), this method provides the same functionality as the [SetData](#) method except that the image holds additional information ([PreserveTransparency](#)).

For modules that use an [Operation](#) (whether [Operation](#) is actually null or not), the array of bytes must not be modified as long as the XImage object is not cleared, disposed of, or garbage-collected.

---

**Example**

None.



# FromStream Function

---

Creates an XImage from a Stream.

## [C#]

```
static XImage FromStream(Stream  
stream, XReadOptions options)
```

## [Visual Basic]

```
Shared Function FromStream(stream  
As Stream, options As  
XReadOptions) As XImage
```

Syntax

- may throw Exception()

Params

Name	Description
stream	The stream containing the graphic.
options	The settings for the read. The XImage takes ownership of this parameter. Relevant property for Default <a href="#">ReadModule: PreserveTransparency.</a>
return	The resulting IndirectObject.

The object takes the ownership of the XReadOptions, which is disposed of when the

object is disposed of. You can make the object release the ownership without disposing of the XReadOptions using the parameterized overloads of [Dispose](#) and [Clear](#). The XReadOptions must not be modified as long as the object has the ownership.

For Default [ReadModule](#), this method provides the same functionality as the [SetStream](#) method except that the image holds additional information ([PreserveTransparency](#)).

---

**Notes**

For modules that use an [Operation](#) (whether [Operation](#) is actually null or not), if [NeedsStream](#) is true, the object takes the ownership of the Stream, which is disposed of when the object is disposed of. You can make the object release the ownership without disposing of the Stream using the parameterized overloads of [Dispose](#) and [Clear](#). The Stream must not be modified as long as the XImage object is not cleared, disposed of, or garbage-collected.

If [NeedsStream](#) is false, the caller is responsible for closing or disposing of the Stream.

---

**Example**

None.



# Dispose Function

---

Dispose of the object.

## [C#]

```
void Dispose()  
void Dispose(out XReadOptions  
outOptions, out Stream outputStream)  
protected void Dispose(bool  
disposing)
```

---

## Syntax

## [Visual Basic]

```
Sub Dispose()  
Sub Dispose(<Out> ByRef outOption  
As XReadOptions, <Out> ByRef  
outStream As Stream)  
Protected Sub Dispose(disposing  
As Boolean)
```

---

## Params

Name	Description
outOptions	The <a href="#">XReadOptions</a> used to create the object.
outStream	The Stream used to create the object if <a href="#">NeedsStream</a> is true.

You can call this function to explicitly dispose of

an object and reduce the garbage collection overhead.

The overload without parameters disposes of the [XReadOptions](#) and of the Stream.

The overload with output parameters returns the [XReadOptions](#) and the Stream. The returned objects have not been disposed of.

---

#### Notes

This method follows the standard design pattern for objects implementing the `IDisposable` interface. The protected `Dispose` method can be overridden for sub-classes wishing to dispose of additional objects.

Do not attempt to use an object after calling `Dispose`.

None.

---

#### Example



# Clear Function

---

Clears the image.

## [C#]

```
void Clear()  
void Clear(out XReadOptions  
outOptions, out Stream outputStream)
```

## [Visual Basic]

```
Sub Clear()  
Sub Clear(<Out> ByRef outOption  
As XReadOptions, <Out> ByRef  
outStream As Stream)
```

---

### Syntax

Name	Description
outOptions	The <a href="#">XReadOptions</a> used to create the object.
outStream	The Stream used to create the object if <a href="#">NeedsStream</a> is true.

---

### Params

Use this method to release resources and return the image to a just-created state.

The overload without parameters disposes of the [XReadOptions](#) and of the Stream.

---

### Notes

The overload with output parameters returns the `XReadOptions` and the Stream. The returned objects have not been disposed of.

None.

---

**Example**



# SetData Function

---

Load an image from data.

## [C#]

```
void SetData(byte[] data)
```

## [Visual Basic]

### Syntax

```
Sub SetData(data() As Byte)
```

- may throw `Exception()`

### Params

Name	Description
data	The data containing the graphic.

Load an image from data. The data is expected to be provided as an array of bytes.

The data can be any of the following types: JPEG, GIF, TIF, BMP, PNG, PSD, PDB, EXIF, WMF, EMF, EPS, PS or SWF (Flash).

### Notes

Different images within the file can be accessed using the [Frame](#) property. Different portions of the image can be selected using the [Selection](#) property.

Here we read a TIFF file and present the data to the XImage object. We then add the image to our document and then save as PDF.

### [C#]

```
XImage theImg = new XImage();
Doc theDoc = new Doc();
// read the data from a file
string thePath =
Server.MapPath("../mypics/mypic.tif");
FileStream theStream = File.OpenRead(thePath);
byte[] theData = new byte[theStream.Length];
theStream.Read(theData, 0,
(int)theStream.Length);
theStream.Close();
// place the data into the image
theImg.SetData(theData);
theDoc.Rect.Inset(20, 20);
theDoc.AddImageObject(theImg, false);
theImg.Clear();
theDoc.Save(Server.MapPath("imagesetdata.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theImg As XImage = New XImage()
Dim theDoc As Doc = New Doc()
' read the data from a file
Dim thePath As String =
Server.MapPath("../mypics/mypic.tif")
Dim theStream As FileStream =
File.OpenRead(thePath)
Dim theData(theStream.Length) As Byte
theStream.Read(theData, 0,
```

---

Example

```
CType(theStream.Length, Integer))
theStream.Close()
' place the data into the image
theImg.SetData(theData)
theDoc.Rect.Inset(20, 20)
theDoc.AddImageObject(theImg, False)
theImg.Clear()
theDoc.Save(Server.MapPath("imagesetdata.pc
theDoc.Clear()
```



imagesetfile.pdf



# SetFile Function

---

Load an image from a file.

## [C#]

```
void SetFile(string path)
```

## [Visual Basic]

Syntax

```
Sub SetFile(path As String)
```

- may throw Exception()

Params

Name	Description
path	The path to the graphic file.

Load an image from file.

The file can be any of the following types: JPEG, GIF, TIFF, PNG, PSD, PDB, EXIF, WMF, EMF, EPS, PS or SWF (Flash)

Notes

Different images within the file can be accessed using the **F** property. Different portions of the image can be selected using the **Selection** property.

Here we open a TIFF file using the XImage object. After we file we add the image to our document and then save the P

### [C#]

```
XImage theImg = new XImage();  
Doc theDoc = new Doc();  
theImg.SetFile(Server.MapPath("../mypics/my  
theDoc.Rect.Inset(20, 20);  
theDoc.AddImageObject(theImg, false);  
theImg.Clear();  
theDoc.Save(Server.MapPath("imagesetfile.pc  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theImg As XImage = New XImage()  
Dim theDoc As Doc = New Doc()  
theImg.SetFile(Server.MapPath("../mypics/my  
theDoc.Rect.Inset(20, 20)  
theDoc.AddImageObject(theImg, False)  
theImg.Clear()  
theDoc.Save(Server.MapPath("imagesetfile.pc  
theDoc.Clear()
```

Example



imagesetfile.pdf





## SetMask Function

---

Assign a soft mask to the image.

### [C#]

```
void SetMask(XImage mask, bool invert)
```

### [Visual Basic]

Syntax

```
Sub SetMask(mask As XImage, invert As Boolean)
```

- may throw Exception()

Params

Name	Description
mask	The image containing the soft mask.
invert	Whether to invert the mask.

Assign a soft mask or alpha channel. Soft masks are used to assign different levels of transparency to an image.

The mask provided will be converted to a grayscale intensity. The mask may be scaled to ensure that the dimensions of the mask match the dimensions of the image. The assigned mask applies to all frames of the image.

Notes

If the Invert parameter is set then the mask will be inverted. Transparent areas become opaque and opaque areas become transparent.

Different images within the mask can be accessed using the `Image` property. Different portions of the mask can be selected using the `Selection` property.

Note that transparency is only applied to an Image if the `IncludeTransparency` property is true (which is generally the case).

Here we read a TIFF file and present the data to the Image, read a mask image and assign that to our Image. Finally we add the Image to our document and then save the PDF.

### [C#]

```
Doc theDoc = new Doc();
XImage theImg = new XImage();
XImage theMsk = new XImage();
theImg.SetFile(Server.MapPath("../mypics/myimg.tif"));
theMsk.SetFile(Server.MapPath("../mypics/mymask.tif"));
theImg.SetMask(theMsk, true);
theMsk.Clear();
theDoc.Color.String = "0 0 0";
theDoc.FillRect();
theDoc.Rect.Inset(20, 20);
theDoc.AddImageObject(theImg, true);
theImg.Clear();
theDoc.Save(Server.MapPath("imagesetmask.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theImg As XImage = New XImage()
Dim theMsk As XImage = New XImage()
theImg.SetFile(Server.MapPath("../mypics/myimg.tif"))
theMsk.SetFile(Server.MapPath("../mypics/mymask.tif"))
```

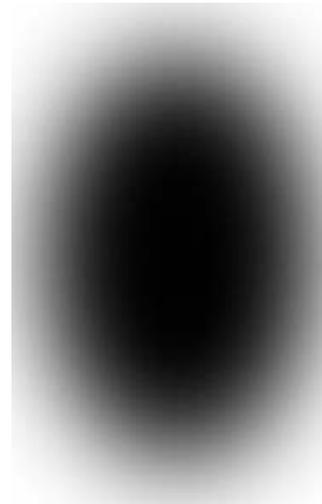
```
theImg.SetMask(theMsk, True)
theMsk.Clear()
theDoc.Color.String = "0 0 0"
theDoc.FillRect()
theDoc.Rect.Inset(20, 20)
theDoc.AddImageObject(theImg, True)
theImg.Clear()
theDoc.Save(Server.MapPath("imagesetmask.pc
theDoc.Clear()
```

Given the following input images.

**Example**



mypic.tif



mymask.jpg

This is the kind of output you might expect.



imagesetmask.pdf



## SetStream Function

---

Load an image from stream.

### [C#]

```
void SetStream(Stream stream)
```

### [Visual Basic]

Syntax

```
Sub SetStream(stream As Stream)
```

- may throw `Exception()`

Params

Name	Description
Stream	The stream containing the graphic.

Load an image from stream.

The stream can contain any of the following types: JPEG, GIF, TIFF, BMP, PNG, PSD, PDB, EXIF, WMF, EMF, EPS, PS or SWF (Flash).

Notes

Different images within the file can be accessed using the [Frame](#) property. Different portions of the image can be selected using the [Selection](#) property.

Here we read a TIFF file and present the data to the XImage  
We then add the image to our document and then save the

### [C#]

```
XImage theImg = new XImage();  
string thePath;  
thePath = Server.MapPath("../mypics/mypic.t  
Stream theStream;  
theStream = File.OpenRead(thePath);  
theImg.SetStream(theStream);  
theStream.Close();  
Doc theDoc = new Doc();  
theDoc.Rect.Inset(20, 20);  
theDoc.AddImageObject(theImg, false);  
theImg.Clear();  
theDoc.Save(Server.MapPath("imagesetstream.  
theDoc.Clear());
```

### [Visual Basic]

```
Dim theImg As XImage = New XImage()  
Dim thePath As String  
thePath = Server.MapPath("../mypics/mypic.t  
Dim theStream As Stream  
theStream = File.OpenRead(thePath)  
theImg.SetStream(theStream)  
theStream.Close()  
Dim theDoc As Doc = New Doc()  
theDoc.Rect.Inset(20, 20)  
theDoc.AddImageObject(theImg, False)  
theImg.Clear()  
theDoc.Save(Server.MapPath("imagesetstream.  
theDoc.Clear()
```

---

Example



imagesetstream.pdf

# BoundingBox Property

---



Type	Default	Read Only	Description
[C#] XRect	The bounds of the current frame.	Yes	The physical bounds of the image in points.
[Visual Basic] XRect			

This property reflects the physical bounds of the image.

It is more relevant for EPS and PS files which operate in terms of physical dimensions rather than pixels.

---

## Notes

Note that EPS and PS files may have bounds which are not anchored at the origin.

None.

---

## Example

# Frame Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	1	No	The currently selected

Some file formats can contain more than one image. The [Frame](#) property reflects the number of images.

You can change the currently selected image using the [Frame](#) property. As you change the frame the [Width](#), [Height](#), [HRes](#) and [VRes](#) properties will change to reflect the dimensions and resolution of the currently selected image. When you add an Image using the [Doc.AddImage](#) method the currently selected frame is added.

---

## Notes

Flash (SWF) movies contain a number of frames. You can set the frame using this property. If you set this property to a negative number it indicates the number of milliseconds (rather than frames) into the

Here we open a TIFF file using the `XImage` object. We then scan for the images within the file and insert them into a new page of the document.

### [C#]

```
XImage theImg = new XImage();
Doc theDoc = new Doc();
theImg.SetFile(Server.MapPath("../mypics/multi")
```





# FrameCount Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	See description.	Yes	The number of frames in the image.

TIFF and GIF files may contain more than one image. The FrameCount property reflects the number of images found in the file. You can select different images using the [Frame](#) property.

Multi-frame TIFF images are generally used for one of two purposes. Sometimes multiple copies of the same image are embedded at different sizes to allow previews to be obtained quickly and easily. This type of multi-frame image is generally created for print purposes. Sometimes multiple pages of a document may be embedded in a TIFF. This type of multi-frame image is generally created by document scanning software or fax software.

---

## Notes

Multi-frame GIF images are often called animated GIFs. Each image represents a frame of the animation. Similarly Flash / SWF movies are frame based and this property allows you to access the total number of frames in the movie.

PostScript is a programming language. So a PostScript program may output a different number of

pages each time it is run. For this reason it is not possible to reliably determine the number of pages in a PostScript file until it is actually displayed. As a result EPS and PS files always have a FrameCount of one no matter how many pages they may display. However it is reasonably safe to assume that EPS files only ever contain one page of content.

---

**Example**

See the [Frame](#) property.

# FrameRate Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	See description.	Yes	The default frame rate for a moving image.

Some formats like Flash consist of a sequence of frames to be played one after the other.

This property tells you the number of frames per second that should be played. Note that this is an indicative property - it indicates a desired value. If movies are complicated then some devices on which they are played may not be able to achieve the desired frame rates.

---

## Notes

See the [Frame](#) property.

---

## Example

# HasRealRes Property

---



Type	Default	Read Only	Description
[C#]bool			
[Visual Basic] Boolean	See description.	Yes	Whether the image specifies the resolution.

This property indicates whether [HRes](#) and [VRes](#) are obtained from the image. [HRes](#) and [VRes](#) contain default values if they are not obtained from the image.

---

## Notes

None.

---

## Example

# Height Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	See description.	Yes	The height of the current frame (pixels).

This property reflects the height of the current frame. It is measured in pixels.

---

## Notes

None.

---

## Example

# HRes Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	See description.	Yes	The horizontal resolution of the current frame (DPI).

This property reflects the horizontal resolution of the current frame. It is measured in dots per inch (DPI).

---

## Notes

None.

---

## Example

# Indirect Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	See description.	Yes	Whether the image will be added using indirect mode.

Images can be added using indirect mode or pass-through mode.

You can find details of these modes in the [Image Handling](#) section of the documentation.

When you add an Image using the [Doc.AddImageObject](#) method it will generally be added using indirect mode.

---

## Notes

However some file types do not support indirect mode and can only be added using pass-through mode. If this is the case the operation of the [Doc.AddImageObject](#) method becomes identical to the [Doc.AddImageFile](#) or [Doc.AddImageData](#) methods.

The value of this property tells you if the image will be added using indirect mode. It is true for all image types with the exception of EPS and PS.

---

**Example**

None.

# NeedsFile Property

---



Type	Default	Read Only	Description
[C#]bool [Visual Basic] Boolean	See description.	Yes	Whether the file needs to exist.

This property is true if the object has been created with an [XReadOptions](#) with [ReadModule](#) that uses an [Operation](#) (whether [Operation](#) is actually null or not) and the file specified is required to exist as long as the object is not cleared, disposed of, or garbage-collected. The object does not delete the file when it is disposed of.

---

## Notes

None.

---

## Example

# NeedsStream Property

---



Type	Default	Read Only	Description
[C#]bool			
[Visual Basic] Boolean	See description.	Yes	Whether the stream needs be kept open.

This property is true if the object has been created with an [XReadOptions](#) with [ReadModule](#) that uses an [Operation](#) (whether [Operation](#) is actually null or not) and the stream provided is required to be kept open and unmodified as long as the object is not cleared, disposed of, or garbage-collected. If it is true, the object has taken the ownership of the Stream, which is disposed of when the object is disposed of. You can make the object release the ownership without disposing of the Stream using the parameterized overloads of [Dispose](#) and [Clear](#)

---

## Notes

None.

---

## Example

# Selection Property

---



Type	Default	Read Only	Description
[C#] XRect [Visual Basic] XRect	The dimensions of the current frame.	No	The current selection rectangle.

You may wish to add only a portion of an image. By changing the selection rectangle you can specify different areas to be added.

---

## Notes

Note that this property only has an effect on an Image if the [Indir](#) property is true (which is generally the case).

Here we open a TIFF file using the XImage object. We add the image to the document and then just a portion of the image using Selection property.

## [C#]

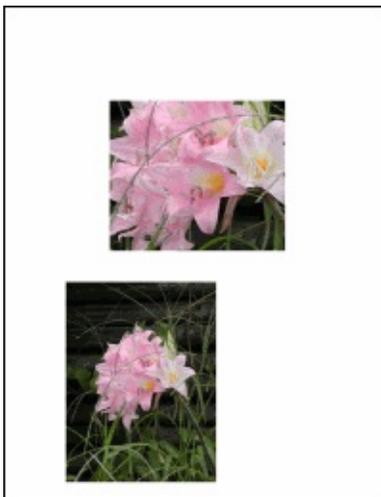
```
XImage theImg = new XImage();
Doc theDoc = new Doc();
theImg.SetFile(Server.MapPath("../mypics/mypic.
theDoc.Rect.String = theImg.Selection.String;
theDoc.Rect.Magnify(0.5, 0.5);
theDoc.Rect.Position(100, 30);
theDoc.AddImageObject(theImg, false);
theImg.Selection.Inset(100, 200);
```

```
theDoc.Rect.String = theImg.Selection.String;
theDoc.Rect.Position(170, 400);
theDoc.AddImageObject(theImg, false);
theImg.Clear();
theDoc.Save(Server.MapPath("imageselect.pdf"));
theDoc.Clear();
```

### [Visual Basic]

Example

```
Dim theImg As XImage = New XImage()
Dim theDoc As Doc = New Doc()
theImg.SetFile(Server.MapPath("../mypics/mypic.
theDoc.Rect.String = theImg.Selection.String
theDoc.Rect.Magnify(0.5, 0.5)
theDoc.Rect.Position(100, 30)
theDoc.AddImageObject(theImg, False)
theImg.Selection.Inset(100, 200)
theDoc.Rect.String = theImg.Selection.String
theDoc.Rect.Position(170, 400)
theDoc.AddImageObject(theImg, False)
theImg.Clear()
theDoc.Save(Server.MapPath("imageselect.pdf"))
theDoc.Clear()
```



imageselect.pdf



# Type Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	See description.	Yes	The type of image.

The type property reflects the type of image that has been loaded using [SetFile](#) or [SetData](#). It can take one of the following values.

- ""
- "BMP"
- "EMF"
- "EXIF"
- "GIF"
- "Icon"
- "JPEG"
- "MemoryBMP"
- "PNG"
- "TIFF"
- "WMF"
- "EPS"
- "PS"
- "SWF"

---

## Notes

None.

---

**Example**

# VRes Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	See description.	Yes	The vertical resolution of the current frame (DPI).

This property reflects the vertical resolution of the current frame. It is measured in dots per inch (DPI).

---

Notes

None.

---

Example

# Width Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	See description.	Yes	The width of the current frame (pixels).

This property reflects the width of the current frame. It is measured in pixels.

---

## Notes

None.

---

## Example



# XPoint Constructor

---

XPoint Constructor.

## [C#]

```
XPoint()  
XPoint(string value)  
XPoint(double x, double y)
```

---

## Syntax

## [Visual Basic]

```
Sub New  
Sub New(value As String)  
Sub New(x As Double, y as Double)
```

---

## Params

Name	Description
value	A string defining the initial point in the format "x y".
x	The x coordinate for the point.
y	The y coordinate for the point.

These methods construct an XPoint object.

---

## Notes

---

Example

None.

# Copy Function

---



Copies a series of X and Y coordinates between arrays of doubles and arrays of XPoints

## [C#]

```
void Copy(XPoint[] source,
double[] destination, int length)
void Copy(XPoint[] source, int
sourceIndex, double[]
destination, int
destinationIndex, int length)
void Copy(double[] source,
XPoint[] destination, int length)
void Copy(double[] source, int
sourceIndex, XPoint[]
destination, int
destinationIndex, int length)
```

## [Visual Basic]

Syntax

```
Sub Copy(source() As XPoint,
destination() As Double, length
As Integer)
Sub Copy(source() As XPoint,
sourceIndex As Integer,
destination() As Double,
destinationIndex As Integer,
length As Integer)
Sub Copy(source() As Double,
destination() As XPoint, length
As Integer)
Sub Copy(source() As Double,
```

```
sourceIndex As Integer,  
destination() As XPoint,  
destinationIndex As Integer,  
length As Integer)
```

---

**Params**

Name	Description
source	An array of points to be copied from.
destination	An array of points to be copied to.
length	The number of points that should be copied.
sourceIndex	The index in the source array at which copying begins.
destinationIndex	The index in the destination array at which copying begins.

Copies a series of X and Y coordinates between arrays of doubles and arrays of XPoints.

The array of doubles is represented as a series of pairs of X and Y coordinates. So the double array should always be twice the length of the XPoint array. Indexes into the arrays are specified in terms of items in the array. This means that indexes into a double array will be twice as large as those into an XPoint array.

---

**Notes**

Lengths are always specified in terms of numbers of points rather than number of items

in the array.

In the case of copying to an array of XPoints, each XPoint in the destination array will be newly created from the source array values overwriting any XPoint which may already be at that location in the array.

---

**Example**

None.

# Equals Function

---



Test whether the two points are effectively the same

## [C#]

```
bool Equals(XPoint other, double epsilon)
bool Equals(XPoint other)
override bool Equals(object other)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As XPoint, epsilon As Double) As Boolean
Function Equals(other As XPoint) As Boolean
Overrides Function Equals(other As Object) As Boolean
```

---

## Params

Name	Description
other	The point to test against.
epsilon	The largest difference in values which will still be defined as equal
return	Whether the two points are the same.

Test whether the two points are effectively the same.

Points are considered equal if they have the same x and y coordinates. This represents value equality for the points in question.

The underlying components of an XPoint are represented as floating point numbers. Floating point numbers are subject to rounding errors, so there has to be a degree of latitude when comparing coordinate values. The degree of latitude is, by default, determined by the limitations defined in the PDF Specification. This is, broadly speaking, 5 decimal points so the default epsilon is typically 0.00001. However if you wish to rely on a specific epsilon value you should provide one.

---

**Notes**

None.

---

**Example**



# GetHashCode Function

---

A hash code for the XPoint

## [C#]

```
override int GetHashCode()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function GetHashCode()  
As Integer
```

---

### Params

Name	Description
return	The returned hash code.

---

### Notes

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

---

### Example

None.



# SetPoint Function

---

Sets the point.

## [C#]

```
void SetPoint(XPoint point)
```

---

### Syntax

## [Visual Basic]

```
Sub SetPoint(point As XPoint)
```

---

### Params

Name	Description
point	The source point.

---

### Notes

This method copies the value from the parameter.

---

### Example

None.



# ToString Function

---

Returns a string representation of the object.

## [C#]

```
override string ToString()
```

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Syntax

Name	Description
return	The string representation of the object.

---

### Params

This method returns the string value of the object. This is equivalent to reading the [String](#) property of the object.

---

### Notes

None.

---

### Example



# Point Property

---



Type	Default	Read Only	Description
[C#] <a href="#">Point</a>			
[Visual Basic] <a href="#">Point</a>	n/a	No	The System.Drawing.Point.

The point as a System.Drawing Point.

Windows coordinates are measured in distances from the top left of the drawing surface while PDF coordinates are measured from the bottom left.

So when you use this property the coordinates must be re-mapped. This needs to be done in the context of a containing object. Properties such as the [Doc.Pos](#) are interpreted in the context of the [Doc.MediaBox](#). If there is no containing object then no re-mapping can be performed.

---

## Notes

You may find it easier to work with .NET Points than PDF points. However remember that operations such as [Transforms](#) work on the underlying PDF coordinates and not on the abstracted Windows coordinates.

The following code adds three words to a document. The

positioning is done using standard .NET Points.

### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
Point pt = theDoc.Pos.Point;
pt.Offset(100, 150);
theDoc.Pos.Point = pt;
theDoc.AddText("One");
pt.Offset(100, 150);
theDoc.Pos.Point = pt;
theDoc.AddText("Two");
pt.Offset(100, 150);
theDoc.Pos.Point = pt;
theDoc.AddText("Three");
theDoc.Save(Server.MapPath("xptpt.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
Dim pt As Point = theDoc.Pos.Point
pt.Offset(100, 150)
theDoc.Pos.Point = pt
theDoc.AddText("One")
pt.Offset(100, 150)
theDoc.Pos.Point = pt
theDoc.AddText("Two")
pt.Offset(100, 150)
theDoc.Pos.Point = pt
theDoc.AddText("Three")
theDoc.Save(Server.MapPath("xptpt.pdf"))
theDoc.Clear()
```

---

Example

One

Two

Three

xptpt.pdf

# String Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	"0 0"	No	The point as a string

Allows you to access to the point as a string.

---

## Notes

The format of the string must be "x y".

The following code.

### [C#]

```
XPoint pt = new XPoint();
pt.String = "20 10";
Response.Write("X = "
+pt.X.ToString());
Response.Write("<br>");
Response.Write("Y = "
+pt.Y.ToString());
```

### [Visual Basic]

```
Dim pt As XPoint = New XPoint()
pt.String = "20 10"
```

---

## Example

```
Response.Write("X = "
+pt.X.ToString())
Response.Write("<br>")
Response.Write("Y = "
+pt.Y.ToString())
```

Produces the following output.

```
X = 20
Y = 10
```

# X Property

---



Type	Default	Read Only	Description
[C#]double			
[Visual Basic] Double	0	No	The horizontal coordinate

Allows you access to the horizontal offset of the point.

---

## Notes

Distances are measured from the left of the document.

---

## Example

None.

# Y Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The vertical coordinate

Allows you access to the vertical offset of the point.

---

## Notes

Distances are measured from the bottom of the document.

---

## Example

None.



# XReadOptions Constructor

---

XReadOptions Constructor.

## [C#]

```
XReadOptions()
```

---

### Syntax

## [Visual Basic]

```
Sub New()
```

---

### Params

Name	Description
none	

---

### Notes

Create an empty XReadOptions.

---

### Example

None.

# ClearCache Function

---



Clear cached data and terminate worker threads and worker processes for a read module.

## [C#]

```
bool ClearCache()
```

---

## Syntax

## [Visual Basic]

```
Function ClearCache() As Boolean
```

---

## Params

Name	Description
return	Whether it has taken an action.

This method returns true for the following value of [ReadModule](#).

---

## Notes

- MSOffice — terminate MS Office application processes.

---

None.

## Example



# Dispose Function

---

Dispose of the object.

## [C#]

```
void Dispose()
```

---

### Syntax

## [Visual Basic]

```
Sub Dispose()
```

---

### Params

Name	Description
none	

You can call this method to explicitly dispose of an object and reduce the garbage collection overhead.

---

### Notes

This method disposes of the [Operation](#).

Do not attempt to use an object after calling Dispose.

---

None.

## Example

# Equals Function

---



Test whether the two XReadOptions are effectively the same.

## [C#]

```
bool Equals(XReadOptions other)
override bool Equals(object
other)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As
XReadOptions) As Boolean
Overrides Function Equals(other
As Object) As Boolean
```

---

## Params

Name	Description
other	The object to test against.
return	Whether the two XReadOptions are the same.

---

## Notes

Test whether the two XReadOptions are the same.

---

**Example**

None.

# GetHashCode Function

---



A hash code for the XReadOptions

## [C#]

```
override int GetHashCode()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function GetHashCode() As Integer
```

---

### Params

Name	Description
return	The returned hash code.

---

### Notes

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

---

### Example

None.



# ReadModule Property

---



Type	Default	Read Only	Description
[C#]ReadModuleType			
[Visual Basic]ReadModuleType	Default	No	Gets or sets the module to use.

The ReadModuleType enumeration may take the following values:

- Default
- Pdf
- SwfVector
- Xps
- XpsAny
- MSOffice
- OpenOffice
- Svg
- Eps
- BasicImage
- Tiff
- Photoshop
- WordGlue
- RichTextFormat

The Default value allows ABCpdf to delegate the read operation to what it considers to be an appropriate module or set of modules, depending on the content provided and the method called.

The Pdf value is the standard PDF import module. It takes advantage of the [Password](#) property.

The SwfVector module is a native Flash / SWF vector import module. It takes advantage of the [Operation](#) and [Frame](#) properties. The Operation must be either null or a [SwfImportOperation](#).

The Xps module is a native XPS and OXPS (Open XPS) import module. It takes advantage of the [Operation](#) property, which must be either null or an [XpsImportOperation](#).

The XpsAny module prints via the XPS printer driver and then imports the resultant document. It takes advantage of the [Operation](#), [FileExtension](#), and [Timeout](#) properties. The Operation provided must be either null or an XpsImportOperation.

The MSOffice module uses Microsoft Office for document conversion. This module is fast and produces high fidelity output, including native form fields and annotations conversion. It takes advantage of the [AddForms](#), [Boomarks](#), [Password](#), [EnableMSOfficeMacros](#), [PreserveTransparency](#) and [Timeout](#) properties.

The OpenOffice module uses OpenOffice.org for document conversion. It takes advantage of the [OpenOfficeParameters](#) and [Timeout](#) properties.

The WordGlue module uses WordGlue .NET for DOC and DOCX conversion. WordGlue .NET is a fully managed component for the conversion of semantic document formats. This requires WordGlue 2.0.0.1 or later to be installed. WordGlue can be downloaded from our web site.

---

Notes

The RichTextFormat module is a native RTF import module. It takes advantage of the [DefaultRect](#) and

[Timeout](#) properties.

The Svg module is a native SVG (Scaleable Vector Graphics) import module. It takes advantage of the [DefaultFont](#) and [DefaultRect](#) properties.

The Eps module is a native EPS (Encapsulated PostScript) import module. It takes advantage of the [ErrorHandling](#) and [Log](#) properties.

The Tiff module is a native TIFF (Tagged Image File Format) import module. It takes advantage of the [PreserveTransparency](#) and [Frame](#) properties. TIFF images may be black and white, grayscale, RGB, CMYK, TIFF or Lab in 1, 8, 16 and 32 bits per component color depth and with or without alpha. Because PDF does not support 32 bit High Dynamic Range (HDR) encodings, TIFFs in this format will be downsampled to 16 bits per component. TIFF images containing JPEG or CCITT compressed frames will be inserted direct into the document without a decompress-recompress cycle. This module is broadly similar in speed to System.Drawing for most images. For CMYK images and images with large embedded color profiles it is a bit slower simply because there is more data to compress. In our tests with large images in the order of about 500 MB it was about ten times faster. In our tests with TIFF images containing JPEG or CCITT frames it is typically about thirty times faster.

The Photoshop module supports the standard PSD and also the large image PDB file types. If the [PreserveTransparency](#) property is set any transparency will be preserved. It allows the direct import of bitmap, RGB, Grayscale, CMYK, Lab, Indexed and Duotone images in 1, 8, 16 and 32 bits per component color depth, all with or without alpha. The PDF format does not support 32 bits per

component HDR images so these are scaled down to 16 bits per component. The [Frame](#) property allows the extraction of individual layers from within the image. TIFF orientation flags are automatically applied so that images that are tagged in this way automatically appear the right way up.

The BasicImage module allows you to read images such as JPEG and multi-page TIFF directly into a PDF document. It takes advantage of the [PreserveTransparency](#) property.

None.

---

[Example](#)

# AddForms Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether form fields should be live.

This property determines whether form fields in source documents are converted into PDF form fields.

---

Notes

# Bookmarks Property

---



Type	Default	Read Only	Description
[C#] BookmarkType	DontCare	No	Bookmark creation strategy.
[Visual Basic] BookmarkType			

The BookmarkType enumeration may take any of the following values:

- DontCare – specifies that module-specific default settings should be used.
- None – specifies that no bookmarks should be imported from source documents.
- Heading – specifies that bookmarks should be created using document headings. For example, styled headings in Microsoft Word might be used.

---

## Notes

This property is currently used only when importing MS Word-supported documents using the [MSOffice](#) read module.

# ContentItem Property

---



Type	Default	Read Only	Description
[C#]ContentItemType			Gets or sets the content items to process.
[Visual Basic]ContentItemType	Default	No	

The ContentItemType enumeration may take the following values:

- Default – specifies that module-specific default settings should be used.
- Content – specifies to import only main content.
- WithMarkup – specifies to import main content and mark-up.

---

## Notes

This property is currently used only when importing MS Word-supported documents using the [MSOffice](#) read module.

None.

---

## Example

# DefaultFont Property

---



Type	Default	Read Only	Description
[C#]string			The font to use for text that does not specify a font.
[Visual Basic]String	"Helvetica"	No	

The font to use for text that does not specify a font.

---

Notes

None.

---

Example

# DefaultRect Property

---



Type	Default	Read Only	Description
[C#]XRect			
[Visual Basic]XRect	"Letter"	No	The default document size for documents that do not specify a size.

Some document types do not require a document size.

For example SVG and PostScript can both be constructed in such a way as to be simply a set of drawing instructions without any details of the size of the output device. Formats like RTF have no default bounding box so always require that one is supplied.

---

## Notes

This property allows you to specify a default size for use if the document does not specify a size.

None.

---

## Example

# DotsPerInch Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	96.0	No	The default resolution to use for the import.

The default resolution to use for the import.

Some file formats allows you to specify pixel based measurements which can only be converted to physical measurements in the context of a resolution. However some formats do not require a resolution to be specified and so a default may need to be supplied.

---

## Notes

This property is currently only used for the import of SVG documents.

None.

---

## Example

# EnableMSOfficeMacros Property

---



Type	Default	Read Only	Description
[C#]bool [Visual Basic] Boolean	false	No	Whether to enable macros when opening MS Office documents.

This property specifies whether to enable macros when opening MS Office documents. It is supported only for MS Word, Excel and PowerPoint. It is ignored in all other cases.

---

## Notes

By default, macros are disabled.

---

## Example

None.

# ErrorHandling Property

---



Type	Default	Read Only	Description
[C#]ErrorHandlingType			
[Visual Basic]ErrorHandlingType	OutputUntilError	No	The error handling behavior.

The ErrorHandlingType enumeration may take the following values:

- OutputUntilError
- NoOutputOnError

The OutputUntilError directs ABCpdf to process the current file as far as possible even if it encounters an error. This can be useful for dealing with common forms of corruption. For example EPS files sometimes get truncated. As long as the truncation is small the output will be the same as the complete file despite the fact that from technical point of view these files are invalid.

---

## Notes

However one cannot know exactly how small a truncation is. It might be that one needs to be absolutely sure that a file is valid; that it is better to throw an exception rather than risk an incomplete image. This behavior is achieved using NoOutputOnError.

---

Example

None.

# ExtraChecks

---



Type	Default Value	Read Only	Description
[C#]bool [Visual Basic]Boolean	false	No	Whether to apply extra processing to enable certain types of corrupt document to be read.

Some PDF files may not be fully compliant with PDF specification. In short they may be corrupt.

ABCpdf will work around minor forms of corruption in which the intent is obvious. However some more major forms of corruption require that the entire document is read and rebuilt. This will only occur if the ExtraChecks property is set to true.

Rebuilding a document is not ideal. Not only is it time consuming, but there may also be more than one way of performing the rebuild. Depending on the strategy used one may end up with two different results. Most often the results will, for all practical purposes, be identical. However on rare occasions they might be different. For this reason the default value of this property is false.

Many clients who need to process large quantities of PDF documents, from sources of varying quality, end

up writing code of the following form.

---

Notes

**[C#]**

```
try {
    theDoc.Read(theFile);
}
catch {
    XReadOptions xr = new
XReadOptions();
    xr.ExtraChecks = true;
    theDoc.Read(theFile, xr);
}
```

**[Visual Basic]**

```
Try
    theDoc.Read(theFile)
Catch
    Dim xr As New XReadOptions()
    xr.ExtraChecks = True
    theDoc.Read(theFile, xr)
End Try
```

None.

---

Example

# Frame Property

---



Type	Default	Read Only	Description
[C#]long [Visual Basic] Long	0	No	The frame to be read from a multiple frame image such as a movie.

This property is used by [read modules](#) that support frames or sub-documents.

Frames are numbered from one upwards. The default of zero indicates that a default frame or frame set should be used.

The SwfVector module uses this property when the [Operation](#) is null. When this occurs a temporary [SwfImportOperation](#) is created and the [ProcessingObjectEventArgs.Info.FrameNumber](#) is set to the value of this property. The default behavior is to read frame one.

---

## Notes

The MSOffice module uses this property to allow you to read individual worksheets from within an Excel spreadsheet. The default behavior is to read all the worksheets in order rather than simply select one of them.

---

**Example**

None.

# FileExtension Property

---



Type	Default	Read Only	Description
[C#]string			Gets or sets the file extension for data sources that do not have file names.
[Visual Basic]String	null	No	

This property is used by modules that can take files with different file extensions when (and only when) the specification of the source does not provide a file extension, such as a Stream or an array of bytes.

---

## Notes

For the [Default](#) modules, it is optional, but the modules may behave differently when it is specified. For the [XpsAny](#) and the [OpenOffice](#) modules, it is mandatory.

None.

---

## Example

# Log Property

---



Type	Default Value	Read Only	Description
[C#]string			
[Visual Basic]String	null	Yes	The log for the read operation.

During a read operation certain problems may be encountered. Problems that can be solved but perhaps only by making assumptions. Problems that might not justify an error but could be raised as a warning.

---

## Notes

For example during the EPS import process an unavailable font might be encountered. ABCpdf might substitute another font to replace the missing one. The output would probably look very similar to the intended output but it might not be identical.

ABCpdf will log these types of events using this property.

None.

---

## Example



# MakeFieldNamesUnique Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether field names should be changed to make them unique.

Input document forms can contain fields with the same name, or no name at all.

If the names of two fields in a PDF are the same, then the fields take the same value.

So if multiple form fields with the same name are added to a PDF, these fields will all appear to contain the same content. This is true even if the form fields in the original document contained different content.

---

## Notes

Setting this property will result in duplicate fields being renamed to allow the content to be different. Field without names will be added a placeholder name so they can be displayed properly in PDF.

None.

---

## Example



# OpenOfficeParameters Property

---



Type	Default	Read Only	Description
[C#]ListDictionary			OpenOffice.org PDF conversion control parameters.
[Visual Basic]ListDictionary	false	No	

This property is used by the OpenOffice.org import module. It is used to pass custom parameters to OpenOffice.org for precise control over the PDF conversion process.

The parameters are a set of named objects. The names and object types vary between versions of OpenOffice.org but ABCpdf defaults to a base set that is appropriate for most conversions.

Name	Description	Type	Value
UseLosslessCompression	Lossless compression of images. All pixels are preserved.	Boolean	false
Quality	Quality level for JPEG compression.	Int32	90
	Resample or		

---

Notes

ReduceImageResolution	downsize the images to a lower number of pixels per inch.	Boolean	false
MaxImageResolution	Target resolution for the images.	Int32	72
UseTaggedPDF	Preserve semantic structures such as table of contents, hyperlinks, and controls.	Boolean	true
ExportNotes	Export notes of Writer and Calc documents as PDF notes.	Boolean	true
ExportNotesPages	Export pages notes.	Boolean	false
UseTransitionEffects	Export Impress slide transition effect to respective PDF effects.	Boolean	false
FormsType	Select the format of submitting forms from within the PDF file. For	Int32	0

	example... 0		
	- FDF, 1 -		
	PDF, 2 -		
	HTML, 3 -		
	XML		

More details and other options can be found on the [OpenOffice.org web site](https://www.openoffice.org).

---

**Example**

None.

# Operation Property

---



Type	Default	Read Only	Description
[C#]Operation			
[Visual Basic]Operation	null	No	Gets or sets the Operat

This property is used by modules for which a suitable derived class exists. If it is null, those modules create a temporary [Operation](#) with behaviours.

ReadModule	Operation	Description
SwfVector	SwfImportOperation	If this property is null, the term <a href="#">SwfImportOperation</a> has its <a href="#">Timeout</a> and sets <a href="#">ProcessingObjectEventArgs</a> to <a href="#">Frame</a> once.
Xps	XpsImportOperation	If this property is null, the term <a href="#">XpsImportOperation</a> comprises <a href="#">GraphicLayer</a> 's of the pages
XpsAny	XpsImportOperation	If this property is null, the term <a href="#">XpsImportOperation</a> comprises <a href="#">GraphicLayer</a> 's of the pages

---

## Notes

None.

---

## Example



# Password Property

---

---



Type	Default	Read Only	Description
[C#] string	null	No	Gets or sets the password needed to read the source.
[Visual Basic] String			

Specify with this property the password for accessing the source, for example, when [reading](#) a PDF document.

---

## Notes

It is used for importing MS Word or MS Excel documents when using Microsoft Office via the XpsAny and MSOffice ReadModule. Note that for other Office applications, eg. MS PowerPoint, ABCpdf cannot import password protected documents. Currently, certain features such as form fields conversion may be disabled if the document is password protected, even if the correct password is supplied.

---

## Example

None.

# PreserveTransparency Property

---



Type	Default	Read Only	Description
[C#]bool [Visual Basic] Boolean	false	No	Gets or sets a value that indicates whether the transparency of raster images should be preserved.

This property is used when the source is a raster image. If the method (e.g. [Doc.AddImageObject](#)) called has a parameter for specifying the same setting, the parameter overrides this property.

---

## Notes

None.

---

## Example

# Timeout Property

---



Type	Default	Read Only	Description
[C#]int [Visual Basic] Integer	60000	No	Gets or sets the timeout in milliseconds for external components.

Timeout is used for external components, such as the XPS printer driver for [ReadModule](#) of XpsAny.

---

## Notes

None.

---

## Example

## SkipRevisions Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	0	No	Skip back a number of revisions when reading an incrementally saved PDF document.

Skip back a number of revisions when reading an incrementally saved PDF document.

PDF documents can be incrementally updated so that changes are appended to the document rather than overwriting the original. This means that it is possible to revert back to a previous version of the document. Setting this value allows you to skip back a specified number of revisions.

---

### Notes

See the [IndirectObject.Revision](#) property and the [ObjectSoup.Revisions](#) property for determining the number of revisions a document contains and the content which is associated with each revision.

---

**Example**

None.

# OpenPortfolios Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to automatically open the default document inside a PDF portfolio rather than the portfolio itself

Whether to automatically open the default document inside a PDF portfolio rather than the portfolio itself.

---

## Notes

If you wish to see the document the way a user would see it when it opens, then this parameter should be set to true. If you wish to operate upon the set of files embedded in the portfolio then it should be set to false.

---

## Example

See the [FileSpecification constructor](#) for the creation of portfolios. See the [Catalog.GetEmbeddedFiles](#) function for how to extract files from a portfolio.



# FromLbwh Function

---



Creates an XRect from a bottom left corner, a width and a height.

## [C#]

```
static XRect FromLbwh(double  
left, double bottom, double  
width, double height)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromLbwh(left As  
Double, bottom As Double, width  
As Double, height As Double) As  
XRect
```

---

## Params

Name	Description
left	The left coordinate.
bottom	The bottom coordinate.
width	The width of the rectangle.
height	The height of the rectangle.

This method constructs an XRect object from a bottom left corner, a width and a height.

---

**Notes**

It assumes that subsequent resizing and positioning operations on this object will also be based around the bottom left corner. As such the **Pin** property will remain at its default of `Corner.BottomLeft`.

---

**Example**

None.

# FromLtwH Function

---



Creates an XRect from a top left corner, a width and a height.

## [C#]

```
static XRect FromLtwH(double  
left, double top, double width,  
double height)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromLtwH(left As  
Double, top As Double, width As  
Double, height As Double) As  
XRect
```

---

## Params

Name	Description
left	The left coordinate.
top	The top coordinate.
width	The width of the rectangle.
height	The height of the rectangle.

This method constructs an XRect object from a top left corner, a width and a height.

---

**Notes**

It assumes that subsequent resizing and positioning operations on this object will also be based around the top left corner. As such the **Pin** property will be set to `Corner.TopLeft`.

---

**Example**

None.

# FromSides Function

---



Creates an XRect from the coordinates of two diagonally opposite corners.

## [C#]

```
static XRect FromSides(double  
xMin, double yMin, double xMax,  
double yMax)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromSides(xMin As  
Double, yMin As Double, xMax As  
Double, yMax As Double) As XRect
```

---

## Params

Name	Description
xMin	The minimum x coordinate.
yMin	The minimum y coordinate.
xMax	The maximum x coordinate.
yMax	The maximum y coordinate.

This method constructs an XRect object from the coordinates of two diagonally opposite corners.

---

**Notes**

If the minimum x values are larger than the minimum x values then the XRect will have a negative width. Similarly if the minimum y values are larger than the maximum y values then the XRect will have a negative height.

---

**Example**

None.

# FromPoints Function

---



Create the smallest XRect that encloses all the points supplied.

## [C#]

```
static XRect FromPoints(XPoint  
points)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromPoints(points  
As XPoint) As XRect
```

---

## Params

Name	Description
points	The points that should be enclosed.

Create the smallest XRect that encloses all the points supplied.

---

## Notes

If no points or null is supplied then the return value is null.

None.

---

**Example**



# XRect Constructor

---

XRect Constructor.

## [C#]

```
XRect()  
XRect(string value)
```

---

## Syntax

## [Visual Basic]

```
Sub New  
Sub New(value As String)
```

---

## Params

Name	Description
value	A string defining the initial rectangle in the format "left bottom right top"

These methods construct an XRect object.

To construct an XRect with initial values use one of the static constructors such as [XRect.FromSides](#), [XRect.FromLbwh](#) or [XRect.FromPoints](#).

---

## Notes

---

Example

None.

# Contains Function

---



Determine if this rectangle contains a specified point or rectangle.

## [C#]

```
bool Contains(XPoint point)
bool Contains(PointF point)
bool Contains(Point point)
bool Contains(float x, float y)
bool Contains(double x, double y)
bool Contains(XRect rect)
```

## [Visual Basic]

```
Function Contains(point As
XPoint) As Boolean
Function Contains(point As
PointF) As Boolean
Function Contains(point As Point)
As Boolean
Function Contains(x As Float, y
as Float) As Boolean
Function Contains(x As Double, y
as Double) As Boolean
Function Contains(rect As XRect)
As Boolean
```

Syntax

Name	Description
rect	The rectangle to test.

---

**Params**

point	The point to test.
x	The x coordinate of a point to test.
y	The y coordinate of a point to test.
return	Whether the provided object is contained by this one.

Determine if this rectangle contains a specified point or rectangle.

---

**Notes**

In the case of rectangles, all four corners must be contained for the function to return true.

If a null point or rectangle is passed this function will return false.

---

**Example**

None.

# FitIn Function

---



Fits the rectangle as content inside another rectangle.

## [C#]

```
void FitIn(XRect rect,  
ContentAlign align,  
ContentScaleMode scaleMode)
```

---

## Syntax

## [Visual Basic]

```
Sub FitIn(rect As XRect, align As  
ContentAlign, scaleMode As  
ContentScaleMode)
```

---

## Params

Name	Description
rect	The containing rectangle.
align	The content alignment.
scaleMode	The content scale mode.

The rectangle is fitted inside the containing rectangle.

The ContentAlign enumeration can take a combination of the following values:

- Center
- Left
- Right
- Top
- Bottom

The ContentScaleMode enumeration can take any of the following values:

---

Notes

- ShowAll – make the content the biggest and completely within the area while keeping the aspect ratio.
- NoBorder – make the content the smallest and covering the entire area while keeping the aspect ratio.
- ExactFit – scale the content possibly with distortion to fit the entire area. It is the same as assignment.
- NoScale – no scaling. The rectangle is repositioned.

None.

---

Example



# GetCorners Function

---

Get the four corners of the rectangle.

## [C#]

```
XPoint[] GetCorners()
```

---

### Syntax

## [Visual Basic]

```
Sub GetCorners() As XPoint()
```

---

### Params

Name	Description
return	The four corners of the rectangle.

---

### Notes

Get the four corners of the rectangle. The first item in the array is the bottom left and subsequent corners are presented clockwise from that point.

---

### Example

None.





# Inset Function

---

Insets the edges of the rectangle.

## [C#]

```
void Inset(double x, double y)
```

---

## Syntax

## [Visual Basic]

```
Sub Inset(x As Double, y As Double)
```

---

## Params

Name	Description
x	The amount to inset the left and right edges.
y	The amount to inset the top and bottom edges.

---

## Notes

Insets the edges of the rectangle by a specified horizontal and vertical amount.

The following code.

## [C#]

```
XRect rc = new XRect();
rc.String = "0 0 200 100";
Response.Write("Rect = " +
rc.String);
Response.Write("<br>");
rc.Inset(10, 20);
Response.Write("Inset = " +
rc.String);
```

### [Visual Basic]

#### Example

```
Dim rc As XRect = New XRect()
rc.String = "0 0 200 100"
Response.Write("Rect = " +
rc.String)
Response.Write("<br>")
rc.Inset(10, 20)
Response.Write("Inset = " +
rc.String)
```

Produces the following output.

```
Rect = 0 0 200 100
Inset = 10 20 190 80
```



# Intersect Function

---

Intersects this rectangle with another rectangle.

## [C#]

```
void Intersect(XRect rect)
```

---

### Syntax

## [Visual Basic]

```
Sub Intersect(rect As XRect)
```

---

### Params

Name	Description
rect	Another rectangle. The supplied rectangle is left unchanged by this function.

This rectangle becomes the intersection of this rectangle and the supplied rectangle.

---

### Notes

If a null rectangle is passed, this rectangle will remain unchanged.

None.

---

### Example



# IntersectsWith Function

---



Determine if this rectangle intersects with another.

## [C#]

```
bool IntersectsWith(XRect rect)
```

---

## Syntax

## [Visual Basic]

```
Function IntersectsWith(rect As XRect) As Boolean
```

---

## Params

Name	Description
rect	Another rectangle.
return	Whether the two rectangles intersect.

Determine if this rectangle intersects with another.

---

## Notes

If a null rectangle is passed this function will return false.

---

None.

## Example



# Magnify Function

---

Magnifies the rectangle.

## [C#]

```
void Magnify(double x, double y)
```

---

## Syntax

## [Visual Basic]

```
Sub Magnify(x As Double, y As Double)
```

---

## Params

Name	Description
x	The horizontal scale factor.
y	The vertical scale factor.

Scales the rectangle width and height by a specified horizontal and vertical amount.

---

## Notes

When you magnify a rectangle one corner of the rectangle is pinned and the width and height of the rectangle adjusted. The corner which is pinned is indicated by the [Pin](#) property. The default pin corner is the bottom left.

The following code.

### [C#]

```
XRect rc = new XRect();  
rc.String = "20 20 220 120";  
Response.Write("Rect = " +  
rc.String);  
Response.Write("<br>");  
rc.Magnify(0.5, 0.5);  
Response.Write("Scale = " +  
rc.String);
```

### [Visual Basic]

Example

```
Dim rc As XRect = New XRect()  
rc.String = "20 20 220 120"  
Response.Write("Rect = " +  
rc.String)  
Response.Write("<br>")  
rc.Magnify(0.5, 0.5)  
Response.Write("Scale = " +  
rc.String)
```

Produces the following output.

```
Rect = 20 20 220 120  
Scale = 20 20 120 70
```



# Move Function

---

Translate the rectangle.

## [C#]

```
void Move(double x, double y)
```

---

### Syntax

## [Visual Basic]

```
Sub Move(x As Double, y As Double)
```

---

### Params

Name	Description
x	The horizontal distance to move the rectangle.
y	The vertical distance to move the rectangle.

---

### Notes

Moves the rectangle maintaining the width and height.

The following code.

## [C#]

```
XRect rc = new XRect();
rc.String = "20 20 220 120";
Response.Write("Rect = " +
rc.String);
rc.Move(50, 50);
Response.Write("<br>");
Response.Write("Move = " +
rc.String);
```

### [Visual Basic]

#### Example

```
Dim rc As XRect = New XRect()
rc.String = "20 20 220 120"
Response.Write("Rect = " +
rc.String)
rc.Move(50, 50)
Response.Write("<br>")
Response.Write("Move = " +
rc.String)
```

Produces the following output.

```
Rect = 20 20 220 120
Move = 70 70 270 170
```



# Position Function

---

Position the bottom left of the rectangle.

## [C#]

```
void Position(double x, double y)
void Position(double x, double y,
Corner corner)
```

---

## Syntax

## [Visual Basic]

```
Sub Position(x As Double, y As
Double)
Sub Position(x As Double, y As
Double, corner As Corner)
```

---

## Params

Name	Description
x	The new left position.
y	The new bottom position.
corner	The corner to position.

Moves the rectangle to the supplied position while maintaining the width and height.

---

## Notes

The corner moved to the location is indicated by the [Pin](#) property but you can override this default by specifying a corner when calling this

function.

The following code.

### [C#]

```
XRect rc = new XRect();  
rc.String = "20 20 220 120";  
Response.Write("Rect = " +  
rc.String);  
Response.Write("<br>");  
rc.Position(50, 50);  
Response.Write("Pos. = " +  
rc.String);
```

### [Visual Basic]

**Example**

```
Dim rc As XRect = New XRect()  
rc.String = "20 20 220 120"  
Response.Write("Rect = " +  
rc.String)  
Response.Write("<br>")  
rc.Position(50, 50)  
Response.Write("Pos. = " +  
rc.String)
```

Produces the following output.

```
Rect = 20 20 220 120  
Pos. = 50 50 250 150
```



# Resize Function

---

Resizes the rectangle.

## [C#]

```
void Resize(double w, double h)
void Resize(double w, double h,
Corner corner)
```

---

## Syntax

## [Visual Basic]

```
Sub Resize(w As Double, h As
Double)
Sub Resize(w As Double, h As
Double, corner as Corner)
```

---

## Params

Name	Description
w	The new width.
h	The new height.
corner	The corner to pin.

Changes the width and height of the rectangle while maintaining the position.

When you change the width or height of a rectangle one corner of the rectangle is pinned to maintain position. The corner which is pinned

---

## Notes

is indicated by the [Pin](#) property but you can override this default by specifying a corner when calling this function.

The following code.

### [C#]

```
XRect rc = new XRect();  
rc.String = "20 20 220 120";  
Response.Write("Rect = " +  
rc.String);  
Response.Write("<br>");  
rc.Resize(50, 150);  
Response.Write("Pos. = " +  
rc.String);
```

### [Visual Basic]

#### Example

```
Dim rc As XRect = New XRect()  
rc.String = "20 20 220 120"  
Response.Write("Rect = " +  
rc.String)  
Response.Write("<br>")  
rc.Resize(50, 150)  
Response.Write("Pos. = " +  
rc.String)
```

Produces the following output.

```
Rect = 20 20 220 120  
Pos. = 20 20 70 170
```





# Union Function

---

Union this rectangle with another rectangle.

## [C#]

```
void Union(XRect rect)
void Union(XRect rect, bool
noAreaIsEmpty)
```

---

## Syntax

## [Visual Basic]

```
Sub Union(rect As XRect)
Sub Union(rect As XRect,
noAreaIsEmpty as Boolean)
```

---

## Params

Name	Description
rect	The rectangle to add to this one. The supplied rectangle is left unchanged by this function.
noAreaIsEmpty	Whether to treat rectangles with no area as empty - default true.

Union the rectangle with another rectangle.

This rectangle becomes the smallest rectangle

that encloses both rectangles.

Rectangles with no area will, by default, be treated as being empty. A union of a non-empty rectangle with an empty rectangle always results in the output being set to the non-empty rectangle value. If both rectangles are empty this rectangle will be left unchanged.

---

**Notes**

If the `noAreaIsEmpty` parameter is set to `false` then rectangles with no area will be treated as points and the output union will include that point.

None.

---

**Example**



# SetRect Function

---

Sets the location and size of the rectangle.

## [C#]

```
void SetRect(double x, double y,  
double w, double h)  
void SetRect(XRect rect)
```

---

## Syntax

## [Visual Basic]

```
Sub SetRect(x As Double, y As  
Double, w As Double, h As Double)  
Sub SetRect(rect As XRect)
```

---

## Params

Name	Description
x	The new left position.
y	The new bottom position.
w	The new width.
h	The new height.
rect	The source rectangle.

Sets the location and size of the rectangle.

The width and height of the rectangle are set to the new width and height.

The rectangle is then moved to the supplied position while maintaining the width and height. The corner moved to the location is indicated by the [Pin](#) property. The default pin corner is the bottom left.

The overload taking an XRect copies the effective location and size. It behaves as if this XRect and the parameter XRect use the default [Pin](#) and the default coordinate settings ([Doc.TopDown](#) and [Doc.Units](#)) so it functions differently from the other overload and from copying using [String](#). [Pin](#) is not copied.

---

Notes

For example suppose you have a Doc object (called doc) for which you have set the Doc.Units to mm and also a separate XRect (called rect) that you have created. If you call `rect.SetRect(doc.Rect)` the rect will contain coordinates in points rather than mm. Similarly if you call `doc.Rect.SetRect(rect)`, the point based units within the rect will be converted to mm for insertion into the doc.Rect. So there is an automatic conversion between coordinate systems.

The following code.

**[C#]**

```
XRect rc = new XRect();
rc.String = "20 20 220 120";
Response.Write("Rect = " +
rc.String);
Response.Write("<br>");
```

```
rc.SetRect(20, 40, 50, 150);  
Response.Write("Pos. = " +  
rc.String);
```

---

**Example**

### **[Visual Basic]**

```
Dim rc As XRect = New XRect()  
rc.String = "20 20 220 120"  
Response.Write("Rect = " +  
rc.String)  
Response.Write("<br>")  
rc.SetRect(20, 40, 50, 150)  
Response.Write("Pos. = " +  
rc.String)
```

Produces the following output.

```
Rect = 20 20 220 120  
Pos. = 20 40 70 190
```



# SetSides Function

---

Sets the sides of the rectangle.

## [C#]

```
void SetSides(double x1, double y1, double x2, double y2)
```

---

## Syntax

## [Visual Basic]

```
Sub SetSides(x1 As Double, y1 As Double, x2 As Double, y2 As Double)
```

---

## Params

Name	Description
x1	The new left position.
y1	The new bottom position.
x2	The new right position.
y2	The new top position.

---

## Notes

Sets the coordinates of the rectangle.

---

None.

## Example



# ToString Function

---

Returns a string representation of the object.

## [C#]

```
override string ToString()
```

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Syntax

Name	Description
return	The string representation of the object.

---

### Params

This method returns the string value of the object. This is equivalent to reading the [String](#) property of the object.

---

### Notes

None.

---

### Example



# Equals Function

---



Test whether the two rectangles are effectively the same

## [C#]

```
bool Equals(XRect other, double epsilon)
bool Equals(XRect other)
override bool Equals(object other)
```

---

### Syntax

## [Visual Basic]

```
Function Equals(other As XRect, epsilon As Double) As Boolean
Function Equals(other As XRect) As Boolean
Overrides Function Equals(other As Object) As Boolean
```

---

### Params

Name	Description
other	The rect to test against.
epsilon	The largest difference in values which will still be defined as equal
return	Whether the two rects are the same.

Test whether the two rectangles are effectively the same.

Rectangles are considered equal if they have the same position, width and height. This represents value equality for the rectangles in question.

---

**Notes**

The underlying components of an XRect are represented as floating point numbers. Floating point numbers are subject to rounding errors, so there has to be a degree of latitude when comparing coordinate values. The degree of latitude is, by default, determined by the limitations defined in the PDF Specification. This is, broadly speaking, 5 decimal points so the default epsilon is typically 0.00001. However if you wish to rely on a specific epsilon value you should provide one.

---

**Example**

None.



# GetHashCode Function

---

A hash code for the XRect

## [C#]

```
override int GetHashCode()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function GetHashCode()  
As Integer
```

---

### Params

Name	Description
return	The returned hash code.

---

### Notes

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

---

### Example

None.

# Bottom Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The bottom coordinate.

Allows you access to the bottom coordinate.

---

## Notes

None.

---

## Example

# Height Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The height of the rectangle.

Allows you access to the height of the rectangle.

When you change the width or height of a rectangle one corner of the rectangle is pinned and the width or height of the rectangle adjusted to match the new width or height. The corner which is pinned is indicated by the [Pin](#) property. The default pin corner is the bottom left.

---

## Notes

None.

---

## Example

# Left Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The left coordinate.

Allows you access to the left coordinate.

---

## Notes

None.

---

## Example

# Pin Property

---



Type	Default Value	Read Only	Description
[C#] Corner			
[Visual Basic] Corner	Corner.BottomLeft	No	The corner of the rectangle to pin.

Allows you change the pinned corner of the rectangle.

The Corner enumeration may take the following values:

- BottomLeft
- TopLeft
- BottomRight
- TopRight

Some operations require that the rectangle is pinned to a location. For example if you want to change the width of a rectangle you can do this either by shifting the left side or the right side. If the pin property is set to the bottom or top left then the left side of the rectangle will be kept fixed and the right side shifted. Conversely if the pin property is set to the bottom or top right then the right side of the rectangle will be kept fixed and the left side shifted.



## Why is my Pin a number?

In older versions of ABCpdf the Pin property was a number. To assign a Corner to the Pin property you needed to cast the value. So you might find code of this form.

```
theRect.Pin =  
(int)XRect.Corner.TopRight
```

In Version 8 the Pin property has been changed to a true enumeration. This means you can assign the Corner directly.

```
theRect.Pin =  
XRect.Corner.TopRight
```

If your code is bound to integers rather than enumerations then it is safe to cast the number to a Corner.

```
theRect.Pin =  
(XRect.Corner)myIntegerValue
```

# Rectangle Property

---



Type	Default	Read Only	Description
[C#] Rectangle			
[Visual Basic] Rectangle	n/a	No	The System.Drawing.Rectangle.

The rectangle as a System.Drawing Rectangle.

Windows coordinates are measured in distances from the top left of the drawing surface while PDF coordinates are measured from the bottom left.

So when you use this property the coordinates must be re-mapped. This needs to be done in the context of a containing object. Properties such as the [Doc.Rect](#) and the [XImage.Selection](#) have containers but others such as the [Doc.MediaBox](#) do not. In these cases the rectangles are assumed to contain themselves.

Notes

You may find it easier to work with .NET Rectangles than PDF rectangles. However remember that operations such as [Transforms](#) work on the underlying PDF coordinates and not on the abstracted Windows coordinates.

The following code adds two blocks of text to a document. The positioning is done using standard .NET Rectangles.

## [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
Rectangle rc = theDoc.MediaBox.Rectangle;
rc.Inflate(-50, -50);
rc.Height = 250;
theDoc.Rect.Rectangle = rc;
theDoc.FrameRect();
theDoc.AddText("First Rectangle...");
rc.Offset(0, 300);
theDoc.Rect.Rectangle = rc;
theDoc.FrameRect();
theDoc.AddText("Second Rectangle...");
theDoc.Save(Server.MapPath("xrectrectangle.pdf"));
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
Dim rc As Rectangle = theDoc.MediaBox.Rectangle
rc.Inflate(-50, -50)
rc.Height = 250
theDoc.Rect.Rectangle = rc
theDoc.FrameRect()
theDoc.AddText("First Rectangle...")
rc.Offset(0, 300)
theDoc.Rect.Rectangle = rc
theDoc.FrameRect()
theDoc.AddText("Second Rectangle...")
theDoc.Save(Server.MapPath("xrectrectangle.pdf"))
theDoc.Clear()
```

---

Example

First  
Rectangle...

Second  
Rectangle...

xrectrectangle.pdf

# Right Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The right coordinate.

Allows you access to the right coordinate.

---

Notes

None.

---

Example

# String Property

---



Type	Default	Read Only	Description
[C#] string	"0 0 0	No	The rect as a string.
[Visual Basic] String	0"		

Allows you access to the rect as a string.

When working in standard PDF coordinates the format of the string is "left bottom right top". When working with [top-down](#) coordinates the format of the string is "left top right bottom".

You can use the following page sizes as shortcuts when assigning a string value to an XRect:

Paper Size	Width in Points	Height in Points
A3	842	1190
A4	595	842
A5	420	595
B4	729	1032
B5	516	729
Letter	612	792
Legal	612	1008
Tabloid	792	1224
Ledger	1224	792

Statement	396	612
Executive	540	720
Folio	612	936
Quarto	610	780
10x14	720	1008

The following code.

### [C#]

```
XRect rc = new XRect();
rc.String = "10 10 200 100";
Response.Write("Width = " +
rc.Width.ToString());
Response.Write("<br>");
Response.Write("Height = " +
rc.Height.ToString());
```

### [Visual Basic]

**Example**

```
Dim rc As XRect = New XRect()
rc.String = "10 10 200 100"
Response.Write("Width = " +
rc.Width.ToString())
Response.Write("<br>")
Response.Write("Height = " +
rc.Height.ToString())
```

Produces the following output.

```
Width = 190
Height = 90
```



# Top Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The top coordinate.

Allows you access to the top coordinate.

---

Notes

None.

---

Example

# Width Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	No	The width of the rectangle.

Allows you access to the width of the rectangle.

When you change the width or height of a rectangle one corner of the rectangle is pinned and the width or height of the rectangle adjusted to match the new width or height. The corner which is pinned is indicated by the [Pin](#) property. The default pin corner is the bottom left.

---

## Notes

None.

---

## Example

# HasArea Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the rectangle has area.

Determine if the rectangle encloses any area.

---

## Notes

If the width or height is less than the minimum resolution of the PDF coordinate space then the rectangle is defined as having no area. If it has no area it is simply a point.

---

## Example

None.



# GetBitmap Function

---

Renders the current area of the current page to a Bitmap.

## [C#]

```
System.Drawing.Bitmap GetBitmap()
```

## [Visual Basic]

```
Function GetBitmap() As  
System.Drawing.Bitmap
```

### Syntax

- may throw `Exception()`

Name	Description
return	The Bitmap containing the image.

### Params

Use this method to render the PDF to a `System.Drawing.Bitmap`.

The output is a render of the current `Doc.Rect` of the current `Doc.Page`.

Any page rotation specified in the PDF page is applied so that the output render is the correct orientation. This may mean that the output width

### Notes

and height are transposed copies of the width and height as specified in the Doc.Rect.

You can then use this Bitmap for drawing to screen or for manipulation using System.Drawing routines.

---

**Example**

See the [AntiAliasPolygons](#) property.

# GetData Function

---



Renders the current area of the current page to memory.

## [C#]

```
byte[] GetData(string name)
```

## [Visual Basic]

```
Function GetData(name As String)  
As Byte()
```

Syntax

- may throw `Exception()`

Params

Name	Description
name	A dummy file name used to determine the type of image required.
return	The image as an array of bytes.

Use this method to render the PDF to memory.

The output is a render of the current [Doc.Rect](#) of the current [Doc.Page](#).

Any page rotation specified in the PDF page is

applied so that the output render is the correct orientation. This may mean that the output width and height are transposed copies of the width and height as specified in the Doc.Rect.

The file path extension determines the format of the output. The file name extensions which may be used are the same as that used for the [Save](#) method. Typical extensions used include .TIF, .TIFF, JPG, GIF, PNG, .BMP or .EMF. EMF is a vector rather than raster format which can be useful when you require resolution independence and smaller files.

---

#### Notes

Normally you will want to render your documents using the Save method. However sometimes you will need to obtain your image as raw data rather than in file format. The GetData method allows you to do this

You may wish to write such an image direct to a client browser rather than going through an intermediate file. The data you obtain using GetData can be written direct to an HTTP stream using Response.BinaryWrite. Similarly you may wish to obtain raw data for insertion into a database.

None.

---

#### Example

# Save Method

---



Renders and saves the current area of the current page.

## [C#]

```
void Save(string path)
void Save(string name, Stream
stream)
```

## [Visual Basic]

### Syntax

```
Sub Save(path As String)
Sub Save(name As String, stream
As Stream)
```

- may throw Exception()

### Params

Name	Description
path	The destination file path.
name	A dummy file name used to determine the type of image required.
stream	The destination stream.

Use this method to render the PDF.

The output is a render of the current [Doc.Rect](#) of the current [Doc.Page](#).

Any page rotation specified in the PDF page is applied so that the output render is the correct orientation. This may mean that the output width and height are transposed copies of the width and height as specified in the [Doc.Rect](#).

The file path extension determines the format of the output. The file name extensions which may be used are [.TIF](#), [.TIFF](#), [.JPG](#), [.GIF](#), [.PNG](#), [.BMP](#), [.JP2](#), [.PSD](#), [.EMF](#), [.PS](#) and [.EPS](#).

---

#### Notes

JP2 is used for the JPEG 2000 format. EMF is a vector rather than raster format which can be useful when you require resolution independence and smaller files. PS is raw vector PostScript-compatible output. EPS is the Encapsulated Postscript format. The particular type of EPS produced by ABCpdf conforms to the DSC (Document Structuring Conventions) standard, which is a subset of EPS intended to make EPS files more usable.

In addition you can render to any of the file types specified as part of the [GetText](#) method - [.TXT](#), [.SVG](#), [.SVG+](#) and [.SVG+2](#).

See the [AntiAliasPolygons](#) property.

---

#### Example

# AntiAliasImages Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether

Determines whether image transformations will be interpolated.

This feature is most useful when the original embedded image resolution is lower than the output resolution. When this property is set to true interpolation is used, increasing output quality.

## Notes

Whether the edges of images are anti-aliased is determined by the AntiAliasImages property.

The following example shows the effect that this parameter has on the output image.

## [C#]

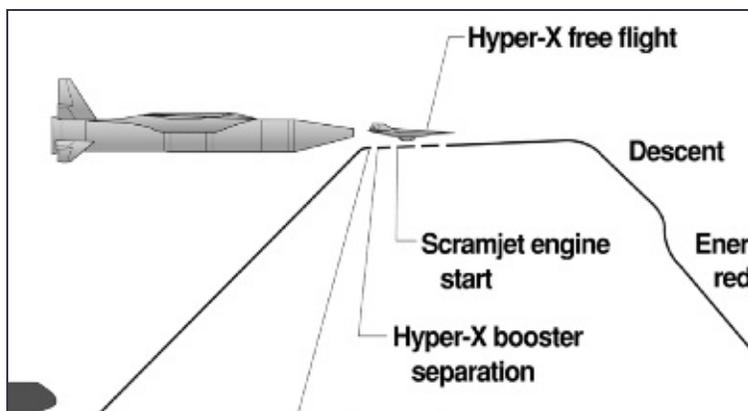
```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/HyperX.png"));
theDoc.Rect.Inset(200, 200);
// Render document with AntiAliasImages = true
theDoc.Rendering.Save(Server.MapPath("Rendering.png"));
// Render document with AntiAliasImages = false
theDoc.Rendering.AntiAliasImages = false;
// Save the image
theDoc.Rendering.Save(Server.MapPath("Rendering.png"));
```

```
theDoc.Clear();
```

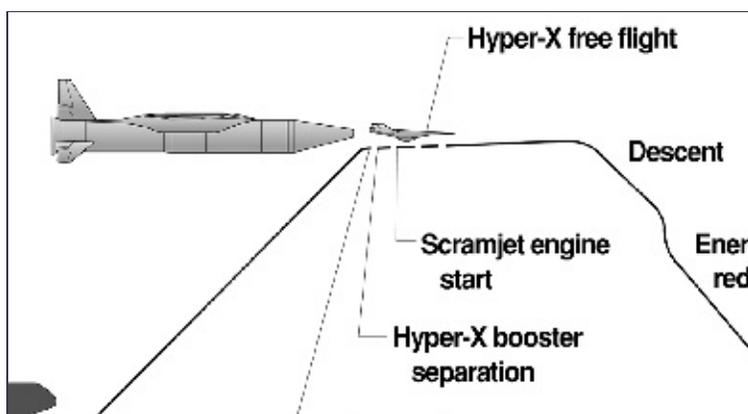
## [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../mypics/HyperX.png"))  
theDoc.Rect.Inset(200, 200)  
' Render document with AntiAliasImages = true  
theDoc.Rendering.Save(Server.MapPath("RenderingAntiAliasImagesTrue.png"))  
' Render document with AntiAliasImages = false  
theDoc.Rendering.AntiAliasImages = False  
' Save the image  
theDoc.Rendering.Save(Server.MapPath("RenderingAntiAliasImagesFalse.png"))  
theDoc.Clear()
```

Example



RenderingAntiAliasImagesTrue.png



RenderingAntiAliasImagesFalse.png



# AntiAliasPolygons Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether

Determines whether polygons will be rendered with anti-aliased (smooth) edges.

---

## Notes

Anti-aliasing is a technique for using gradients of color to eliminate jagged edges on drawn. The object edges are blurred to reduce pixelation.

The following example shows the effect that this parameter has on the rendered output.

### [C#]

```
Doc theDoc = new Doc();
// Add a polygon
theDoc.Color.String = "255 0 0";
theDoc.AddPoly("32 650 50 704 68 650 22 683 79 650");
theDoc.Rect.String = "20 650 80 704";
// Render the drawn area with AntiAliasPolygons
Bitmap antiAliasedBitmap = theDoc.Rendering.GetBitmap();
// Render the drawn area without AntiAliasPolygons
theDoc.Rendering.AntiAliasPolygons = false;
Bitmap aliasedBitmap = theDoc.Rendering.GetBitmap();
// Add magnified aliased image
```

```

theDoc.Rect.String = "5 20 605 560";
theDoc.AddImageBitmap(aliasedBitmap, false);
// Anotate
theDoc.Color.String = "black";
theDoc.FontSize = 30;
theDoc.Pos.String = "20 750";
theDoc.AddText("Original path:");
theDoc.Pos.String = "20 620";
theDoc.AddText("Magnified rendered image:");
// Render the document with aliased image
theDoc.Rendering.DotsPerInch = 36;
theDoc.Rect.String = theDoc.MediaBox.String;
theDoc.Rendering.Save(Server.MapPath("Rendering"));
// Add magnified antialiased image
theDoc.Rect.String = "5 20 605 560";
theDoc.AddImageBitmap(antiAliasedBitmap, false);
// Render the document with antialiased image
theDoc.Rendering.DotsPerInch = 36;
theDoc.Rect.String = theDoc.MediaBox.String;
theDoc.Rendering.Save(Server.MapPath("Rendering"));
theDoc.Clear();

```

### [Visual Basic]

```

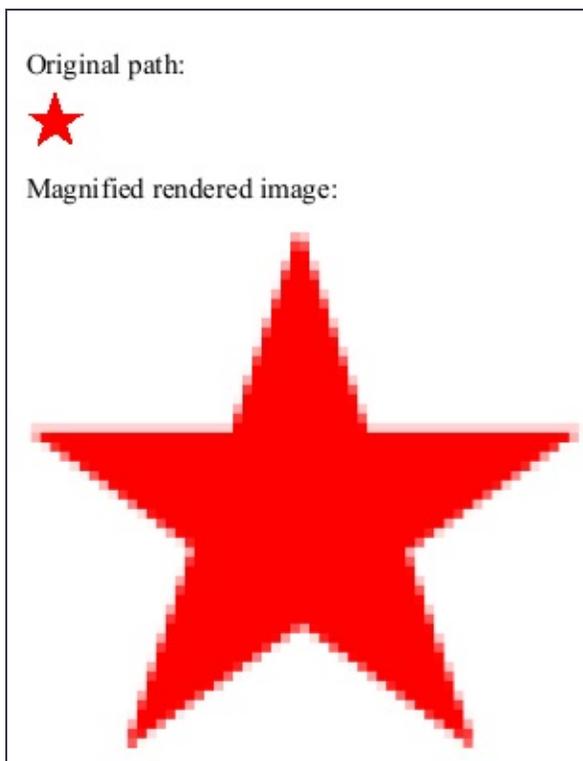
Dim theDoc As Doc = New Doc()
' Add a polygon
theDoc.Color.String = "255 0 0"
theDoc.AddPoly("32 650 50 704 68 650 22 683 79 650 32 650")
theDoc.Rect.String = "20 650 80 704"
' Render the drawn area with AntiAliasPolygons
Dim antiAliasedBitmap As Bitmap = theDoc.Rendering.Save(Server.MapPath("AntiAliasPolygons"))
' Render the drawn area without AntiAliasPolygons
theDoc.Rendering.AntiAliasPolygons = False
Dim aliasedBitmap As Bitmap = theDoc.Rendering.Save(Server.MapPath("AliasedPolygons"))
' Add magnified aliased image
theDoc.Rect.String = "5 20 605 560"
theDoc.AddImageBitmap(aliasedBitmap, False)

```

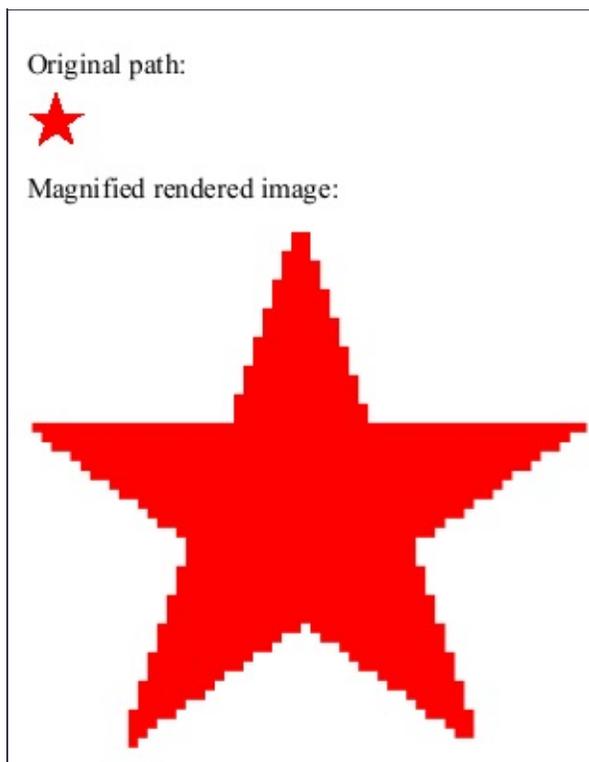
---

Example

```
' Anotate
theDoc.Color.String = "black"
theDoc.FontSize = 30
theDoc.Pos.String = "20 750"
theDoc.AddText("Original path:")
theDoc.Pos.String = "20 620"
theDoc.AddText("Magnified rendered image:")
' Render the document with aliased image
theDoc.Rendering.DotsPerInch = 36
theDoc.Rect.String = theDoc.MediaBox.String
theDoc.Rendering.Save(Server.MapPath("Rendering
' Add magnified antialiased image
theDoc.Rect.String = "5 20 605 560"
theDoc.AddImageBitmap(antiAliasedBitmap, False)
' Render the document with antialiased image
theDoc.Rendering.DotsPerInch = 36
theDoc.Rect.String = theDoc.MediaBox.String
theDoc.Rendering.Save(Server.MapPath("Rendering
theDoc.Clear()
```



RenderingAntiAliasPolygonsTrue.png



RenderingAntiAliasPolygonsFalse.png

# AntiAliasScene Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to apply entire scene anti-aliasing.

Determines whether to apply entire scene anti-aliasing.

If this property is set to true then text, polygons and images will be anti-aliased together instead of individually. This uses considerably more memory and takes considerably longer but can result in higher quality output.

---

## Notes

When this property is set to true this implicitly disables any other anti-aliasing applied to individual object types.

Anti-aliasing is a technique for using gradients of color to eliminate jagged edges when objects are drawn.

None.

---

## Example



# AntiAliasText Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether

Determines whether text will be rendered with anti-aliased edges

---

## Notes

Anti-aliasing is a technique for using gradients of color to eliminate jagged edges on objects that are drawn. The object edges are blurred to reduce pixelation.

The following example shows the effect that this parameter has on text rendering.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(50, 50);
// Add some text
theDoc.FontSize = 48;
theDoc.Pos.String = "50 690";
int id = theDoc.AddText("Abc");
// Render images
theDoc.Rect.String = theDoc.GetInfo(id, "rect");
theDoc.Rendering.AntiAliasText = true;
Bitmap antialiasedBitmap = theDoc.Rendering.GetBitmap(id);
theDoc.Rendering.AntiAliasText = false;
```

```

Bitmap aliasedBitmap = theDoc.Rendering.GetBitmap(
// Add enlarged images to the pdf file
theDoc.Rect.Magnify(5, 5);
theDoc.Rect.Move(0, -300);
theDoc.AddImageBitmap(aliasedBitmap, false);
theDoc.Rect.Move(0, -300);
theDoc.AddImageBitmap(antialiasedBitmap, false);
// Annotate
theDoc.Rect.String = theDoc.MediaBox.String;
theDoc.Color.String = "255 0 0";
theDoc.FontSize = 36;
theDoc.Pos.String = "50 740";
theDoc.AddText("Original text:");
theDoc.Pos.String = "50 620";
theDoc.AddText("Magnified aliased image:");
theDoc.Pos.String = "50 320";
theDoc.AddText("Magnified antialiased image:");
// Save render of pdf files
theDoc.Rendering.AntiAliasText = true;
theDoc.Rendering.DotsPerInch = 36;
theDoc.Rendering.Save(Server.MapPath("Rendering"));
theDoc.Clear();

```

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(50, 50)
' Add some text
theDoc.FontSize = 48
theDoc.Pos.String = "50 690"
Dim id As Integer = theDoc.AddText("Abc")
' Render images
theDoc.Rect.String = theDoc.GetInfo(id, "rect")
theDoc.Rendering.AntiAliasText = True
Dim antialiasedBitmap As Bitmap = theDoc.Rendering.GetBitmap(
theDoc.Rendering.AntiAliasText = False
Dim aliasedBitmap As Bitmap = theDoc.Rendering.GetBitmap(

```

---

**Example**

```
' Add enlarged images to the pdf file
theDoc.Rect.Magnify(5, 5)
theDoc.Rect.Move(0, -300)
theDoc.AddImageBitmap(aliasedBitmap, False)
theDoc.Rect.Move(0, -300)
theDoc.AddImageBitmap(antialiasedBitmap, False)
' Annotate
theDoc.Rect.String = theDoc.MediaBox.String
theDoc.Color.String = "255 0 0"
theDoc.FontSize = 36
theDoc.Pos.String = "50 740"
theDoc.AddText("Original text:")
theDoc.Pos.String = "50 620"
theDoc.AddText("Magnified aliased image:")
theDoc.Pos.String = "50 320"
theDoc.AddText("Magnified antialiased image:")
' Save render of pdf files
theDoc.Rendering.AntiAliasText = True
theDoc.Rendering.DotsPerInch = 36
theDoc.Rendering.Save(Server.MapPath("Rendering"))
theDoc.Clear()
```

Original text:

Abc

Magnified aliased image:

Abc

Magnified antialiased image:

Abc

RenderingAntiAliasText.png

# AutoRotate Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether pages should be automatically rotated.

A PDF document may contain pages in portrait or landscape mode.

If the pages are in landscape mode the viewing application should rotate them by 90 degrees before displaying them.

---

## Notes

If you use ABCpdf to render a landscape page then it will automatically apply the correct rotation. However if you want to disable this automatic rotation you can set this property to false.

None.

---

## Example

# BitsPerChannel Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	8	No	The output bits per color channel.

This property determines the precision of the output color channels.

The value is specified in terms of the number of bits used to represent each channel of color. This can be 1, 8 or 16.

All [color spaces](#) support 8 and 16 bits per channel. Only the Gray color space supports 1 bit per channel.

---

## Notes

The TIFF output format supports all output precisions. The JP2 (JPEG 2000) and PSD (Photoshop) formats support 8 and 16 bits per channel. The JPG, GIF, PNG and .BMP. output formats support 8 bit RGB and 8 bit Gray only.

See the [DefaultHalftone](#) property.

---

## Example



# ColorSpace Property

---



Type	Default Value
[C#] <code>XRendering.ColorSpaceType</code>	<code>XRendering.ColorSpaceType</code>
[Visual Basic] <code>XRendering.ColorSpaceType</code>	

The name of the output color space.

The `XRendering.ColorSpaceType` enumeration may take the following values:

- `Rgb` - red, green and blue
- `Gray` - grayscale
- `Cmyk` - cyan, magenta, yellow and black
- `Lab` - a device independent color space
- `Indexed` - indexed RGB; see the [PaletteSize](#) property

The following table shows the supported color spaces of each output format:

Output format	RGB	Gray	
TIFF	●	●	
PSD (Photoshop)	●	●	
JP2 (JPEG 2000)	●	●	
JPG	●	●	
BMP	●	●	

\* The JPEG file format itself does not support indexed colors. Using Indexed color space will result in a JPEG output with the number of colors in the color profile will still be RGB. This is what you would expect if you converted it to a JPEG file.

### Why is my ColorSpace a string?

In older versions of ABCpdf the ColorSpace property was a string.

```
theDoc.Rendering.ColorSpace = "CMYK"
```

In Version 8 the ColorSpace property was changed to a true enumeration as it allows the compiler to ensure that the values you should look like this.

```
theDoc.Rendering.ColorSpace = XRendering.ColorSpaceType.CMYK
```

The names of the items in the XRendering.ColorSpaceType enumeration are the same as the strings used in previous versions. So changing your code to use the enumeration is a simple replace operation.

Note that the enumeration is the XRendering.ColorSpaceType enumeration used for rendering. There is a different ColorSpace enumeration used for document. The two are not the same.

Alternatively if you need to convert between enumerations and strings you can convert from a string to an enumeration use the following code.

```
XRendering.ColorSpaceType csType =  
(XRendering.ColorSpaceType)Enum.Parse(typeof(XRendering.ColorSpaceType), csString, true)
```

To convert from an enumeration to a string use the following code.

```
string csString = csType.ToString("G")
```

The following example shows the effect that this parameter has on

### [C#]

```
Doc theDoc = new Doc();  
theDoc.AddImage(Server.MapPath("../mypics/Shuttl  
theDoc.Rect.String = theDoc.MediaBox.String;  
// Render document in Gray colorspace  
theDoc.Rendering.ColorSpace = "Gray";  
theDoc.Rendering.DotsPerInch = 36;  
theDoc.Rendering.Save(Server.MapPath("Rendering  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.AddImage(Server.MapPath("../mypics/Shuttl  
theDoc.Rect.String = theDoc.MediaBox.String  
' Render document in Gray colorspace  
theDoc.Rendering.ColorSpace = "Gray"  
theDoc.Rendering.DotsPerInch = 36  
theDoc.Rendering.Save(Server.MapPath("Rendering  
theDoc.Clear()
```

Example



Shuttle.jpg



RenderingColorSpace.png



# DefaultHalftone Property

---



Type	Default Value	Read Only	De
[C#] string	"Spot,30,100,CosineDot"	No	Ha
[Visual Basic] String			

Specifies the default halftone type and options.

The default halftone remains in effect until an embedded halftone

The Type can be:

- Spot
- ErrorDiffusion
- OrderedDither
- Threshold

If the type is Spot then an angle in degrees, frequency and spot size are specified. These should be separated by commas with no extra spaces. The following functions are available:

- SimpleDot
- InvertedSimpleDot
- DoubleDot
- InvertedDoubleDot
- CosineDot
- Double
- InvertedDouble
- Line

- LineX
- LineY
- Round
- Ellipse
- EllipseA
- InvertedEllipseA
- EllipseB
- EllipseC
- InvertedEllipseC
- Square
- Cross
- Rhomboid
- Diamond

If the type is Threshold you should specify a value - for example used when calculating the value of a black and white pixel. If the threshold the pixel will be black otherwise it will be white.

For full details of how Halftones work you should see the Adobe available from the Adobe web site.

The following example shows the effect that this parameter has on

**[C#]**

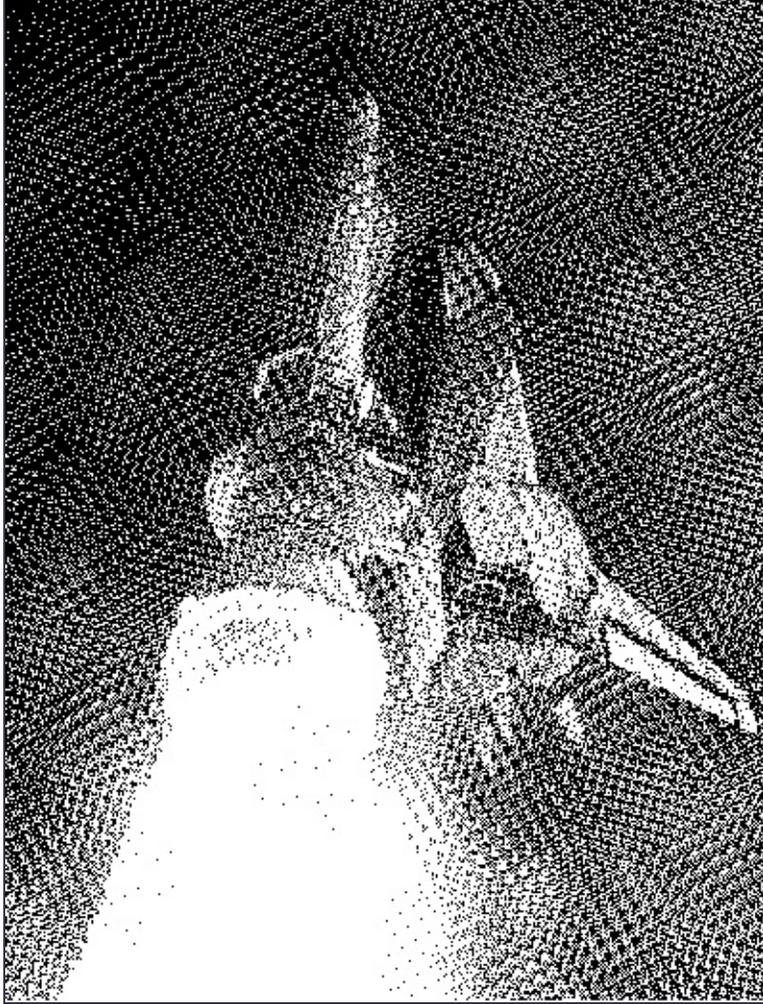
```
Doc theDoc = new Doc();
XImage image = new XImage();
image.SetFile(Server.MapPath("../mypics/Shuttle"));
theDoc.Rect.String = image.Selection.String;
theDoc.AddImage(image);
// Save rendered image as black and white picture
function
theDoc.Rendering.UseEmbeddedHalftone = false;
theDoc.Rendering.DotsPerInch = 50;
theDoc.Rendering.ColorSpace = XRendering.ColorSpace.Grayscale;
theDoc.Rendering.BitsPerChannel = 1;
```

```
theDoc.Rendering.DefaultHalftone = "Spot,30,100,
theDoc.Rendering.Save(Server.MapPath("Rendering
// Save rendered image as black and white pictu
function
theDoc.Rendering.DefaultHalftone = "Spot,0,100,
theDoc.Rendering.Save(Server.MapPath("Rendering
theDoc.Clear();
```

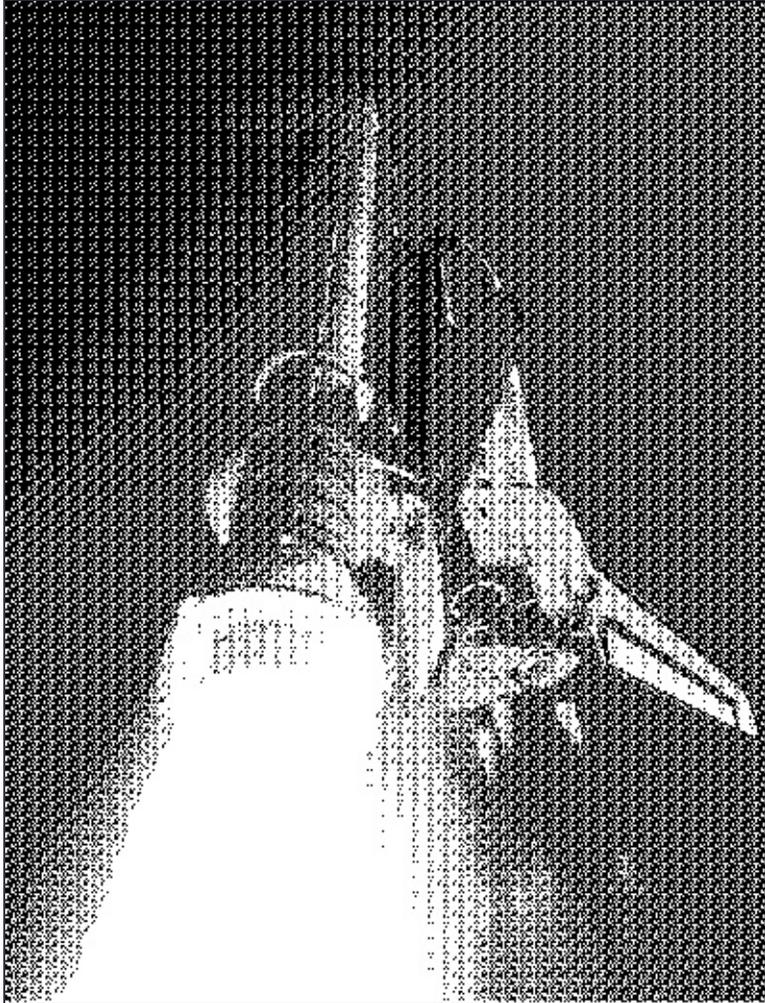
### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim image As XImage = New XImage()
image.SetFile(Server.MapPath("../mypics/Shuttle
theDoc.Rect.String = image.Selection.String
theDoc.AddImage(image)
' Save rendered image as black and white pictur
function
theDoc.Rendering.UseEmbeddedHalftone = False
theDoc.Rendering.DotsPerInch = 50
theDoc.Rendering.ColorSpace = XRendering.Colors
theDoc.Rendering.BitsPerChannel = 1
theDoc.Rendering.DefaultHalftone = "Spot,30,100
theDoc.Rendering.Save(Server.MapPath("Rendering
' Save rendered image as black and white pictur
function
theDoc.Rendering.DefaultHalftone = "Spot,0,100,
theDoc.Rendering.Save(Server.MapPath("Rendering
theDoc.Clear()
```

Example



RenderingHalftoneLine.png



RenderingHalftoneDiamond.png

# DotsPerInch Property

---



Type	Default Value	Read Only	Description
[C#] double			
[Visual Basic] Double	72.0	No	The output resolution in dots per inch (DPI).

This property determines the resolution of the output image.

Because PDF documents specify distances in physical units (typically points) a resolution unit is needed to convert these physical units to pixel based units.

---

## Notes

For example a document 612 points wide rendered at 144 DPI will result in an output image 1224 pixels wide.

Changing this property will change the values of both the [DotsPerInchX](#) and [DotsPerInchY](#) properties.

See the [DefaultHalftone](#) property.

---

## Example



# DotsPerInchX Property

---



Type	Default Value	Read Only	Description
[C#] double			
[Visual Basic] Double	72.0	No	The horizontal output resolution in dots per inch (DPI).

This property determines the horizontal resolution of the output image.

Because PDF documents specify distances in physical units (typically points) a resolution unit is needed to convert these physical units to pixel based units.

---

## Notes

For example a document 612 points wide rendered at 144 DPI will result in an output image 1224 pixels wide.

See the [SaveCompression](#) property.

---

## Example

# DotsPerInchY Property

---



Type	Default Value	Read Only	Description
[C#] double			
[Visual Basic] Double	72.0	No	The vertical output resolution in dots per inch (DPI).

This property determines the vertical resolution of the output image.

Because PDF documents specify distances in physical units (typically points) a resolution unit is needed to convert these physical units to pixel based units.

---

## Notes

For example a document 612 points high rendered at 144 DPI will result in an output image 1224 pixels high.

See the [SaveCompression](#) property.

---

## Example

# DrawAnnotations Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to render annotations on the document background.

Determines whether fields and annotations will be rendered.

PDF fields and annotations are not part of the PDF content. Instead, they are rendered on the document background. You may choose to render these fields or you may not.

---

## Notes

The export of file types like XPS, DOCX and HTML is implemented using the `Doc.Save` method. Because of this reason the `DrawAnnotations` property will determine if annotations are rendered in these formats using the `Doc.Save` method.

The following example shows the effect that this parameter has on the rendering of a document.

## [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/Annotation.pdf"));
theDoc.Rect.Pin = XRect.Corner.TopLeft;
theDoc.Rect.Height = 300;
// Render document with DrawAnnotations (default)
theDoc.Rendering.Save(Server.MapPath("RenderingWithAnnotations.pdf"));
// Render document without DrawAnnotations
theDoc.Rendering.DrawAnnotations = false;
theDoc.Rendering.Save(Server.MapPath("RenderingWithoutAnnotations.pdf"));
```

```
theDoc.Rendering.Save(Server.MapPath("Rendering  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../mypics/Annotatic  
theDoc.Rect.Pin = XRect.Corner.TopLeft  
theDoc.Rect.Height = 300  
' Render document with DrawAnnotations (default  
theDoc.Rendering.Save(Server.MapPath("Rendering  
' Render document without DrawAnnotations  
theDoc.Rendering.DrawAnnotations = False  
theDoc.Rendering.Save(Server.MapPath("Rendering  
theDoc.Clear()
```

Example

Planet	Distance From Sun (miles)	Diameter (miles)	Year Length (days)	Day Length (days)
M	This is Annotation	,030	88	58.00
Venus	67,000,000	7,520	225	225.00
Earth	93,000,000	7,925	365	1.00
Mars	142,000,000	4,210	687	1.00
Jupiter	484,000,000	88,730	4,344	0.40
Saturn	888,000,000	74,975	10,768	0.40
Uranus	1,800,000, 000	31,760	30,660	0.70
	2,800,000,			

RenderingDrawAnnotationsTrue.png

Planet	Distance From Sun (miles)	Diameter (miles)	Year Length (days)	Day Length (days)
Mercury	36,000,000	3,030	88	58.00
Venus	67,000,000	7,520	225	225.00
Earth	93,000,000	7,925	365	1.00
Mars	142,000,000	4,210	687	1.00
Jupiter	484,000,000	88,730	4,344	0.40
Saturn	888,000,000	74,975	10,768	0.40
Uranus	1,800,000,000	31,760	30,660	0.70
	2,800,000,			

RenderingDrawAnnotationsFalse.png

# IccCmyk Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	"standard"	No	The path to the default color profile.

A path to the default CMYK ICC color profile.

The profile that will be used to convert any device CMYK specific file to the device independent working color space.

This is used when the output [ColorSpace](#) is Lab or another device color space. If the [IccOutput](#) indicates that a color profile should output is always device independent. If the IccOutput indicates the profile should be used then the output is always device dependent.

This property can take a path to an icm file. However there are a values you can use. If the property takes the value "device" then color space will be used. If the property takes the value "standard" in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then [IccCmyk](#), [IccGray](#) and IccOutput should also be set to "standard" color profiles. All color spaces are assumed to be device independent spaces.

If this property is set to "device" then IccRgb, IccCmyk, IccGray and IccOutput should also be set to "device". All color spaces are all assumed to be device dependent color spaces.

If this property is set to a file name with no path information, then <windows>\system32\spool\drivers\color" will be searched to loc

The following example shows the effect that this parameter has on

### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(20, 40);
theDoc.FontSize = 96;
// Add CMYK Content
theDoc.Color.String = "200 20 20 20";
theDoc.AddText("Gallia est omnis divisa in part
unam incolunt Belgae, aliam Aquitani, tertiam c
lingua Celtae, nostra Galli appellantur.");
theDoc.Rect.String = theDoc.MediaBox.String;
theDoc.Rendering.DotsPerInch = 36;
theDoc.Rendering.IccCmyk =
Server.MapPath("../mypics/cmyk.icc");
theDoc.Rendering.ColorSpace = XRendering.ColorS
// Save the image
theDoc.Rendering.Save(Server.MapPath("Rendering
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(20, 40)
theDoc.FontSize = 96
' Add CMYK Content
theDoc.Color.String = "200 20 20 20"
theDoc.AddText("Gallia est omnis divisa in part
unam incolunt Belgae, aliam Aquitani, tertiam c
lingua Celtae, nostra Galli appellantur.")
theDoc.Rect.String = theDoc.MediaBox.String
```

Example

```
theDoc.Rendering.DotsPerInch = 36
theDoc.Rendering.IccCmyk =
Server.MapPath("../mypics/cmyk.icc")
theDoc.Rendering.ColorSpace = XRendering.ColorSpace
' Save the image
theDoc.Rendering.Save(Server.MapPath("RenderingIccCmyk.png"))
theDoc.Clear()
```

Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,

RenderingIccCmyk.png

# IccGray Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	"standard"	No	The path to the default Gray ICC color profile.

A path to the default Gray ICC color profile.

The profile that will be used to convert any device Gray specified in the PDF file to the device independent working color space.

This is used when the output `ColorSpace` is Lab or another device independent color space. If the `IccOutput` indicates that a color profile should be used the output is always device independent. If the `IccOutput` indicates that no color profile should be used then the output is always device dependent.

This property can take a path to an icm file. However there are also two special values you can use. If the property takes the value "device" then the device color space will be used. If the property takes the value "standard" then a built in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then `IccRgb`, `IccCmyk`, `IccGray` and `IccOutput` should also be set to "standard" or paths to color

profiles. All color spaces are assumed to be device independent color spaces.

If this property is set to "device" then IccRgb, IccCmyk, IccGray and IccOutput should also be set to "device". All color spaces are all assumed to be device color spaces.

If this property is set to a file name with no path information, then the folder "`<windows>\system32\spool\drivers\color`" will be searched to locate the file.

None.

---

**Example**

# IccOutput Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	"standard"	No	The path to the default output ICC color profile.

A path to the default output ICC color profile.

Rendering can occur to either an device color space or to a device independent color space. By specifying an output color profile you are telling ABCpdf the characteristics of the output device independent color space you wish to use.

This property can take a path to an icm file. However there are also two special values you can use. If the property takes the value "device" then the device color space will be used. If the property takes the value "standard" then a built in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then [IccRgb](#), [IccCmyk](#) and [IccGray](#) should also be set to "standard" or paths to color profiles. All color spaces are assumed to be device independent color spaces.

If this property is set to "device" then [IccRgb](#), [IccCmyk](#) and [IccGray](#) should also be set to "device".

All color spaces are all assumed to be device color spaces.

If this property is set to a file name with no path information, then the folder "`<windows>\system32\spool\drivers\color`" will be searched to locate the file.

---

**Example**

None.

# IccRgb Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	"standard"	No	The path to the default RGB ICC color profile.

A path to the default RGB ICC color profile.

The profile that will be used to convert any device RGB specified in the PDF file to the device independent working color space.

This is used when the output [ColorSpace](#) is Lab or another device independent color space. If the [IccOutput](#) indicates that a color profile should be used the output is always device independent. If the [IccOutput](#) indicates that no color profile should be used then the output is always device dependent.

This property can take a path to an icm file. However there are also two special values you can use. If the property takes the value "device" then the device color space will be used. If the property takes the value "standard" then a built in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set to "standard" or paths to color

profiles. All color spaces are assumed to be device independent color spaces.

If this property is set to "device" then IccRgb, IccCmyk, IccGray and IccOutput should also be set to "device". All color spaces are all assumed to be device color spaces.

If this property is set to a file name with no path information, then the folder "`<windows>\system32\spool\drivers\color`" will be searched to locate the file.

None.

---

**Example**

# Log Property

---



Type	Default Value	Read Only	Description
[C#] string	""	Yes	The log for the last render.
[Visual Basic] String			

During rendering inconsistencies may be detected in the PDF or in the environment.

For example if a PDF contains a corrupt image then it may not be possible to display it. If a PDF contains a reference to a font which is not on the current system then a font substitution may be required.

---

## Notes

ABCpdf will do its best to complete the render even if errors or inconsistencies are found. However it will log these type of issues and make the log available via this property. This can be useful for debugging purposes.

None.

---

## Example



# Metadata Property

---



Type	Default Value	Read Only	Description
[C#] IDictionary<string, string>	null	No	A collection of TIFF tags that should be written to the output file.
[Visual Basic] IDictionary<String, String>			

This property can be used to insert custom tags for TIFF output.

To use this property, create a dictionary and assign it to this property. Then add a sequence of tag names and values for output to file. You will find a full list of TIFF tag names in the TIFF specification.

Most tags, such as the PageName, take a string value. However some tags may take one or more numbers instead. For example the PageNumber tag takes two numbers, the first of which specifies the (zero based) index of the current page and the second of which specifies the total number of pages (zero indicates undetermined). So to indicate the second page of a nine page output, you might use code of the following form.

**[C#]**

```
theDoc.Rendering.Metadata.Add("PageNumber",
```

---

**Notes**

```
"2 9");  
theDoc.Rendering.Metadata.Add("PageName",  
"Page Two");
```

**[Visual Basic]**

```
theDoc.Rendering.Metadata.Add("PageNumber",  
"2 9")  
theDoc.Rendering.Metadata.Add("PageName",  
"Page Two")
```

To indicate that a value should be cleared you can assign an entry with a value of null. This can be useful if you want to, for example, disable the automatic insertion of page name and number tags.

Custom TIFF tags can be indicated using a hash followed by the number of the tag. For example the EXIF standard describes a UserComment tag with ID 37510. To insert such a tag you would use the name "#37510". All custom tags are assumed to be string based.

None.

---

**Example**

# MinimumLineWidth Property

---



Type	Default Value	Read Only	Description
[C#] double			
[Visual Basic] Double	0.25	No?	The minimum stroked line width for output

The minimum stroked line width for output.

This represents the minimum line width for stroked objects in device units (typically pixels). Lines that are narrower than this are set to this minimum line width. This can be used to ensure that narrow lines will still be visible even when drawn on high resolution outputs.

---

## Notes

None.

---

## Example

# NamedSeparation Property

---



Type	Default Value	Read Only	Description
[C#] string	""	No	Named separations.
[Visual Basic] String			

Named separations to be used in addition to the color channels specified in the [ColorSpace](#).

Separate color channels for the named separations are only created if the output type is TIFF and the output color space is CMYK or RGB.

If these conditions are met, ABCpdf will write each separation to a different TIFF page and set the TIFF ink name to the name of that separation.

This property can be either a comma separated list of the named separations, or it can be the keyword 'All' which indicates that all named separations referenced by the page should be output.

For example, the following code will create a tiff file with 5 separated colors, each written to its own page. The ink name for each page will be the name of the separation plane: 'Cyan', 'Magenta', 'Yellow', 'Black' and 'Fluorescent Orange'.

```
doc.Rendering.NamedSeparation
="Fluorescent Orange";
doc.Rendering.ColorSpace =
XRendering.ColorSpaceType.Cmyk;
doc.Rendering.Save("separated.tiff");
```

### **[Visual Basic]**

```
doc.Rendering.NamedSeparation
="Fluorescent Orange"
doc.Rendering.ColorSpace =
XRendering.ColorSpaceType.Cmyk
doc.Rendering.Save("separated.tiff")
```

Note that ABCpdf will not create spot colors if it encounters a page group color space that is different from the specified color space. However in this situation it will still write the process colors individually to the TIFF output.

None.

---

**Example**

# NumTiffStrips Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			Number of strips to generate when writing a tiff file
[Visual Basic] <code>Integer</code>	0	No	

Number of strips to generate when writing a tiff file.

---

## Notes

If the number of strips is set to zero then ABCpdf will determine the number of strips it thinks is appropriate for the file.

---

## Example

None.

# Overprint Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to a

Overprint is a way of combining colors when dealing with subtractive color spaces such as CMYK or DeviceN.

Normally when one color is painted over another it replaces the underlying color. So a black over a cyan becomes yellow. If Overprint is set then the top color will be placed over the top of the base color. So black over cyan becomes both black and cyan inks. This can be useful for printing purposes.

---

## Notes

In ABCpdf overprint is only applied when the Overprint property is set to true. If the output `ColorSpace` is CMYK, the `IccOutput` is "device", the PDF dictionary entry `/Overprint` indicates that overprint mode should be applied (see the [Adobe PDF Reference](#) for details) and no transparency has been detected.

The following example shows the effect that this parameter has on the output.

## [C#]

```
Doc theDoc = new Doc();
// Open document with overprint
theDoc.Read(Server.MapPath("../mypics/Overprint.pdf"));
// Render the document (we need to go to CMYK for overprint)
```

```

theDoc.Rendering.ColorSpace = XRendering.ColorSpace.DeviceRGB;
theDoc.Rendering.Overprint = true;
theDoc.Rendering.IccRgb = "device";
theDoc.Rendering.IccGray = "device";
theDoc.Rendering.IccCmyk = "device";
theDoc.Rendering.IccOutput = "device";
// Put image back into another PDF and render to
RGB
// (so that we can see the effect of the overprint)
Doc theRgb = new Doc();
theRgb.AddImageData(theDoc.Rendering.GetData("Image1.pdf"));
theDoc.Clear();
theRgb.Rendering.DotsPerInch = 36;
theRgb.Rendering.ColorSpace = XRendering.ColorSpace.DeviceRGB;
theRgb.Rendering.Save(Server.MapPath("Rendering/Overprint.pdf"));
theRgb.Clear();

```

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
' Open document with overprint
theDoc.Read(Server.MapPath("../mypics/Overprint.pdf"))
' Render the document (we need to go to CMYK for overprint)
theDoc.Rendering.ColorSpace = XRendering.ColorSpace.DeviceCMYK
theDoc.Rendering.Overprint = True
theDoc.Rendering.IccRgb = "device"
theDoc.Rendering.IccGray = "device"
theDoc.Rendering.IccCmyk = "device"
theDoc.Rendering.IccOutput = "device"
' Put image back into another PDF and render to
' (so that we can see the effect of the overprint)
Dim theRgb As Doc = New Doc()
theRgb.AddImageData(theDoc.Rendering.GetData("Image1.pdf"));
theDoc.Clear()
theRgb.Rendering.DotsPerInch = 36
theRgb.Rendering.ColorSpace = XRendering.ColorSpace.DeviceRGB
theRgb.Rendering.Save(Server.MapPath("Rendering/Overprint.pdf"));
theRgb.Clear();

```

Example

```
theRgb.Clear()
```



RenderingOverprint.png

# PaletteSize Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	256	No	The number of colors in the palette for indexed color output.

This property is used to specify the number of colors in the palette when using the "Indexed" color space. The valid range for this property is between 2 and 256.

---

## Notes

Black and white is always included in the palette. So a `PaletteSize` of two will always produce a black and white image output.

# ResizeImages Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to resize images for vector output.

Whether to resize images for vector output.

If this option is set then raster images will be resized to the current resolution before embedding in the vector format. This can reduce file size when producing formats like EMF that are destined for a particular printer at a particular resolution.

---

## Notes

Currently this option only takes effect for EMF output. However in the future it may be extended to other vector formats such as Postscript.

None.

---

## Example

# SaveAlpha Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to save the alpha channel in the rendered output.

This property determines whether the Save method includes an alpha channel in the rendered output.

Only some types of output support alpha. These are:

- PNG
- BMP
- TIFF (Grayscale, RGB and CMYK)
- PSD (Photoshop)
- Bitmap (via the [GetBitmap](#) function)

---

## Notes

By including an alpha channel in your output you can identify areas that are transparent and areas on which no drawing has taken place.

The following example shows the effect that this parameter has on a PDF. We create a PDF with some text on it. We then render the PDF with the SaveAlpha property set to true. We then add the transparent image into a new PDF with a blue background. The image shows through where the image is transparent.

## [C#]

```
Doc theDoc = new Doc();
// Add some text
theDoc.FontSize = 196;
theDoc.TextStyle.HPos = 0.5;
theDoc.TextStyle.VPos = 0.3;
theDoc.AddText("Hello World");
// Render the PDF with alpha
theDoc.Rendering.SaveAlpha = true;
Bitmap alphaBitmap = theDoc.Rendering.GetBitmap();
// Create a blue PDF
theDoc = new Doc();
theDoc.Color.String = "0 0 255";
theDoc.FillRect();
// Add the transparent Bitmap into the PDF
// so that the underlying blue can show through
theDoc.AddImageBitmap(alphaBitmap, true);
// Save render of pdf
theDoc.Rendering.Save(Server.MapPath("Rendering"));
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
' Add some text
theDoc.FontSize = 196
theDoc.TextStyle.HPos = 0.5
theDoc.TextStyle.VPos = 0.3
theDoc.AddText("Hello World")
' Render the PDF with alpha
theDoc.Rendering.SaveAlpha = True
Dim alphaBitmap As Bitmap = theDoc.Rendering.GetBitmap()
' Create a blue PDF
theDoc = New Doc
theDoc.Color.String = "0 0 255"
theDoc.FillRect()
' Add the transparent Bitmap into the PDF
```

Example

```
' so that the underlying blue can show through  
theDoc.AddImageBitmap(alphaBitmap, True)  
' Save render of pdf  
theDoc.Rendering.Save(Server.MapPath("Rendering  
theDoc.Clear()
```



RenderingSaveAlpha.png

# SaveAppend Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	false	No	Whether to append to (rather than overwrite) existing image files.

This property determines whether the Save method appends to (rather than overwrites) existing image files.

If this property is set to true and a file already exists then the Save method appends the rendered image to the existing file rather than just overwriting it. In this way you can build up multi-page images.

---

## Notes

For in-memory operations you can use this property set to true in conjunction with a MemoryStream and the Save overload that supports streams.

This property is used for TIFF output only.

See the [SaveCompression](#) property.

---

## Example



# SaveCompression Property

---



Type	Default Value	Read Only	Description
[C#] Compression	None	No	The preferred compression method.
[Visual Basic] Compression			

This property determines the preferred compression method.

Not all file formats support all compression methods. Not all compression methods support all color spaces. This is why this property indicates a preferred method.

The Compression enumeration accepts the following values:

- None [Uncompressed]
- G3 [G3 Black and White Fax Compression]
- G4 [G4 Black and White Fax Compression]
- LZW [LZW Compression]
- Jpeg [Tiff Jpeg Compression]
- AdobeFlate [Adobe Flate Compression]
- Flate [Flate Compression]
- PackBits [Macintosh PackBits]

---

## Notes

In general this property is most useful when producing output in formats like TIFF which support a range of color spaces and compression methods.

The following example shows how to render a PDF into a multiple compressed Fax TIFF using different vertical and horizontal reso

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../Rez/spaceshuttle
// set up the rendering parameters
theDoc.Rendering.ColorSpace =
XRRendering.ColorSpaceType.Gray;
theDoc.Rendering.BitsPerChannel = 1;
theDoc.Rendering.DotsPerInchX = 200;
theDoc.Rendering.DotsPerInchY = 400;
// loop through the pages
int n = theDoc.PageCount;
for (int i = 1; i <= n; i++) {
    theDoc.PageNumber = i;
    theDoc.Rect.String = theDoc.CropBox.String;
    theDoc.Rendering.SaveAppend = (i != 1);
    theDoc.Rendering.SaveCompression =
XRRendering.Compression.G4;
    theDoc.Rendering.Save(Server.MapPath("fax.tif
}
theDoc.Clear();
```

Example

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../Rez/spaceshuttle
' set up the rendering parameters
theDoc.Rendering.ColorSpace =
XRRendering.ColorSpaceType.Gray
theDoc.Rendering.BitsPerChannel = 1
theDoc.Rendering.DotsPerInchX = 200
theDoc.Rendering.DotsPerInchY = 400
```

```
' loop through the pages
Dim n As Integer = theDoc.PageCount
For i As Integer = 1 To n
    theDoc.PageNumber = i
    theDoc.Rect.String = theDoc.CropBox.String
    theDoc.Rendering.SaveAppend = (i <> 1)
    theDoc.Rendering.SaveCompression =
XRendering.Compression.G4
    theDoc.Rendering.Save(Server.MapPath("fax.tif"))
Next
theDoc.Clear()
```

# SaveQuality Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	75	No	The output file quality compression.

This property determines the quality of the output image.

It can take values between 0 and 100 ranging from lowest to high

---

## Notes

This property is used for JPEG and JPEG 2000 output only.

The following example shows the effect that this parameter has on

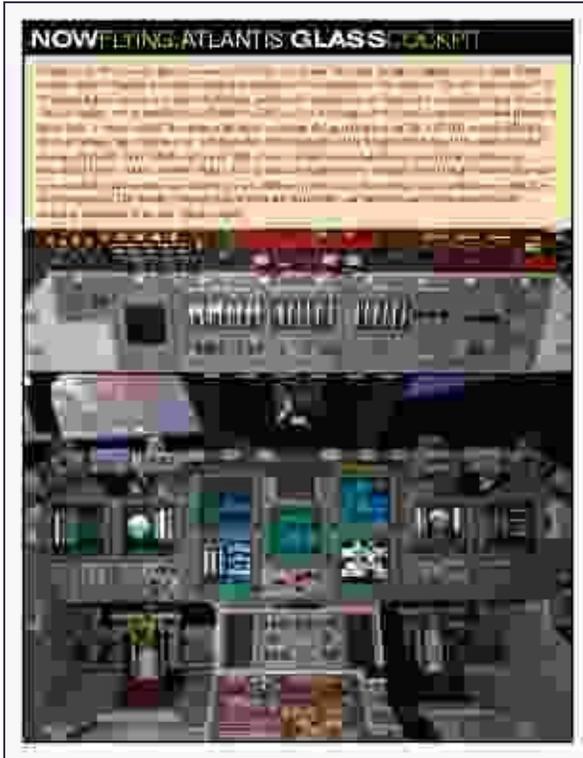
### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/SpaceShut
theDoc.Rendering.DotsPerInch = 36;
// Save at low quality
theDoc.Rendering.SaveQuality = 5;
theDoc.Rendering.Save(Server.MapPath("Rendering
// Save at high quality
theDoc.Rendering.SaveQuality = 75;
theDoc.Rendering.Save(Server.MapPath("Rendering
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../mypics/SpaceShut  
theDoc.Rendering.DotsPerInch = 36  
' Save at low quality  
theDoc.Rendering.SaveQuality = 5  
theDoc.Rendering.Save(Server.MapPath("Rendering  
' Save at high quality  
theDoc.Rendering.SaveQuality = 75  
theDoc.Rendering.Save(Server.MapPath("Rendering  
theDoc.Clear()
```

Example



RenderingQuality5.jpg [file size 9.26 KB]

## NOW FLYING: ATLANTIS' GLASS COCKPIT

Flying for the first time on Atlantis' mission on STS-101, eleven new 300 color flat-panel display screens in the Shuttle cockpit replace 32 gauges and electromechanical displays and four cathode-ray tube displays. The new "glass cockpit" is 75 percent lighter and uses less power than before, and its color displays provide a more pilot's opinion of key functions. The new cockpit will be installed in all Shuttles by 2002, and it sets the stage for the next cockpit improvement planned to fly by 2005: a "smart cockpit" that reduces the pilot's workload during critical periods. On STS-101, Atlantis will fly as the most updated Space Shuttle ever, with more than 400 new modifications incorporated during a 10-month period at Boeing's Palmdale, Calif., Shuttle factory in 1999. Atlantis' airlock was relocated to the payload bay to prepare for International Space Station assembly flights; the communication system was updated; several weight reduction measures were installed; enhancements were made to provide additional protection to the cooling system; and the crew cabin floor was strengthened. The Shuttle Columbia is at the Palmdale factory this year, receiving many of the same upgrades, including installation of the new "glass cockpit."



RenderingQuality5.jpg [file size 37.3 KB]

# UseEmbeddedHalftone Property

---



Type	Default Value	Read Only	Description
[C#] bool			Whether to use embedded halftones.
[Visual Basic] Boolean	True	No	

Specifies whether to use embedded halftones.

---

## Notes

If embedded halftones are not used then the [DefaultHalftone](#) will be used for all halftoning.

---

## Example

See the [DefaultHalftone](#) property.

# CompressObjects Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to use object stream compression to reduce file size.

This property determines if object stream compression should be used.

Object stream compression allows groups of simple atoms to be stored in a separate stream (an ObjStm) and compressed. This can reduce the output file size.

Typically the PDF specification is efficiently organized, atoms are small and not many are required. Thus the overhead associated with uncompressed atoms is typically minimal. Since not all PDF readers support object streams we default to avoiding this type of structure. In particular, older versions of Acrobat (eg 8) are not happy reading documents which incorporate both object streams and linearized content.

In a few cases the PDF specification has not been efficiently organized. The most notable situation in which this occurs is in the case of Tagged PDF - often required for accessibility. The structures required for tagging can result in a multitude of

---

**Notes**

similar atoms and using object stream compression can make a significant difference to the size of output. It is perhaps notable that tagged PDF was introduced in PDF 1.4 and then object streams were introduced immediately afterwards in PDF 1.5. File size reduction will vary considerably depending on content but 10% might be typical.

The other situation in which this option can make a notable difference is in the case of very small PDF documents. In documents measuring ten or twenty kilobytes, atoms can make up a significant proportion of the document size. In these situations you might be able to reduce the size by perhaps 30% simply by enabling this option.

Object stream compression is incompatible with **Incremental** update. If this flag is set then object stream compression will not be used.

**Future Changes.** As PDF documents become more accessible there may be more reason to apply object stream compression. Similarly as PDF readers improve there may be less reason not to use it. As such it is possible that the default value of this property might be changed. If you are relying on a particular value you should set it in your code.

None.

---

**Example**



# EmbeddedGraphics Property

---



Type	Default Value	Read Only
[C#] EmbeddedGraphics	EmbeddedGraphicsType.Automatic	No
[Visual Basic] EmbeddedGraphics		

This property specifies how PDF vector graphics such as paths are implemented. Different exporters can take advantage of different formats. However, only the DOCX and HTML exporters currently embedded graphic output.

The EmbeddedGraphicsType enumeration may take the following

- Automatic - selects a sensible default value based on the export format and content
- None - no image output
- Image - graphics are rendered to a bitmapped image (valid for DOCX and HTML)
- Vml - graphics are rendered using VML (valid for DOCX only)
- HtmlCanvas - graphics are rendered using the HTML 5 Canvas element (valid for HTML only)

Please note the following limitations inherent in the VML specific. VML does not support the non-zero fill rule so VML paths may not be identical to PDF paths. VML does not support sophisticated dashed or dotted lines are approximated. VML does not specify l

---

**Notes** bitmapped images and vector paths should interact so the Z-order is dependent on the VML display implementation - a graphic which is drawn after an image in a PDF may vanish under the image using certain VML viewers.

The HTML5 Canvas element is a blank element on which JavaScript can draw. For older versions of Internet Explorer, the canvas element is automatically converted into VML via [ExplorerCanvas](#).

Please note the following limitations inherent in the HTML Canvas specification. The even-odd fill rule is not supported so Canvas paths may not be identical to PDF paths. The HTML Canvas does not support dashing so dashed or dotted lines are not displayed. Because VML supports the non-zero fill rule, paths will look different in older versions of Internet Explorer which are working via the ExplorerCanvas VML conversion.

If an invalid type is specified for a specific export format, an exception will be thrown.

None.

---

**Example**

# FileExtension Property

---



Type	Default Value	Read Only	Description
[C#]string [Visual Basic]String	null	No	Gets or sets the file extension for the target when it is not otherwise specified.

This property is used to specify an export format when (and only when) the specification of the destination does not provide a file extension, such as a Stream or an array of bytes.

---

## Notes

The supported values are "xps" and "swf".

None.

---

## Example

# Folder Property

---



Type	Default Value	Read Only	Description
[C#]string			The folder where to save additional files (images, fonts, etc).
[Visual Basic]String	null	No	

This property specifies the folder where to store additional data such as images and fonts. it is only used when exporting documents to HTML. It is ignored otherwise.

If you set a relative path, it will be relative to the location of the output path passed to Doc.Save(). If you set an absolute path, the output html will contain absolute URLs.

---

## Notes

If you are saving to a stream and specify a relative path, then the path is relative to the current directory.

If you are saving to file and do not specify a folder, then a folder with the same name as the output file will be created in the directory where the output file is located.

None.

---

**Example**

# FontSubstitution Property

---



Type	Default Value
[C#]FontSubstitutionType	FontSubstitutionType.Automatic
[Visual Basic] FontSubstitutionType	

This property specifies when fonts are substituted in export operations. DOCX and HTML exporters currently support this property. It is not supported by other exporters, for example, the XPS exporter.

The FontSubstitutionType enumeration may take the following values:

---

## Notes

- Automatic - selects a sensible default value based on the export options.
- Always - always substitute fonts with those installed on the rendering system.
- Missing - only substitute fonts which are missing or could not be found.

None.

---

## Example

# IDConstant Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to assign a constant file version identifier.

This property determines if the file identifiers should be constant rather than unique.

PDF documents have two unique file identifiers. One is a file specific identifier which stays constant throughout the life of the file. The other is a file version identifier which is updated every time the document is saved.

---

## Notes

This property allows you to ensure that the same identifiers are always used. This can be useful for debugging if you need to do a binary level comparison of files and you need to ensure that random elements have been eliminated.

None.

---

## Example



# IDHexadecimal Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to assign non-ASCII file identifiers.

This property determines if file identifiers should be hexadecimal or ASCII.

File identifiers take the form of strings. The characters in these strings can span any range. However some applications demand that identifiers are restricted to the ASCII range while some demand that identifiers contain characters outside the ASCII range.

---

## Notes

Setting this value to false will allow you to restrict newly generated identifiers to the ASCII range.

None.

---

## Example

# IDUpdate Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to update the file version identifier.

This property determines if the file version identifier should be updated.

PDF documents have two unique file identifiers. One is a file specific identifier which stays constant throughout the life of the file. The other is a file version identifier which is updated every time the document is saved.

---

## Notes

This property allows you to suppress the version update.

None.

---

## Example

# Incremental Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to use incremental update to preserve an audit trail.

This property determines if incremental update should be used.

Incremental update leaves the structure of any original document intact and appends any changes to the end of the file.

Because the original PDF is unchanged it is possible to revert back to the state before the changes were applied. Because only updated objects are written to disk it can result in faster write times.

---

## Notes

However most importantly you have to use incremental update if you wish to preserve any signed signature fields present in the document. Indeed after any signature field is signed you need to commit changes to file using incremental update.

Incremental Update is incompatible with the [Linearization](#), [Remapping](#) and [CompressObjects](#) options. As such it will override these settings.

---

**Example**

None.

# Linearize Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to linearize the output for fast web viewing.

This property determines if output should be linearized for fast viewing over the web.

Linearized PDFs are organized in a special way to enable efficient access over the web. The output is valid PDF in all ways but it is structured to allow web viewers to access only the portions of the PDF that they need.

From a practical point of view this means that a Linearized PDF opens instantly rather than requiring that the entire PDF is downloaded before it can be seen.

---

## Notes

Adobe sometimes refer to Linearization as Fast Web View.

There is a performance and memory overhead associated with linearization. However this is something that you should normally only notice when handling huge documents.

---

Example

None.

# Remap Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to reduce size by remapping objects.

This property determines if output size should be reduced by remapping objects.

PDF documents consist of a sequence of numbered objects. If objects have been deleted there may be gaps in this sequence. Gaps can lead to PDFs which are larger than necessary.

---

## Notes

If this property is set to true then, when the document is saved, remapping will occur to eliminate these gaps.

Note that if the [Linearize](#) property is set to true then remapping will occur no matter the value of this property.

None.

---

## Example



# SaveQuality Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>	50	No	The XPS output quality for lossy compression.
[Visual Basic] <code>Integer</code>			

This property determines the quality of output images in XPS export.

Under some circumstances it can be necessary to decompress and recode JPEG and JPEG 2000 images that are stored in a PDF. This typically occurs when exporting to another format which does not support exactly the same features as are supported in PDF.

---

## Notes

This property determines the output quality for the recompression of such images. This is only used during export to XPS.

---

## Example

None.

# Template Property

---



Type	Default Value	Read Only	Description
[C#]string			
[Visual Basic]String	null	No	The path to the template file.

This property specifies the template file, which provides some format-specific data essential to the usefulness of the output when saving in certain formats such as SWF.

When saving in SWF, if the property is null, the output contains the content of the current rectangle of the current PDF page.

When saving in SWF, if the property is the special value [XSaveTemplateData.Template\\_OnePagePerFrame](#), each frame of the output contains the content of a PDF page. The contents of pages of different sizes will be centered. You can optionally specify settings that are normally provided by a template file. Such settings are separated by NUL and the name-value separator is colon. The values of settings are specified in string in the invariant culture.

---

## Notes

Settings for XSaveTemplateData.Template_OnePagePerFrame			
Name	Type	Default Value	Description
	[C#]ushort	64 (¼)	The frame rate in

FrameRate	[Visual Basic] UShort	frame per second)	1/256 frames per second.
-----------	--------------------------	-------------------	--------------------------

See the example project for how to use a SWF template file.

Here we specify one page per frame and 2 frames per second format.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.pd
theDoc.SaveOptions.Template =
XSaveTemplateData.Template_OnePagePerFrame +
"\0FrameRate:512";
theDoc.Save(Server.MapPath("swfsave.swf"));
```

Example

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample.pd
theDoc.SaveOptions.Template =
XSaveTemplateData.Template_OnePagePerFrame _
& ControlChars.NullChar & "FrameRate:512" _
theDoc.Save(Server.MapPath("swfsave.swf"))
```

# TemplateData Property

---



Type	Default Value	Read Only	Description
[C#]XSaveTemplateData			
[Visual Basic]XSaveTemplateData	null	No	The template data.

---

## Notes

This property specifies the template data that is not obtained from [Template](#) and which provides some format-specific data essential to the usefulness of the output when saving in certain formats such as SWF.

---

## Example

None.

# WritePageSeparator Property

---



Type	Default Value	Read Only
[C#]XSaveOptions.PageSeparatorMethod		
[Visual Basic] XSaveOptions.PageSeparatorMethod	null	No

This delegate is called during HTML export.

If it is null, a default page separator is produced for each page. You can customize the page separator by setting this property.

The definition of the XSaveOptions.PageSeparatorMethod delegate is as follows.

## [C#]

```
delegate void PageSeparatorMethod(int pageNum,
    ExportArgs e);
```

---

## Notes

## [Visual Basic]

```
Delegate Sub PageSeparatorMethod(pageNum As Integer, e As ExportArgs)
```

pageNum is the page number, starting with 1.

The page separator is to be written to e.Writer. e.Writer is an XmlWriter for HTML export.

The following example shows how to customize the page separator.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.poc
theDoc.SaveOptions.WritePageSeparator = delegat
pageNum, XSaveOptions.ExportArgs e) {
    XmlWriter writer = (XmlWriter)e.Writer;
    if(pageNum > 1) {
        writer.WriteStartElement("hr");
        writer.WriteEndElement();
    }
    writer.WriteStartElement("div");
    writer.WriteAttributeString("align", "right")
    writer.WriteString(string.Format("Page {0}",
pageNum));
    writer.WriteFullEndElement();
};
theDoc.Save(Server.MapPath("PageSeparator.htm")
theDoc.Dispose();
```

**Example**

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample.poc
theDoc.SaveOptions.WritePageSeparator = Address
WriteSeparator
theDoc.Save(Server.MapPath("PageSeparator.htm")
theDoc.Dispose()
```

```
Private Shared Sub WriteSeparator(pageNum As
```

```
Integer, e As ExportArgs)
    Dim writer As XmlWriter = CType(e.Writer,
    XmlWriter)
    If pageNum > 1 Then
        writer.WriteStartElement("hr")
        writer.WriteEndElement()
    End If
    writer.WriteStartElement("div")
    writer.WriteAttributeString("align", "right")
    writer.WriteString(String.Format("Page {0}",
    pageNum))
    writer.WriteFullEndElement()
End Sub
```



# SetMeasureResolution Function

---

Sets the measurement resolutions.

## [C#]

```
void SetMeasureResolution(double dpi)  
void SetMeasureResolution(double dpiX, double dpiY)
```

---

## Syntax

## [Visual Basic]

```
Sub SetMeasureResolution(dpi As Double)  
Sub SetMeasureResolution(dpiX As Double, dpiY As Double)
```

---

## Params

Name	Description
dpi	The new measurement resolution in DPI.
dpiX	The new horizontal measurement resolution in DPI.
dpiY	The new vertical measurement resolution in DPI.

Sets the measurement resolutions, which affects the measurement of certain output formats that does not support physical sizes. SWF measurements are pixel-based while PDF measurements are in points (1/72 of an inch). This specifies how many pixels in SWF represents 1 point in PDF represents.

---

## Notes

The values do not affect the pixel sizes of raster-image content. If one of the values (dpiX or dpiY) is invalid, the other value may be used.

both values.

For SWF, these values are used only if [XSaveOptions.TemplateData](#)

Here we use 72 DPI so that 1 inch in PDF becomes 72 pixels

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample.pdf"));
theDoc.SaveOptions.TemplateData = new
XSaveTemplateData();
theDoc.SaveOptions.TemplateData.SetMeasureF
theDoc.Save(Server.MapPath("swfsave.swf"));
```

---

Example

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample.pdf"))
theDoc.SaveOptions.TemplateData = New XSave
theDoc.SaveOptions.TemplateData.SetMeasureF
theDoc.Save(Server.MapPath("swfsave.swf"))
```

# ImageDisplaySmoothing Property

---



Type	Default	Read Only	Description
[C#] RasterSmoothing	Default	No	The smoothing for displaying images.
[Visual Basic] RasterSmoothing			

This property specifies the smoothing for displaying images.

It can take any of the following values:

- Default – specifies the default algorithm.
- Alias – specifies no anti-aliasing.
- AntiAlias – specifies anti-aliasing.

---

## Notes

Some output formats support different smoothing types for displaying images.

For SWF, this affects the FillStyleType field of the FILLSTYLE structure for images.

None.

---

## Example



# JpegQuality Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	-1	No	The JPEG quality level.

This property specifies the quality level when encoding images in JPEG. The value ranges from 0 (the worst quality, smaller output) to 100 (the best quality, larger output). Setting the property to -1 is considered as not specifying the quality level, and an appropriate default quality level is used.

---

## Notes

None.

---

## Example

# MeasureDpiX Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	Yes	The horizontal measurement resolution.

To change the value, use [SetMeasureResolution](#).

---

## Notes

None.

---

## Example

# MeasureDpiY Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	Yes	The vertical measurement resolution.

To change the value, use [SetMeasureResolution](#).

---

## Notes

None.

---

## Example

# ReencodeJpeg Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether JPEG images are re-encoded in JPEG.

This property is effectual where both the source and the destination formats are JPEG. It specifies whether JPEG encoding occurs. When it is false, the source is directly used as the output without re-encoding. When it is true, the source is re-encoded in JPEG.

---

## Notes

Reencoding allows a JPEG image to use a different [quality level](#). However, reencoding a JPEG image at the same/a higher quality level does not improve the image quality and may degrade it.

None.

---

## Example



# InstallLicense Method

---

Install a license.

## [C#]

```
bool InstallLicense(string  
license)
```

---

## Syntax

## [Visual Basic]

```
Function InstallLicense(license  
As String) As Boolean
```

---

## Params

Name	Description
license	The license to install.
return	True if a license is installed, otherwise false.

Use this method to install a license. Call this method at application startup before any ABCpdf objects have been created. You only need to call this method once though calling it additional times will not cause problems.

---

## Notes

Any license installed using this method will remain available to the current process (or application pool) until it unloads.

---

**Example**

See [Manual Installation](#).



# InstallRedistributionLicense Method

---

Install a redistribution license.

## [C#]

```
bool  
InstallRedistributionLicense(string  
license)
```

---

## Syntax

## [Visual Basic]

```
Function  
InstallRedistributionLicense(license  
As String) As Boolean
```

---

## Params

Name	Description
license	The license to install.
return	True if a license is installed, otherwise false.

Use this method to install a redistribution license. Call this method at application startup before any ABCpdf objects have been created. You only need to call this method once though calling it additional times will not cause

---

**Notes**

problems.

Any license installed using this method will remain available to the current process (or application pool) until it unloads.

---

**Example**

See [Manual Installation](#).



# InstallSystemLicense Method

---

Install a system license.

## [C#]

```
bool InstallSystemLicense(string license)
```

---

## Syntax

## [Visual Basic]

```
Function  
InstallSystemLicense(license As  
String) As Boolean
```

---

## Params

Name	Description
license	The license to install.
return	True if a license is installed, otherwise false.

Use this method to install a system license. Call this method at application startup before any ABCpdf objects have been created.

This method saves the license for use on a system-wide basis. So, once a license is successfully installed, it will stay on the system and be picked up by ABCpdf automatically.

---

**Notes**

Licenses are specific to the process model so licenses installed for with 32-bit ABCpdf will not be available to 64-bit ABCpdf and vice versa. Indeed, a license may be installed successfully without being valid for the current process so it is a good idea to check [LicenseValid](#) after installing a system license.

For security reasons, you should not use this method unless you have complete control of the system on which you are installing.

---

**Example**

See [Manual Installation](#).



# InstallTrialLicense Method

---

Install a trial license

## [C#]

```
bool InstallTrialLicense(string license)
```

---

## Syntax

## [Visual Basic]

```
Function  
InstallTrialLicense(license As  
String) As Boolean
```

---

## Params

Name	Description
license	The license to install.
return	True if a license is installed, otherwise false.

Use this method to install a trial license. Call this method at application startup before any ABCpdf objects have been created. You only need to call this method once though calling it additional times will not cause problems.

---

## Notes

Any license installed using this method will remain available to the current process (or

application pool) until it unloads.

---

**Example**

See [Manual Installation](#).



## Register Method

---

Register and install a trial license.

### [C#]

```
void Register()
```

---

### Syntax

### [Visual Basic]

```
Sub Register()
```

---

### Params

Name	Description
return	None.

Use this method to register ABCpdf and install a trial license if no license is installed.

You should never need to call this method as both the process of installing and the process of running the PDFSettings application will automatically register ABCpdf.

---

### Notes

If for some reason the registration fails, all you need to do is run the PDFSettings application as Administrator.

---

**Example**

None.



# SetConfigSection Method

---

Set the application configuration section.

## [C#]

```
void SetConfigSection(ConfigSection section
```

---

### Syntax

## [Visual Basic]

```
Sub SetConfigSection(section As ConfigSecti
```

Name	Description
section	The ABCpdf configuration section.
return	None.

---

### Params

Use this method to change ABCpdf's configuration section (

Normally, you will not need to call this method because ABC detects the presence of the default configuration file. It does `System.Configuration.ConfigurationManager.GetSection("A` So, as long as you have a valid `ABCpdf10.Section` in your c there should be no need to call this method.

For local applications, configuration files are normally stored in the application folder as the application and are called `<application.exe>.c` `<application.exe>` is the file name of the application. For web applications, the configuration section is stored in `Web.config`.

---

**Notes**

The ABCpdf section name should be ABCpdf10.Section. The WebSupergoo.ABCpdf10.ConfigSection. Refer to the example details. ConfigSection is an opaque class that contains an array of Preferences.

Each preference has a Key and a Value.

If a preference is not found in the configuration file, it is read from the registry.

The preference key is the same as the registry key name. For example, `Key="LogErrors" Value="1" />` is equivalent to a DWORD registry value named LogErrors and with Value set to 1. Refer to [Registry Keys](#) for further details. Make sure that the value matches the type of the [Registry Keys](#).

Warning: changing preferences from ABCpdf events/callbacks that use those preferences may cause problems.

Here is an example of a .config file that contains a valid ABCpdf section. The illustrated ABCpdf preferences are LogErrors, Password, and Password. Other preferences will be read from the Registry.

---

**Example**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="ABCpdf10.Section"
type="WebSupergoo.ABCpdf10.ConfigSection, A
allowLocation="true"
allowDefinition="Everywhere"
allowExeDefinition="MachineToLocalUser"
overrideModeDefault="Allow"
restartOnExternalChanges="true"
requirePermission="true" />
  </configSections>
  <ABCpdf10.Section>
```

```
<Preferences>
  <clear />
  <add Key="TempDirectory" Value="C:\Te
  <add Key="LogErrors" Value="1" />
  <add Key="MakeURLsUnique" Value="0" /
  <add Key="MSOfficeShow" Value="1" />
</Preferences>
</ABCpdf10.Section>
</configuration>
```

# Key Property

---



Type	Default Value	Read Only	Description
[C#]string			A trial license key used for remote deployment.
[Visual Basic]String	None	Yes	

This property contains a trial license that you can use for remote deployment.

---

**Notes**

If ABCpdf has not yet been [registered](#), this property will be empty.

---

**Example**

None.

# LicenseDescription Property

---



Type	Default Value	Read Only	Description
[C#]string			
[Visual Basic]String	None	Yes	The current license for the software.

This property describes the current license for the software in human readable format. Typically, this will be trial, standard or professional.

---

## Notes

---

## Example

None.

# Licensee Property

---



Type	Default Value	Read Only	Description
[C#]string			
[Visual Basic]String	None	Yes	The current licensee for the software.

This property retrieves the licensee for the software and presents it in a human readable format.

The licensee is the legal entity to which ABCpdf is registered. The licensee is selected at the point at which a license is purchased and that license is specific to that entity. Licensees are normally companies but may be individuals.

---

## Notes

---

## Example

None.

# LicenseType Property

---



Type	Default	Read Only	Description
[C#] LicenseType	n/a	Yes	The current feature level for the software.
[Visual Basic] LicenseType			

The LicenseType enumeration may take the following values:

---

## Notes

- None
- TimePeriod
- Standard
- Professional

---

## Example

None.

# LicenseValid Property

---



Type	Default Value	Read Only	Description
[C#]bool			
[Visual Basic] Boolean	None	Yes	Whether the current license is valid.

This property determines whether the current license is valid.

---

## Notes

After installing a license, you may wish to check that it is installed and valid using this property.

---

## Example

None.

# LogErrors Property

---



Type	Default Value	Read Only	Description
[C#]bool			
[Visual Basic] Boolean	False	No	Whether to log errors in the Application Event Log.

This property determines whether to log errors in the Application Event Log.

---

## Notes

---

## Example

None.

# Version Property

---



Type	Default Value	Read Only	Description
[C#]int			The software version.
[Visual Basic] Integer	None	Yes	

This property gets the software version.

---

Notes

---

Example

None.



# ToString Function

---

Returns a string representation of the object.

## [C#]

```
override string ToString()
```

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Syntax

Name	Description
return	The string representation of the object.

---

### Params

This method returns the string value of the object. This is equivalent to reading the [String](#) property of the object.

---

### Notes

None.

---

### Example



# Ascender Property

---



Type	Default	Read Only	Description
[C#]int [Visual Basic] Integer	-1	No	An adjustment to allow the text ascender to coincide with the top of the text area.

The distance between the line spacing below the top of the text frame and the first baseline.

Because [LineSpacing](#) is always applied at the top, the top of the text rectangle needs to be shifted up by [LineSpacing](#) for better control of the position of the first line.

---

## Notes

This property is measured in 1000ths of the font size. When it is -1, default spacing will be applied.

---

## Example

None.

# Bold Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to apply a synthetic bold effect.

This property determines whether a synthetic bold effect is applied to text.

It is generally better to specify a bold typeface rather than synthesize a bold effect using the current typeface.

---

## Notes

However under some circumstances this may not be possible and you may prefer to apply a synthetic bold effect.

In this example we add some bold text to a document.

### [C#]

```
Doc theDoc = new Doc();
string theText;
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
```

```
theDoc.TextStyle.Size = 96;
theDoc.TextStyle.Bold = true;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylebold.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur."
theDoc.Rect.Inset(20, 40)
theDoc.TextStyle.Size = 96
theDoc.TextStyle.Bold = True
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("stylebold.pdf"))
theDoc.Clear()
```

Example

**Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,**

stylebold.pdf

# CharSpacing Property

---



Type	Default	Read Only	Description
[C#]double			
[Visual Basic]Double	0.0	No	The inter-character spacing.

This property controls the spacing between each character. It is sometimes called tracking but should not be confused with [kerning](#) which is slightly different.

Each character in a string of text has a width which is used for positioning the next character. The CharSpacing is added to the width of each character.

---

## Notes

In the horizontal writing mode, specifying a positive value has the effect of stretching out the text. Specifying negative values has the effect of condensing the text. In the vertical writing mode, positive values condense the text, and negative values stretch out the text. See the [FontObject.WritingMode](#) property.

Because this property is measured as an absolute value the visual effect will be greater if your text is smaller.

In this example we add three blocks of text to a document. The first block uses the default spacing. The second block uses a

positive value to stretch out the text. The last block uses a negative value to condense the text.

### [C#]

```
Doc theDoc = new Doc();
theDoc.TextStyle.Size = 96;
theDoc.AddText("Zero CharSpacing");
theDoc.Rect.Move(0, -300);
theDoc.TextStyle.CharSpacing = 10;
theDoc.AddText("Positive CharSpacing");
theDoc.Rect.Move(0, -300);
theDoc.TextStyle.CharSpacing = -10;
theDoc.AddText("Negative CharSpacing");
theDoc.Save(Server.MapPath("stylecspace.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.TextStyle.Size = 96
theDoc.AddText("Zero CharSpacing")
theDoc.Rect.Move(0, -300)
theDoc.TextStyle.CharSpacing = 10
theDoc.AddText("Positive CharSpacing")
theDoc.Rect.Move(0, -300)
theDoc.TextStyle.CharSpacing = -10
theDoc.AddText("Negative CharSpacing")
theDoc.Save(Server.MapPath("stylecspace.pdf"))
theDoc.Clear()
```

**Example**

Zero  
CharSpacing

Positive  
CharSpacing

Negative  
CharSpacing

stylecspace.pdf

# CharUsage Property

---



Type	Default	Read Only	Description
[C#] CharUsageType	Default	No	The usage of characters.
[Visual Basic] CharUsageType			

This property specifies the characters to use.

The CharUsageType enumeration can take any of the following values:

- Default
- SymbolUnicode — enables the range 0xf000–0xf0ff for symbol fonts. Symbol fonts are usually used in ANSI encoding, and their characters are in the range 0–255. Since symbol characters do not represent Latin characters, the range 0xf000–0xf0ff in Unicode is also mapped to them.

---

## Notes

None.

---

## Example



# Direction Property

---



Type	Default Value	Read Only	Description
[C#] DirectionType	DirectionType.Default	No	The default text direction
[Visual Basic] DirectionType			

This property specifies the default primary text direction.

The DirectionType enumeration may take the following values:

- Default - left-to-right reading direction
- LeftToRight - left-to-right reading direction (e.g. English)
- RightToLeft - right-to-left reading direction (e.g. Hebrew).

If a run of English (left-to-right) text is followed by a run of Hebrew (right-to-left) text, LeftToRight gives English text on the left and Hebrew text on the right, whereas RightToLeft gives English text on the right and Hebrew text on the left. In any case, the English text is still left-to-right, and the Hebrew text is still right-to-left.

---

## Notes

To support ligatures (glyph shaping) and right-to-left text, LeftToRight or RightToLeft must be specified at least once somewhere. It can be in this property or in the `dir`

attribute of `StyleRun` (for `Doc.AddHtml`). If only `Default` is specified, there will be no ligature support and right-to-left text may appear incorrect.

None.

---

**Example**

# Font Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The current Font ID.

The font used for drawing text.

This property holds the current Font ID and determines the style of text that is added to the document using methods like [Doc.AddText](#).

---

## Notes

To get a Font ID you need to add your font to the current document using the [Doc.AddFont](#) method. See the [Doc.Font](#) property for further details.

None.

---

## Example

# HPos Property

---



Type	Default Value	Read Only	Description
[C#] double			The current horizontal positioning factor (0 to 1).
[Visual Basic] Double	0?	No?	

This property determines the horizontal offset of blocks of text - used for left alignment, right alignment or centering.

The offset is measured as a proportion of the distance from the left. A value of zero indicates left alignment, a value of one half indicates centered text and a value of one indicates right alignment. Intermediate values can be used for intermediate offsets.

---

## Notes

To vertically align text use the [VPos](#) property. To justify text use the [Justification](#) property.

The following code adds two blocks of text to a document. The first block is left aligned and the second is right aligned. Before adding the text we change the current rectangle and frame it so that you can see how the text is aligned.

## [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.Rect.Magnify(1.0, 0.5);
theDoc.Rect.Inset(40, 40);
theDoc.FrameRect();
theDoc.AddText("Left justified text...");
theDoc.Rect.Move(0, theDoc.Rect.Height +
80);
theDoc.FrameRect();
theDoc.TextStyle.HPos = 1.0;
theDoc.AddText("Right justified text...");
theDoc.Save(Server.MapPath("dochpos.pdf"));
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.Rect.Magnify(1.0, 0.5)
theDoc.Rect.Inset(40, 40)
theDoc.FrameRect()
theDoc.AddText("Left justified text...")
theDoc.Rect.Move(0, theDoc.Rect.Height +
80)
theDoc.FrameRect()
theDoc.TextStyle.HPos = 1.0
theDoc.AddText("Right justified text...")
theDoc.Save(Server.MapPath("dochpos.pdf"))
theDoc.Clear()
```

---

Example

Right  
justified  
text...

Left justified  
text...

dochpos.pdf

# Indent Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	0	No	The first line of paragraph indent.

This property determines the horizontal indent applied to the first line of every paragraph.

---

## Notes

If the indent is positive the start of the first line is shifted to the right by the specified number of points. If the indent is negative it is shifted to the left by the specified number of units.

In this example we add a block of text to a document. We specify a [ParaSpacing](#) value to space out the paragraphs and an `Indent` value to indent the first line of each paragraph.

### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in
partes tres, quarum unam incolunt Belgae, alian
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes lingua,
institutis, legibus inter se differunt.";
```

```
theText = theText + theText;
theText = theText + "\r\n" + theText + "\r\n";
theText = theText + theText + theText +
theText;
theDoc.Rect.Inset(20, 40);
theDoc.TextStyle.Size = 16;
theDoc.TextStyle.ParaSpacing = 16;
theDoc.TextStyle.Indent = 48;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("styleindent.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String = "Gallia est omnis
divisa in partes tres, quarum unam incolunt
Belgae, aliam Aquitani, tertiam qui ipsorum
lingua Celtae, nostra Galli appellantur. Hi
omnes lingua, institutis, legibus inter se
differunt."
theText = theText + theText
theText = theText + vbCrLf + theText + vbCrLf
theText = theText + theText + theText + theText
theDoc.Rect.Inset(20, 40)
theDoc.TextStyle.Size = 16
theDoc.TextStyle.ParaSpacing = 16
theDoc.TextStyle.Indent = 48
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("styleindent.pdf"))
theDoc.Clear()
```

---

Example



# Italic Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to apply a synthetic italic effect.

This property determines whether a synthetic italic effect is applied to text.

It is generally better to specify an italic typeface rather than synthesize an italic effect using the current typeface.

---

## Notes

However, under some circumstances this may not be possible and you may prefer to apply a synthetic italic effect.

In this example we add some italic text to a document.

### [C#]

```
Doc theDoc = new Doc();
string theText;
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
```

```
theDoc.TextStyle.Size = 96;
theDoc.TextStyle.Italic = true;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("styleitalic.pdf"));
theDoc.Clear();
```

### [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()
Dim theText As String
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur."
theDoc.Rect.Inset(20, 40)
theDoc.TextStyle.Size = 96
theDoc.TextStyle.Italic = True
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("styleitalic.pdf"))
theDoc.Clear()
```

*Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,*

styleitalic.pdf

# Justification Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0.0	No	The horizontal justification factor.

Allows you to adjust the horizontal justification.

Every line of text is drawn within a bounding box determined by the [Doc.Rect](#) property. The justification property can be used to space out words so that they fit the bounding box exactly.

Typically you will only need to use values of zero - no justification and one - full justification. However intermediate values can be used to partially justify text.

---

## Notes

As each line is drawn the difference between the width of the line and the width of the bounding box is evaluated to determine the amount of free width. This free width is divided by the number of space characters in the line and then multiplied by the justification factor to produce an inter-word spacing for the line. Lines at the end of paragraphs are not justified.

Text alignment is determined by the [TextStyle.HPos](#) and [TextStyle.VPos](#) properties.

In this example we add two blocks of text to a document. The first is added with no justification and the second is added with a justification factor of one.

### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
theDoc.TextStyle.Size = 48;
theDoc.AddText(theText);
theDoc.Rect.Move(0, -350);
theDoc.TextStyle.Justification = 1.0;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylejustification.doc"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String = "Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur."
theDoc.Rect.Inset(20, 40)
theDoc.TextStyle.Size = 48
theDoc.AddText(theText)
theDoc.Rect.Move(0, -350)
theDoc.TextStyle.Justification = 1.0
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("stylejustification.doc"))
theDoc.Clear()
```

---

Example

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur.

stylejustification.pdf

# Kerning Property

---



Type	Default Value	Read Only	Description
[C#] KerningType	Default	No	The kerning method.
[Visual Basic] KerningType			

The kerning method.

Kerning is similar to [character spacing](#) in that it controls how far apart two characters are. However rather than being a constant, it is a value which is determined by the two characters themselves. So the kerning for "tt" would likely be different than for "te".

The KerningType enumeration may take the following values:

---

## Notes

- None
- Default

The default kerning method is based around the kerning tables in the TrueType font file. Not all fonts contain kerning tables so not all fonts will kern.

The following shows how to insert a table of contents while

disabling kerning.

### [C#]

```
string text =
File.ReadAllText("tableofcontents.txt");
text = text.Replace("\r", "<br>"); // make our
carriage returns into breaks
text = text.Replace(" ", "    "); // make our
indent at start of line into nbsp
using (Doc doc = new Doc()) {
    doc.TextStyle.Size = 36;
    doc.TextStyle.Kerning =
XTextStyle.KerningType.None;
    doc.Rect.Inset(10, 10);
    doc.Page = doc.AddPage();
    doc.AddHtml(text.Replace(" ~", "<leader>."
</leader>"));
    doc.Save("TableOfContentsWithLeaders.pdf");
}
```

### [Visual Basic]

```
Dim text As String =
File.ReadAllText("tableofcontents.txt")
text = text.Replace(vbCr, "<br>")
' make our carriage returns into breaks
text = text.Replace(" ", "    ")
' make our indent at start of line into nbsp
Using doc As New Doc()
    doc.TextStyle.Size = 36
    doc.TextStyle.Kerning =
XTextStyle.KerningType.None
    doc.Rect.Inset(10, 10)
    doc.Page = doc.AddPage()
    doc.AddHtml(text.Replace(" ~", "<leader>."
</leader>"))
    doc.Save("TableOfContentsWithLeaders.pdf")
```

---

Example

End Using  
End Sub

Using the following input text.

```
Chapter 1: Getting Started ~1
Introduction ~2
Next Steps ~3
Chapter 2: What To Do ~4
Some Difficult Bits ~15
Some More Difficult Bits ~20
Chapter 3: In Conclusion ~21
Summary ~22
Endword ~23
```

We get the following output.

Chapter 1: Getting Started .....	1
Introduction.....	2
Next Steps .....	3
Chapter 2: What To Do.....	4
Some Difficult Bits .....	15
Some More Difficult Bits.....	20
Chapter 3: In Conclusion .....	21
Summary .....	22
Endword.....	23



# LeftMargin Property

---



Type	Default	Read Only	Description
[C#] double	0	No	The paragraph indent.
[Visual Basic] Double			

Allows you to apply a left margin to a block of text.

The left margin is a horizontal indent applied to every line of text. You can achieve the same effect by inseting the left of the current [Rect](#) but using this property can be more convenient.

---

## Notes

If the value is positive the block is shifted to the right by the specified number of units. If the value is negative it is shifted to the left by the specified number of units.

In the following example we add three blocks of text to a document. The first block uses the default left margin. The subsequent blocks use different left margin settings to indent the text.

### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in
```

```
partes tres, quarum unam incolunt Belgae, alian
Aquitani...";
theDoc.TextStyle.Size = 48;
theDoc.AddText(theText);
theDoc.Rect.Move(0, -250);
theDoc.TextStyle.LeftMargin = 100;
theDoc.AddText(theText);
theDoc.Rect.Move(0, -250);
theDoc.TextStyle.LeftMargin = 200;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylemargin.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String = "Gallia est omnis
divisa in partes tres, quarum unam incolunt
Belgae, alian Aquitani..."
theDoc.TextStyle.Size = 48
theDoc.AddText(theText)
theDoc.Rect.Move(0, -250)
theDoc.TextStyle.LeftMargin = 100
theDoc.AddText(theText)
theDoc.Rect.Move(0, -250)
theDoc.TextStyle.LeftMargin = 200
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("stylemargin.pdf"))
theDoc.Clear()
```

Example

Gallia est omnis divisa in  
partes tres, quarum unam  
incolunt Belgae, aliam  
Aquitani...

Gallia est omnis divisa in  
partes tres, quarum unam  
incolunt Belgae, aliam  
Aquitani...

Gallia est omnis  
divisa in partes tres,  
quarum unam  
incolunt Belgae,  
aliam Aquitani...

stylelmargin.pdf

# LineSpacing Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0.0	No	The inter-line spacing.

Allows you to adjust the distance between lines of text.

At the start of every new line of text, the text drawing position is shifted vertically by the distance specified in this property. If the value is positive this will space the lines out. If the value is negative it will shift the lines together.

---

## Notes

You can use the font's line spacing:

```
doc.TextStyle.LineSpacing =  
doc.GetInfoDouble(doc.Font, "LineSpacing")  
* doc.TextStyle.Size / 1000
```

In the following example we add three blocks of text to a document. The first block uses the default line spacing. The second block uses a positive value to space out the lines. The last block uses a negative value to shift the lines together.

[C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in
partes tres, quarum unam incolunt Belgae, aliam
Aquitani...";
theDoc.TextStyle.Size = 48;
theDoc.AddText(theText);
theDoc.Rect.Move(0, -250);
theDoc.TextStyle.LineSpacing = 20;
theDoc.AddText(theText);
theDoc.Rect.Move(0, -350);
theDoc.TextStyle.LineSpacing = -20;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylelspace.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String = "Gallia est omnis
divisa in partes tres, quarum unam incolunt
Belgae, aliam Aquitani..."
theDoc.TextStyle.Size = 48
theDoc.AddText(theText)
theDoc.Rect.Move(0, -250)
theDoc.TextStyle.LineSpacing = 20
theDoc.AddText(theText)
theDoc.Rect.Move(0, -350)
theDoc.TextStyle.LineSpacing = -20
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("stylelspace.pdf"))
theDoc.Clear()
```

---

Example

Gallia est omnis divisa in  
partes tres, quarum unam  
incolunt Belgae, aliam  
Aquitani...

Gallia est omnis divisa in  
partes tres, quarum unam  
incolunt Belgae, aliam  
Aquitani...

Gallia est omnis divisa in  
partes tres, quarum unam  
incolunt Belgae, aliam  
Aquitani...

stylelspace.pdf

# Outline Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	0	No	The width of character outlining.

This property determines whether a character outlining style is applied and the weight of the outline.

---

## Notes

If the outline property is zero no outlining is done. If the outline property is greater than zero it indicates the width of lines used to outline the text.

In this example we add some text to a document varying the outline style to show how different values affect the final result.

### [C#]

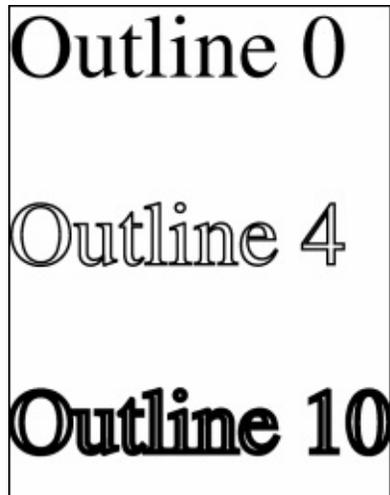
```
Doc theDoc = new Doc();
theDoc.TextStyle.Size = 144;
theDoc.AddText("Outline 0");
theDoc.Rect.Move(0, -300);
theDoc.TextStyle.Outline = 4;
theDoc.AddText("Outline 4");
theDoc.Rect.Move(0, -300);
theDoc.TextStyle.Outline = 10;
```

```
theDoc.AddText("Outline 10");  
theDoc.Save(Server.MapPath("styleoutline.pdf"))  
theDoc.Clear();
```

### [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()  
theDoc.TextStyle.Size = 144  
theDoc.AddText("Outline 0")  
theDoc.Rect.Move(0, -300)  
theDoc.TextStyle.Outline = 4  
theDoc.AddText("Outline 4")  
theDoc.Rect.Move(0, -300)  
theDoc.TextStyle.Outline = 10  
theDoc.AddText("Outline 10")  
theDoc.Save(Server.MapPath("styleoutline.pdf"))  
theDoc.Clear()
```



styleoutline.pdf

# ParaSpacing Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	0.0	No	The inter-paragraph spacing.

Allows you to adjust the distance between paragraphs.

At the start of every new paragraph the text drawing position is shifted vertically by the distance specified in this property. If the value is positive this will space the paragraphs out. If the value is negative it will shift the paragraphs together.

---

## Notes

In this example we add two blocks of text to a document. The first block uses the default paragraph spacing. The second block use a positive value to space out the paragraphs.

### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in
partes tres, quarum unam incolunt Belgae, alian
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur. Hi omnes lingua,
institutis, legibus inter se differunt.";
```

```
theText = theText + "\r\n" + theText + "\r\n" +
theText + "\r\n" + theText;
theDoc.Rect.Inset(20, 40);
theDoc.TextStyle.Size = 16;
theDoc.AddText(theText);
theDoc.Rect.Move(0, -350);
theDoc.TextStyle.ParaSpacing = 20;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylepspace.pdf"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Dim theText As String = "Gallia est omnis
divisa in partes tres, quarum unam incolunt
Belgae, aliam Aquitani, tertiam qui ipsorum
lingua Celtae, nostra Galli appellantur. Hi
omnes lingua, institutis, legibus inter se
differunt."
```

---

#### Example

```
theText = theText + vbCrLf + theText + vbCrLf +
theText + vbCrLf + theText
theDoc.Rect.Inset(20, 40)
theDoc.TextStyle.Size = 16
theDoc.AddText(theText)
theDoc.Rect.Move(0, -350)
theDoc.TextStyle.ParaSpacing = 20
theDoc.AddText(theText)
theDoc.Save(Server.MapPath("stylepspace.pdf"))
theDoc.Clear()
```

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur. Hi omnes lingua, instituta, legibus inter se differunt.

styleSPACE.pdf

# PreserveSpace Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to preserve white space characters for plain text.

This property determines whether white space characters are preserved when [Doc.AddText](#) is called. It does not affect [Doc.AddHtml](#).

---

## Notes

---

## Example

None.

# Size Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	10.0	No	The current text size.

This property determines the size of text that is added to the document using methods like [AddText](#).

The Size property is equivalent to the the [Doc.FontSize](#) property but unlike the FontSize property it allows fractional point sizes to be specified.

---

## Notes

The font size is measured in units.

The following example adds two blocks of styled text to a document. The first block is in 96.5 point type and the second is in 192.5 point type.

### [C#]

```
Doc theDoc = new Doc();
theDoc.TextStyle.Size = 96.5;
theDoc.AddText("Small ");
theDoc.TextStyle.Size = 192.5;
theDoc.AddText("Big");
```

```
theDoc.Save(Server.MapPath("stylesize.pdf"));  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
theDoc.TextStyle.Size = 96.5  
theDoc.AddText("Small ")  
theDoc.TextStyle.Size = 192.5  
theDoc.AddText("Big")  
theDoc.Save(Server.MapPath("stylesize.pdf"))  
theDoc.Clear()
```

Example



stylesize.pdf

# Strike Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to apply a strikethrough effect.

This property determines whether a strikethrough is applied to text.

---

## Notes

In this example we add some strikethrough styled text to a document.

### [C#]

```
Doc theDoc = new Doc();
string theText;
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
theDoc.TextStyle.Size = 96;
theDoc.TextStyle.Strike = true;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylestrike.pdf"));
```

```
theDoc.Clear();
```

### [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()  
Dim theText As String  
theText = "Gallia est omnis divisa in partes  
tres, quarum unam incolunt Belgae, aliam  
Aquitani, tertiam qui ipsorum lingua Celtae,  
nostra Galli appellantur."  
theDoc.Rect.Inset(20, 40)  
theDoc.TextStyle.Size = 96  
theDoc.TextStyle.Strike = True  
theDoc.AddText(theText)  
theDoc.Save(Server.MapPath("stylestrike.pdf"))  
theDoc.Clear()
```

~~Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,~~

stylestrike.pdf

# Strike2 Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to apply a double strikethrough effect.

This property determines whether a double strikethrough is applied to text.

---

## Notes

In this example we add some double strikethrough styled text to document.

### [C#]

```
Doc theDoc = new Doc();
string theText;
theText = "Gallia est omnis divisa in partes
tres, quarum unam incolunt Belgae, aliam
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
theDoc.TextStyle.Size = 96;
theDoc.TextStyle.Strike2 = true;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("stylestrike2.pdf"))
```

```
theDoc.Clear();
```

### [Visual Basic]

Example

```
Dim theDoc As Doc = New Doc()  
Dim theText As String  
theText = "Gallia est omnis divisa in partes  
tres, quarum unam incolunt Belgae, aliam  
Aquitani, tertiam qui ipsorum lingua Celtae,  
nostra Galli appellantur."  
theDoc.Rect.Inset(20, 40)  
theDoc.TextStyle.Size = 96  
theDoc.TextStyle.Strike2 = True  
theDoc.AddText(theText)  
theDoc.Save(Server.MapPath("stylestrike2.pdf"))  
theDoc.Clear()
```



~~Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,~~

stylestrike2.pdf

# String Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	Variable	No	The text style as a string.

A string representation of the text style.

---

## Notes

This covers all the properties of this class and can be used for a save and restore stack.

The following code.

### [C#]

```
XTextStyle ts = new XTextStyle();
ts.String = "24.5 10 0 0 0 0 0";
Response.Write("Size = " +
ts.Size.ToString() + "<br>");
Response.Write("Indent = " +
ts.Indent.ToString());
```

### [Visual Basic]

```
Dim ts As XTextStyle = New
XTextStyle()
```

---

## Example

```
ts.String = "24.5 10 0 0 0 0 0"
Response.Write("Size = " +
ts.Size.ToString() + "<br>")
Response.Write("Indent = " +
ts.Indent.ToString())
```

Produces the following output.

```
Size = 24.5
Indent = 10
```

# Underline Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to underline text.

This property determines whether underlining is applied to text.

---

## Notes

In this example we add some underlined text to a document.

### [C#]

```
Doc theDoc = new Doc();
string theText = "Gallia est omnis divisa in
partes tres, quarum unam incolunt Belgae, alian
Aquitani, tertiam qui ipsorum lingua Celtae,
nostra Galli appellantur.";
theDoc.Rect.Inset(20, 40);
theDoc.TextStyle.Size = 96;
theDoc.TextStyle.Underline = true;
theDoc.AddText(theText);
theDoc.Save(Server.MapPath("styleunderline.pdf");
theDoc.Clear();
```

### [Visual Basic]

---

---

Example

```
Dim theDoc As Doc = New Doc()  
Dim theText As String = "Gallia est omnis divis  
in partes tres, quarum unam incolunt Belgae, al  
Aquitani, tertiam qui ipsorum lingua Celtae,  
nostra Galli appellantur."  
theDoc.Rect.Inset(20, 40)  
theDoc.TextStyle.Size = 96  
theDoc.TextStyle.Underline = True  
theDoc.AddText(theText)  
theDoc.Save(Server.MapPath("styleunderline.pdf")  
theDoc.Clear()
```

Gallia est  
omnis divisa  
in partes tres,  
quarum unam  
incolunt  
Belgae, aliam  
Aquitani,

styleunderline.pdf

# VPos Property

---



Type	Default Value	Read Only	Description
[C#] double			The current vertical positioning factor (0 to 1).
[Visual Basic] Double	0?	No?	

This property determines the vertical offset of blocks of text - used for bottom alignment, top alignment or middle alignment.

The offset is measured as a proportion of the distance from the top. A value of zero indicates top alignment, a value of one half indicates center aligned text and a value of one indicates bottom aligned text. Intermediate values can be used for intermediate offsets.

---

## Notes

To horizontally align text use the [HPos](#) property. To justify text use the [Justification](#) property.

The following code adds two blocks of text to a document. The first block is bottom aligned and the second is top aligned. Before adding the text we change the current rectangle and frame it so that you can see how the text is aligned.

## [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 96;
theDoc.Rect.Magnify(1.0, 0.5);
theDoc.Rect.Inset(40, 40);
theDoc.FrameRect();
theDoc.AddText("Top aligned text...");
theDoc.Rect.Move(0, theDoc.Rect.Height +
80);
    theDoc.FrameRect();
theDoc.TextStyle.TextStyle.VPos = 1.0;
theDoc.AddText("Bottom aligned text...");
theDoc.Save(Server.MapPath("docvpos.pdf"));
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.Rect.Magnify(1.0, 0.5)
theDoc.Rect.Inset(40, 40)
theDoc.FrameRect()
theDoc.AddText("Top aligned text...")
theDoc.Rect.Move(0, theDoc.Rect.Height +
80)
theDoc.FrameRect()
theDoc.TextStyle.TextStyle.VPos = 1.0
theDoc.AddText("Bottom aligned text...")
theDoc.Save(Server.MapPath("docvpos.pdf"))
theDoc.Clear()
```

---

Example

Bottom  
aligned text...

Top aligned  
text...

docvpos.pdf

# WordSpacing Property

---



Type	Default	Read Only	Description
[C#]double			
[Visual Basic]Double	0.0	No	The inter-word spacing.

This property controls the spacing between each word.

This property works in a similar way to the [CharSpacing](#) property but the value is added to only space characters in the string. This has the effect of spacing out words or pushing them together depending on the sign of the value, and the property bears the same relation to the font's writing mode as [CharSpacing](#).

---

## Notes

Because this property is an absolute value the visual effect will be greater if your text is smaller.

In this example we add three blocks of text to a document. The first block uses the default spacing. The second block uses a positive value to stretch out the text. The last block uses a negative value to condense the text.

## [C#]

```
Doc theDoc = new Doc();
```

```
string theText = "This is an example of word  
spacing.";  
theDoc.TextStyle.Size = 72;  
theDoc.AddText(theText);  
theDoc.Rect.Move(0, -300);  
theDoc.TextStyle.WordSpacing = 20;  
theDoc.AddText(theText);  
theDoc.Rect.Move(0, -300);  
theDoc.TextStyle.WordSpacing = -20;  
theDoc.AddText(theText);  
theDoc.Save(Server.MapPath("stylewspace.pdf"));  
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()  
Dim theText As String = "This is an example of  
word spacing."  
theDoc.TextStyle.Size = 72  
theDoc.AddText(theText)  
theDoc.Rect.Move(0, -300)  
theDoc.TextStyle.WordSpacing = 20  
theDoc.AddText(theText)  
theDoc.Rect.Move(0, -300)  
theDoc.TextStyle.WordSpacing = -20  
theDoc.AddText(theText)  
theDoc.Save(Server.MapPath("stylewspace.pdf"))  
theDoc.Clear()
```

---

Example

This is an example  
of word spacing.

This is an example  
of word spacing.

Thisisanexampleof  
wordspacing.

stylewspace.pdf



# XTransform Constructor

---

XTransform Constructor.

## [C#]

```
XTransform()  
XTransform(string text)  
XTransform(double[] entries)  
XTransform(Matrix matrix)  
XTransform(double m11, double  
m12, double m21, double m22,  
double tx, double ty)  
XTransform(XRect src, XRect dst)
```

## Syntax

## [Visual Basic]

```
Sub New  
Sub New(text As String)  
Sub New(entries As Double())  
Sub New(entries As Matrix)  
Sub New(m11 As Double, m12 As  
Double, m21 As Double, m22 As  
Double, tx As Double, ty As  
Double)  
Sub New(src As XRect, dst As  
XRect)
```

Name	Description

---

**Params**

text	A string defining the initial rectangle in the format returned by the <a href="#">String</a> property.
matrix	A System.Drawing.Drawing2D Matrix object specifying the values for the transform.
entries	An array of doubles specifying the values for the matrix. These are in the order m11, m12, m21, m22, tx and ty.
m11	Matrix entry 1,1.
m12	Matrix entry 1,2.
m21	Matrix entry 2,1.
m22	Matrix entry 2,2.
tx	The x translation to be applied.
ty	The y translation to be applied.
src	The source rectangle.
dst	The source rectangle.

These methods construct an XTransform object.

The default constructor creates an identity transform.

---

**Notes**

The constructor which takes two rectangles creates a transform which maps the source rectangle to the destination rectangle using positive scale values.

---

None.

## Example



# Invert Function

---

Invert the transform.

## [C#]

```
void Invert()
```

---

### Syntax

## [Visual Basic]

```
Sub Invert()
```

Name	Description
None	

---

### Params

When you invert a transform a rotation clockwise becomes identical rotation anti-clockwise. A translation to the left becomes a translation to the right. A zoom in becomes a zoom out.

Note that not every transform can be inverted. If you specify magnification of zero you have shrunk your world space to a point. In this case it is not possible to invert the transform to the original back again. However this kind of transform is uncommon in the real world and normally only occurs as a result of programming errors.

---

### Notes

If you apply the invert method to a non-invertable transform the transform will remain unmodified.

Here we add some text rotated at 45 degrees anti-clockwise in the middle of the document. We then invert the transform and add more text. Because the transform has been inverted the text appears rotated 45 degrees clockwise.

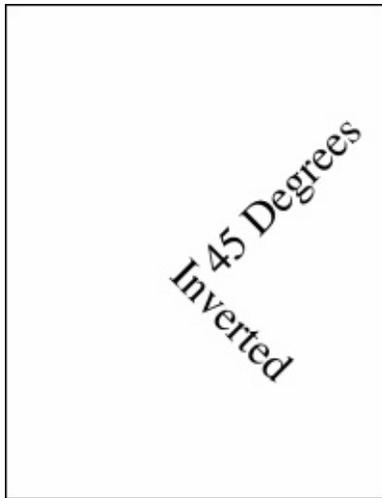
### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 72;
theDoc.Rect.String = "0 0 999 999";
theDoc.Pos.String = "302 396";
theDoc.Transform.Rotate(45, 302, 396);
theDoc.AddText("45 Degrees");
theDoc.Pos.String = "302 396";
theDoc.Transform.Invert();
theDoc.AddText("Inverted");
theDoc.Save(Server.MapPath("transforminvert"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 72
theDoc.Rect.String = "0 0 999 999"
theDoc.Pos.String = "302 396"
theDoc.Transform.Rotate(45, 302, 396)
theDoc.AddText("45 Degrees")
theDoc.Pos.String = "302 396"
theDoc.Transform.Invert()
theDoc.AddText("Inverted")
theDoc.Save(Server.MapPath("transforminvert"))
theDoc.Clear()
```

Example



transforminvert.pdf

# Equals Function

---



Determines if two transforms are effectively the same.

## [C#]

```
bool Equals(XTransform other,  
double epsilon)  
bool Equals(XTransform other)  
override bool Equals(object  
other)
```

## [Visual Basic]

---

### Syntax

```
Function Equals(other As  
XTransform, epsilon As Double) As  
Boolean  
Function Equals(other As  
XTransform) As Boolean  
Overrides Function Equals(other  
As Object) As Boolean
```

---

### Params

Name	Description
other	The transform to be compared against this one
epsilon	The largest difference in values which will still be defined as equal
return	Whether the two transforms are the same.

Determines if two transforms are effectively the same.

Transforms contain a set of elements which are represented as floating point numbers. Floating point numbers are subject to rounding errors so the epsilon value is used to determine the resolution of the comparison. Only if two elements differ by more than the value of epsilon will they be defined as not equal.

---

**Notes**

Internally Acrobat uses double precision floating point numbers for a very high level of accuracy. However within a PDF itself these numbers are typically only represented to around five decimal points. You may wish to represent similar levels of accuracy in your epsilon values. The default epsilon is zero so by default, transforms have to match perfectly for them to be considered equal.

None.

---

**Example**



# Magnify Function

---

Scale about a locked anchor point.

## [C#]

```
void Magnify(double scaleX, double scaleY,  
double anchorX, double anchorY)
```

---

### Syntax

## [Visual Basic]

```
Sub Magnify(scaleX As Double, scaleY As Dou  
anchorX As Double, anchorY As Double)
```

---

### Params

Name	Description
scaleX	The amount of horizontal scaling to apply.
scaleY	The amount of vertical scaling to apply.
anchorX	The horizontal coordinate about which the stretch should be applied.
anchorY	The vertical coordinate about which the stretch should be applied.

This method stretches the world space about a locked anchor point. Different degrees of horizontal and vertical stretch can be used.

---

### Notes

Another way of looking at this kind of transform is as a zoom

anchor point is the location you're zooming in on and the scale factors indicate the level of zoom.

Here we add two chunks of text. The default text is added in black and the magnified text is drawn in red. We specify the middle of the document as the anchor point which means that all scaling is done relative to the middle of the document. Our horizontal scale factor is 1.5 so the text has been stretched horizontally somewhat.

### [C#]

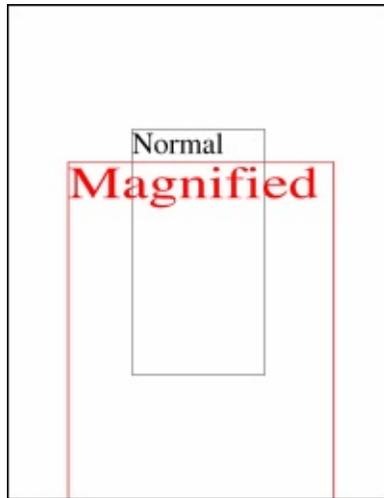
```
Doc theDoc = new Doc();
theDoc.Rect.Inset(200, 200);
theDoc.FontSize = 48;
theDoc.AddText("Normal");
theDoc.FrameRect();
theDoc.Rect.Move(0, -100);
theDoc.Color.String = "255 0 0";
theDoc.Transform.Magnify(2, 1.5, 302, 396);
theDoc.AddText("Magnified");
theDoc.FrameRect();
theDoc.Save(Server.MapPath("transformmagnif"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(200, 200)
theDoc.FontSize = 48
theDoc.AddText("Normal")
theDoc.FrameRect()
theDoc.Rect.Move(0, -100)
theDoc.Color.String = "255 0 0"
theDoc.Transform.Magnify(2, 1.5, 302, 396)
theDoc.AddText("Magnified")
theDoc.FrameRect()
```

**Example**

```
theDoc.Save(Server.MapPath("transformmagnif  
theDoc.Clear()
```



transformmagnify.pdf



# PreMultiply Function

---

Pre-multiplies this transformation matrix by the supplied transform.

## [C#]

```
void PreMultiply(XTransform  
transform)
```

---

## Syntax

## [Visual Basic]

```
Sub PreMultiply(transform as  
XTransform)
```

---

## Params

Name	Description
transform	The transform to combine with this one.

Pre-multiplies this transformation matrix by the supplied transform.

---

## Notes

The final result is contained in this transform.

See also the [Mutiply](#) function.

---

Example

None.



# PostMultiply Function

---

Post-multiplies this transformation matrix by the supplied transform.

## [C#]

```
void PostMultiply(XTransform  
transform)
```

---

## Syntax

## [Visual Basic]

```
Sub PostMultiply(transform as  
XTransform)
```

---

## Params

Name	Description
transform	The transform to combine with this one.

Post-multiplies this transformation matrix by the supplied transform.

---

## Notes

The final result is contained in this transform.

See also the [PreMutiply](#) function.

---

Example

None.



# Reset Function

---

Reset to the identity.

## [C#]

```
void Reset()
```

---

## Syntax

## [Visual Basic]

```
Sub Reset()
```

---

## Params

Name	Description
None	

---

## Notes

This method resets the transform to its original state. This state is known as the identity and indicates that no transformation will be applied.

Here we add some text rotated at 60 degrees around the m the document. We then reset the transform and draw some This text is drawn with no rotation because the transform has reset.

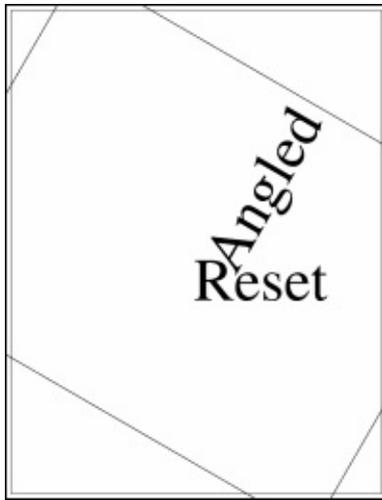
## [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(10, 10);
theDoc.FontSize = 96;
theDoc.Transform.Rotate(60, 302, 396);
theDoc.Pos.String = "302 396";
theDoc.AddText("Angled");
theDoc.FrameRect();
theDoc.Transform.Reset();
theDoc.Pos.String = "302 396";
theDoc.AddText("Reset");
theDoc.FrameRect();
theDoc.Save(Server.MapPath("transformreset.
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(10, 10)
theDoc.FontSize = 96
theDoc.Transform.Rotate(60, 302, 396)
theDoc.Pos.String = "302 396"
theDoc.AddText("Angled")
theDoc.FrameRect()
theDoc.Transform.Reset()
theDoc.Pos.String = "302 396"
theDoc.AddText("Reset")
theDoc.FrameRect()
theDoc.Save(Server.MapPath("transformreset.
theDoc.Clear()
```

Example



transformreset.pdf



## Rotate Function

---

Rotate about a locked anchor point (angle in degrees).

### [C#]

```
void Rotate(double angle, double anchorX,  
double anchorY)
```

---

### Syntax

### [Visual Basic]

```
Sub Rotate(angle As Double, anchorX As Double,  
anchorY As Double)
```

---

### Params

Name	Description
angle	The angle to rotate in degrees.
anchorX	The horizontal coordinate about which the rotation should be applied.
anchorY	The vertical coordinate about which the rotation should be applied.

---

### Notes

This method rotates the world space about a locked anchor point. The angle is specified in degrees anti-clockwise.

Here we add a number of chunks of text rotated at different

about the middle of the document.

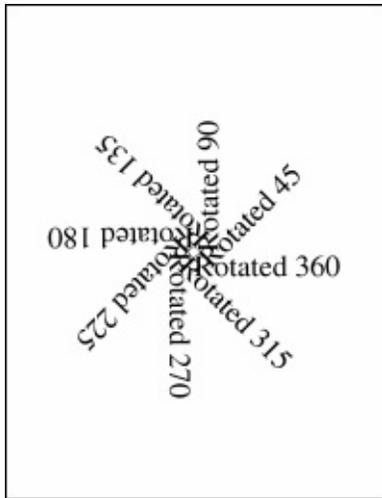
### [C#]

```
Doc theDoc = new Doc();
theDoc.FontSize = 48;
theDoc.TextStyle.Indent = 48;
for (int i = 1; i <= 8; i++) {
    int theAngle = i * 45;
    theDoc.Pos.String = "302 396";
    theDoc.Transform.Reset();
    theDoc.Transform.Rotate(theAngle, 302, 396);
    theDoc.AddText("Rotated " + theAngle.ToString());
}
theDoc.Save(Server.MapPath("transformrotate"));
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.FontSize = 48
theDoc.TextStyle.Indent = 48
For i As Integer = 1 To 8
    Dim theAngle As Integer = i * 45
    theDoc.Pos.String = "302 396"
    theDoc.Transform.Reset()
    theDoc.Transform.Rotate(theAngle, 302, 396)
    theDoc.AddText("Rotated " + theAngle.ToString())
Next
theDoc.Save(Server.MapPath("transformrotate"));
theDoc.Clear();
```

Example



transformrotate.pdf



## Skew Function

---

Skew horizontally and vertically about a locked anchor point

### [C#]

```
void Skew(double skewX, double skewY, double anchorX, double anchorY)
```

---

### Syntax

### [Visual Basic]

```
Sub Skew(skewX As Double, skewY As Double, anchorX As Double, anchorY As Double)
```

---

### Params

Name	Description
skewX	The amount of horizontal skewing to apply.
skewY	The amount of vertical skewing to apply.
anchorX	The horizontal coordinate about which the stretch should be applied.
anchorY	The vertical coordinate about which the stretch should be applied.

This method skews the world space about a locked anchor point. Different degrees of horizontal and vertical stretch can be used.

---

A skew or shear is a mathematical operation which shifts

**Notes** points by an amount proportional to the distance from the anchor point. This shift is scaled by the horizontal and vertical skew factors.

Here we draw two rectangles into our document. The black rectangle is drawn before the skew operation and the red one drawn after it.

### [C#]

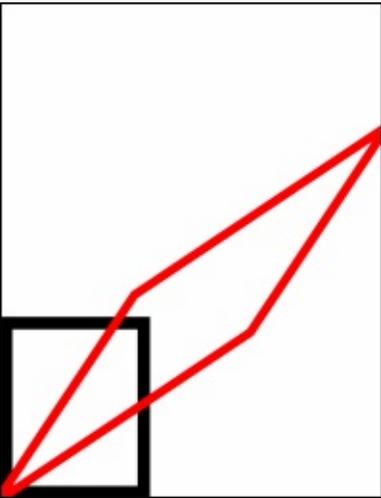
```
Doc theDoc = new Doc();
theDoc.Rect.Width = 200;
theDoc.Rect.Height = 250;
theDoc.Rect.Position(20, 20);
theDoc.Width = 20;
theDoc.FrameRect();
theDoc.Transform.Skew(1.5, 1.5, 20, 20);
theDoc.Color.String = "255 0 0"; // red
theDoc.FrameRect();
theDoc.Save(Server.MapPath("transformskew.p
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Width = 200
theDoc.Rect.Height = 250
theDoc.Rect.Position(20, 20)
theDoc.Width = 20
theDoc.FrameRect()
theDoc.Transform.Skew(1.5, 1.5, 20, 20)
theDoc.Color.String = "255 0 0" ' red
theDoc.FrameRect()
theDoc.Save(Server.MapPath("transformskew.p
theDoc.Clear()
```

---

**Example**



transformskew.pdf



# SetTransform Function

---

Set the transform.

## [C#]

```
void SetTransform(double m11,  
double m12, double m21, double  
m22, double tx, double ty)  
void SetTransform(XTransform  
transform)
```

## [Visual Basic]

### Syntax

```
Sub SetTransform(m11 As Double,  
m12 As Double, m21 As Double, m22  
As Double, tx As Double, ty As  
Double)  
Sub SetTransform(transform As  
XTransform)
```

### Params

Name	Description
m11	The new horizontal scale.
m12	The new vertical skew.
m21	The new horizontal skew.
m22	The new vertical scale.
tx	The new horizontal translation.
ty	The new vertical translation.
transform	The source transform.

This method sets the transform.

---

**Notes**

None.

---

**Example**



# ToString Function

---

Returns a string representation of the object.

## [C#]

```
override string ToString()
```

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Syntax

Name	Description
return	The string representation of the object.

---

### Params

This method returns the string value of the object. This is equivalent to reading the [String](#) property of the object.

---

### Notes

None.

---

### Example





## TransformPoint Function

---

Applies this transform to a specified point.

### [C#]

```
void TransformPoint(Point[]  
point)  
void TransformPoint(PointF[]  
point)  
void TransformPoint(XPoint point)  
void TransformPoint(ref double x,  
ref double y)
```

### [Visual Basic]

#### Syntax

```
Sub TransformPoint(point As  
Point())  
Sub TransformPoint(point As  
PointF())  
Sub TransformPoint(point As  
XPoint())  
Sub TransformPoint(ByRef point As  
Double, ByRef y As Double)
```

Name	Description
point	The point to be transformed.
x	The x coordinate of a point to be transformed.
	The y coordinate of a point to be

#### Params

y transformed.

---

**Notes**

Applies this transform to a specified point.

None.

---

**Example**

# TransformPoints Function

---



Applies this transform to a specified array of points.

## [C#]

```
override Point[]  
TransformPoints(Point[] points)  
override PointF[]  
TransformPoints(PointF[] points)  
override XPoint[]  
TransformPoints(XPointF points)
```

## [Visual Basic]

### Syntax

```
Overrides Sub  
TransformPoints(points As  
Point()) As Point[]  
Overrides Sub  
TransformPoints(points As  
PointF()) As PointF[]  
Overrides Sub  
TransformPoints(points As  
XPoint()) As XPoint[]
```

### Params

Name	Description
points	The array of points to be transformed.
return	The array that was passed in.

Applies this transform to a specified array of points.

---

**Notes**

The behavior of this method is the same as that of the `System.Drawing.Drawing2D.Matrix.TransformPoints` function.

None.

---

**Example**



# Translate Function

---

Translate horizontally and vertically.

## [C#]

```
void Translate(double x, double y)
```

---

### Syntax

## [Visual Basic]

```
Sub Translate(x As Double, y As Double)
```

---

### Params

Name	Description
x	The distance to translate to the right.
y	The distance to translate upwards.

---

### Notes

This method shifts the world space a specified distance up the right. Objects on the PDF will appear to translate upward the right.

Here we draw two rectangles into our document. The black drawn before the translation operation and the red one is drawn after.

## [C#]

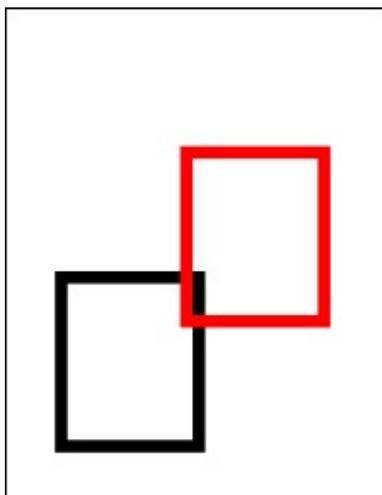
```
Doc theDoc = new Doc();
```

```
theDoc.Rect.Width = 200;
theDoc.Rect.Height = 250;
theDoc.Rect.Position(100, 100);
theDoc.Width = 20;
theDoc.FrameRect();
theDoc.Transform.Translate(200, 200);
theDoc.Color.String = "255 0 0"; // red
theDoc.FrameRect();
theDoc.Save(Server.MapPath("transformtransl
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Width = 200
theDoc.Rect.Height = 250
theDoc.Rect.Position(100, 100)
theDoc.Width = 20
theDoc.FrameRect()
theDoc.Transform.Translate(200, 200)
theDoc.Color.String = "255 0 0" ' red
theDoc.FrameRect()
theDoc.Save(Server.MapPath("transformtransl
theDoc.Clear()
```

Example



transformtranslate.pdf



# GetHashCode Function

---



A hash code for the XTransform.

## [C#]

```
override int GetHashCode()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function GetHashCode() As Integer
```

---

### Params

Name	Description
return	The returned hash code.

---

### Notes

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

---

### Example

None.

# String Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	"1 0 0 1 0 0"	No	The transform as a string.

Allows you access to the transform as a string.

The format of the string must be "m11 m12 m21 m22 mX mY".

---

## Notes

To transform a point (x, y) to another point (x', y'), the following formula is used:

$$\begin{aligned}x' &= (x * m11) + (y * m21) + mX \\y' &= (x * m12) + (y * m22) + mY\end{aligned}$$

None.

---

## Example

# AngleUnit Property

---



Type	Default	Read Only	Description
[C#]AngleUnitType			The angle unit, degree or radians.
[Visual Basic]AngleUnitType	Degrees	No	

Specify the angle unit as used by the [Rotate](#) method.

The AngleUnitType enumeration may take the following values:

---

## Notes

- Degrees
- Radians

## [C#]

```
Doc doc = new Doc();
doc.FontSize=96;
doc.TextStyle.HPos=0.5;
doc.TextStyle.VPos=0.5;
doc.Transform.Rotate(90, doc.Rect.Width/2,
doc.Rect.Height/2);
doc.TextStyle.Underline = true;
doc.AddText("Hello World rotated by 90 degrees'
doc.Page = doc.AddPage();
doc.Transform.AngleUnit =
```

```
WebSupergoo.ABCpdf10.XTransform.AngleUnitType.F
doc.Transform.Rotate(-1 * Math.PI/2, doc.Rect.W
2, doc.Rect.Height / 2);
doc.AddText("Hello World rotated back by PI/2
radians");
doc.Save(Server.MapPath("transformrotate.pdf"))
```

## [Visual Basic]

```
Dim doc as New Doc()
doc.FontSize=96
doc.TextStyle.HPos=0.5
doc.TextStyle.VPos=0.5
doc.Transform.Rotate(90, doc.Rect.Width/2,
doc.Rect.Height/2)
doc.TextStyle.Underline = True
doc.AddText("Hello World rotated by 90 degrees'
doc.Page = doc.AddPage()
doc.Transform.AngleUnit =
WebSupergoo.ABCpdf10.XTransform.AngleUnitType.F
doc.Transform.Rotate(-1 * Math.PI/2, doc.Rect.W
2, doc.Rect.Height / 2)
doc.AddText("Hello World rotated back by PI/2
radians")
doc.Save(server.MapPath("transformrotate.pdf"))
```

Example

Hello World  
rotated by 90  
degrees

transforrotate.pdf, page 1

Hello World  
rotated back by  
PI/2 radians

transforrotate.pdf, page 2

# Elements Property

---



Type	Default	Read Only	Description
[C#] double[]			
[Visual Basic] Double()	n/a	No	The transform as an array of floating-point values.

The transform as an array of floating-point values.

Windows coordinates are measured in distances from the top left of the drawing surface while PDF coordinates are measured from the bottom left.

---

## Notes

Remember that transforms operate on the underlying PDF coordinates rather than on any Windows coordinates. So by specifying a rotation around the origin you are specifying a rotation anchored at the bottom left of the document page.

None.

---

## Example

# Matrix Property

---



Type	Default	Read Only	Description
[C#] Matrix			
[Visual Basic] Matrix	n/a	No	The transform as a System.Drawing.Drawing2D Matrix.

The transform as a System.Drawing.Drawing2D Matrix.

This matrix type holds element values as single precision floating point values so it is less accurate than the underlying XTransform.

Windows coordinates are measured in distances from the top left of the drawing surface while PDF coordinates are measured from the bottom left.

---

## Notes

Remember that transforms operate on the underlying PDF coordinates rather than on any Windows coordinates. So by specifying a rotation around the origin you are specifying a rotation anchored at the bottom left of the document page.

None.

---

## Example



# MediaMatrix Property

---



Type	Default	Read Only	Description
[C#] MediaMatrix	n/a	No	The transform as a System.Windows.Media Matrix.
[Visual Basic] MediaMatrix			

The transform as a System.Windows.Media Matrix.

This matrix type holds elements as double precision floating point values.

Windows coordinates are measured in distances from the top left of the drawing surface while PDF coordinates are measured from the bottom left.

---

## Notes

Remember that transforms operate on the underlying PDF coordinates rather than on any Windows coordinates. So by specifying a rotation around the origin you are specifying a rotation anchored at the bottom left of the document page.

None.

---

## Example



# OffsetX Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	Yes	The x translation.

Allows you access to the x translation.

---

## Notes

None.

---

## Example

# OffsetY Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0	Yes	The y translation.

Allows you access to the y translation.

---

## Notes

None.

---

## Example

# FromString Function

---



Construct an appropriate type of `IndirectObject` given a string value.

## [C#]

```
static IndirectObject  
FromString(string value)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromString(value  
As String) As IndirectObject
```

---

## Params

Name	Description
value	The string representing the value of the object.
return	The resulting <code>IndirectObject</code> .

The text you pass this function must be in native PDF format. This means that unusual characters in text strings must be appropriately escaped.

---

## Notes

For full details of the way that PDF objects are represented you should see the Adobe PDF Specification.

---

**Example**

None.



# IndirectObject Constructor

---

IndirectObject Constructor.

## [C#]

```
IndirectObject()
```

---

### Syntax

## [Visual Basic]

```
Sub New()
```

---

### Params

Name	Description
none	

Creates an indirect object containing a [NullAtom](#).

---

### Notes

Typically you will need to replace the NullAtom before you can do anything useful with the object.

---

### Example

None.





# Dispose Function

---

Dispose of the object.

## [C#]

```
void Dispose()  
protected void Dispose(bool  
disposing)
```

---

### Syntax

## [Visual Basic]

```
Sub Dispose()  
Protected Sub Dispose(disposing  
As Boolean)
```

---

### Params

Name	Description
none	

You can call this function to explicitly dispose of an object and reduce the garbage collection overhead.

This method follows the standard design pattern for objects implementing the IDisposable interface. The protected Dispose method can be overridden for sub-classes wishing to dispose of additional objects.

---

### Notes

Do not attempt to use an object after calling  
Dispose.

---

**Example**

None.



# Clone Function

---

Create a deep copy of the current `IndirectObject`.

## [C#]

```
IndirectObject Clone()
```

---

### Syntax

## [Visual Basic]

```
Function Clone() As  
IndirectObject
```

---

### Params

Name	Description
return	The newly created copy.

This function creates a new object that is a copy of this instance.

The copy is a deep copy and all contained objects are copied as part of the clone process. The copy is not associated with any [ObjectSoup](#).

---

### Notes

Note that many methods require that an object be part of a soup. For this reason it is quite common to call `doc.ObjectSoup.Add` with the

newly cloned object before calling methods on it. If at a later date the object needs to be deleted this can be done using [ObjectSoup.Remove](#).

---

**Example**

None.

# Equals Function

---



Test whether the two IndirectObjects are the same.

## [C#]

```
bool Equals(IndirectObject other)
override bool Equals(object
other)
bool Equals(IndirectObject other,
ComparisonType type)
```

## [Visual Basic]

### Syntax

```
Function Equals(other As
IndirectObject) As Boolean
Overrides Function Equals(other
As Object) As Boolean
Function Equals(other As
IndirectObject, type As
ComparisonType) As Boolean
```

### Params

Name	Description
other	The object to test against.
return	Whether the objects are equal.

This method can be used to determine whether

the specified object is equal to the current object.

Objects are considered equal if they refer to the same underlying object within the PDF document. So by default this method determines object equality rather than value equality.

To perform other types of equality test you can use the overload that accepts a `ComparisonType`. The comparison type specifies the type of data to compare.

---

**Notes**

The `ComparisonType` enumeration is a flags type enumeration so the different values can be combined together using bitwise operations. It may take the following values:

- None (nothing)
- Object (the underlying PDF object)
- ID (the `ID` of the `IndirectObject`)
- Atom (the value of the `Atom`)
- Data (the data associated with the stream - if this is a stream)

Only if all the comparisons are true are the objects said to be equal.

---

**Example**

None.



# GetHashCode Function

---

A hash code for the IndirectObject.

## [C#]

```
override int GetHashCode()  
int GetHashCode(ComparisonType  
type)
```

---

## Syntax

## [Visual Basic]

```
Overrides Function GetHashCode()  
As Integer  
Function GetHashCode(type As  
ComparisonType) As Integer
```

---

## Params

Name	Description
type	The elements to use in the hash code.
return	The returned hash code.

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

The default hash code is derived from the underlying object within the PDF document. In some situations you may wish to derive a hash code from just some aspects of the

IndirectObject. You can do this using the overload which takes a ComparisonType argument.

The ComparisonType enumeration is a flags type enumeration so the different values can be combined together using bitwise operations. It may take the following values:

- None (nothing)
- Object (the underlying PDF object)
- ID (the **ID** of the IndirectObject)
- Atom (the value of the **Atom**)
- Data (the data associated with the stream - if this is a stream)

---

Notes

The hash code can be made up of a variety of parts of the indirect object combined together. The ID and the Atom hash codes are derived from the value of the ID and Atom respectively. The Data hash code is made from a sample of the compressed data contained in the stream. So two StreamObjects, compressed differently, will return different hash codes even if the uncompressed data is identical. The data sample is kept small - in the order of a few hundred bytes - to ensure a fast response even for very large streams.

None.

---

Example



## Resolve Function

---

Resolves any indirect references and returns the Atom.

### [C#]

```
Atom Resolve(Atom atom)
```

---

### Syntax

### [Visual Basic]

```
Function Resolve(atom As Atom) As Atom
```

---

### Params

Name	Description
atom	The Atom to resolve.
return	The final Atom or null if no Atom could be found.

[Atoms](#) come in two basic types. Data atoms like [NumAtoms](#) and [NameAtoms](#) contain actual data. [RefAtoms](#) contain a reference to an [IndirectObject](#) which contains another Atom.

Quite often you will want to resolve any [RefAtoms](#) and just obtain the Atom within the final [IndirectObject](#) - you want the final item of data rather than a reference to a piece of data.

---

**Notes**

This function takes an Atom. If it is a RefAtom it finds the Atom to which it points. It keeps doing this until it finds and returns the final data Atom in the chain.

This method may return null if null is passed in, if a RefAtom cannot be resolved or if a circular dependency is detected.

The reason this function is a member of the IndirectObject is because resolving RefAtoms requires access to the [ObjectSoup](#). The ObjectSoup is taken from the IndirectObject.

---

**Example**

None.



# ResolveRef Function

---

Resolves any indirect references and returns the final RefAtom.

## [C#]

```
RefAtom ResolveRef(Atom atom)
```

---

## Syntax

## [Visual Basic]

```
Function ResolveRef(atom As Atom)  
As RefAtom
```

---

## Params

Name	Description
atom	The Atom to resolve.
return	The final RefAtom or null if no valid RefAtom could be found.

[Atoms](#) come in two basic types. Data atoms like [NumAtoms](#) and [NameAtoms](#) contain actual data. [RefAtoms](#) contain a reference to an IndirectObject which contains another Atom.

Quite often you will want to resolve any RefAtoms and just obtain the RefAtom pointing to the final IndirectObject - you want a pointer to the final item of data rather than a reference

somewhere up the chain.

---

**Notes**

This function takes an Atom. If it is a RefAtom it finds the IndirectObject to which it points. It keeps doing this until it finds the final IndirectObject in the chain. It returns the RefAtom which points to this IndirectObject.

This method may return null if null is passed in, if the supplied Atom is not a RefAtom, if a RefAtom cannot be resolved or if a circular dependency is detected.

The reason this function is a member of the IndirectObject is because resolving RefAtoms requires access to the [ObjectSoup](#). The ObjectSoup is taken from the IndirectObject.

---

**Example**

None.



# ResolveObj Function

---

Resolves any indirect references and returns the final IndirectObject.

## [C#]

```
IndirectObject ResolveObj(Atom  
atom)
```

---

## Syntax

## [Visual Basic]

```
Function ResolveObj(atom As Atom)  
As IndirectObject
```

---

## Params

Name	Description
atom	The Atom to resolve.
return	The final IndirectObject or null if no valid object could be found.

[Atoms](#) come in two basic types. Data atoms like [NumAtoms](#) and [NameAtoms](#) contain actual data. [RefAtoms](#) contain a reference to an IndirectObject which contains another Atom.

Quite often you will want to resolve any RefAtoms and just obtain the final IndirectObject - you want a pointer to the final

item of data rather than a reference somewhere up the chain.

---

**Notes**

This function takes an Atom. If it is a RefAtom it finds the IndirectObject to which it points. It keeps doing this until it finds the final IndirectObject in the chain. It returns this IndirectObject.

This method may return null if null is passed in, if the supplied Atom is not a RefAtom, if a RefAtom cannot be resolved or if a circular dependency is detected.

The reason this function is a member of the IndirectObject is because resolving RefAtoms requires access to the [ObjectSoup](#). The ObjectSoup is taken from the IndirectObject.

---

**Example**

None.



# ToString Function

---

The string representation of the IndirectObject.

## [C#]

```
override string ToString()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

### Params

Name	Description
return	The string representation of the object.

This function derives the content of the object as it will be inserted into the final PDF document.

---

### Notes

Note that the the string value of an object may be large and it may contain unusual characters.

None.

---

### Example





# Transcode Function

---

Transcodes and reloads the IndirectObject

## [C#]

```
IndirectObject Transcode()
```

---

## Syntax

## [Visual Basic]

```
Function Transcode() As  
IndirectObject
```

---

## Params

Name	Description
return	An appropriate subclass of IndirectObject.

PDF objects are not hard typed. The only thing that distinguishes one from another are the named attributes which are applied to them. To enable an appropriate class of IndirectObject to be created ABCpdf has to examine the attributes on the base object and then create an appropriate subclass depending on those characteristics. This is known as transcoding.

Normally transcoding takes place automatically as objects are loaded or added. However

---

**Notes**

sometimes changes occur which may result in core aspects of the object changing. These changes may require caches to be regenerated or even an entirely new object to be created - one which better represents the new structure of the object. This is particular the case for the `FontObject` class as it is heavily dependent on caching and often on other objects in the `ObjectSoup`.

The `Transcode` method re-interprets this object and if it is appropriate, it returns a newly created object of an appropriate subclass. If transcoding results in a new object being created it will displace the current object from the soup. As such you will need to discard the old object if a new one is returned.

---

**Example**

None.

# ID Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The ID of the PDF object.

The unique ID of the PDF object.

---

Notes

None.

---

Example

# Gen Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The Generation of the PDF object.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The Generation of the PDF object.

---

Notes

None.

---

Example

# Atom Property

---



Type	Default	Read Only	Description
[C#] Atom			
[Visual Basic] Atom	n/a	No	The Atom contained by the IndirectObject.

The [Atom](#) contained by the IndirectObject.

Atoms can only be contained by one object at a time. So - if you assign a new Atom to this property and the supplied Atom is currently contained by another object - a clone of the Atom will be inserted rather than a reference to the original.

---

## Notes

None.

---

## Example

# Version Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The minimum version of the PDF specification required to support this object.
[Visual Basic] <code>Integer</code>	1	No	

The minimum version of the PDF specification required to support this object.

---

## Notes

For example a version of 3 indicates that the features specified by the object require a PDF 1.3 capable browser. Adobe Acrobat 4 was the first browser to support PDF 1.3.

---

## Example

None.

# Revision Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The revision of the document in which this object is stored.
[Visual Basic] <code>Integer</code>	0	Yes	

The revision of the document in which this object is stored.

PDF documents can be incrementally updated so that changes are appended to the document rather than overwriting the original. This means that it is possible to revert back to a previous version of the document. This feature is mostly used for document signatures so that the act of signing the document does not invalidate previous signatures.

---

## Notes

Objects from the oldest update will have a Revision of one and larger values indicate more recent updates. A value of zero indicates that the object was not read from stream.

None.

---

## Example



# Alive Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the IndirectObject is currently alive.

Whether the IndirectObject is currently alive.

---

## Notes

Objects contained within a PDF document are considered to be alive. Objects which have been created or cloned but have not yet been assigned to a PDF are not considered to be alive.

---

## Example

None.

# Doc Property

---



Type	Default	Read Only	Description
[C#] Doc			
[Visual Basic] Doc	n/a	Yes	The Doc containing this IndirectObject.

The [Doc](#) containing this IndirectObject.

---

## Notes

Note that although IndirectObjects are normally held inside a containing Doc object it is possible to detach them from the container. If this is the case the value of the Doc property will be null.

---

## Example

None.

# Soup Property

---



Type	Default	Read Only	Description
[C#] ObjectSoup	n/a	Yes	The soup containing this IndirectObject.
[Visual Basic] ObjectSoup			

The [ObjectSoup](#) containing this IndirectObject.

Note that although IndirectObjects are normally held inside a containing soup object it is possible to detach them from the container. If this is the case the value of the soup property will be null.

---

## Notes

None.

---

## Example

# AdbeExtLevel Property

---



Type	Default	Read Only	Description
[C#] <code>int[]</code> [Visual Basic] <code>Integer()</code>	null	No	The minimum extension level of Adobe Supplement to the PDF specification required to support this object.

The minimum extension level of Adobe Supplement to the PDF specification required to support this object.

The value must be null or contain two elements. If it is null, the object requires no Adobe extension (for getting), or the extension level is removed (for setting). The first element is the base version; the second element is the extension level.

---

## Notes

For example, a base version of 7 and an extension level of 3 indicate that the features specified by the object require a PDF consumer supporting base version 1.7 and extension level 3. Adobe Reader 9.1 was the first browser to support it.

---

**Example**

None.



# Dispose Function

---

Dispose of the object.

## [C#]

```
void Dispose()  
protected void Dispose(bool  
disposing)
```

---

### Syntax

## [Visual Basic]

```
Sub Dispose()  
Protected Sub Dispose(disposing  
As Boolean)
```

---

### Params

Name	Description
none	

You can call this function to explicitly dispose of an object and reduce the garbage collection overhead.

This method follows the standard design pattern for objects implementing the IDisposable interface. The protected Dispose method can be overridden for sub-classes wishing to dispose of additional objects.

---

### Notes

Do not attempt to use an object after calling  
Dispose.

---

**Example**

None.



# CopyTo Function

---

Copies the objects in the Soup to an Array.

## [C#]

```
void CopyTo(IndirectObject[]  
array, int index)
```

---

## Syntax

## [Visual Basic]

```
Sub CopyTo(array As  
IndirectObject(), index As  
Integer)
```

---

## Params

Name	Description
array	The array that is the destination for the elements.
index	The zero-based index in array at which copying begins.

Copies the elements of the Soup to an array starting at a particular array index.

---

## Notes

The array must be one-dimensional and have zero-based indexing.

---

**Example**

None.



## Add Function

---

Adds an object to the Soup.

### [C#]

```
int Add(IndirectObject value)
```

---

### Syntax

### [Visual Basic]

```
Function Add(value As  
IndirectObject) As Integer
```

---

### Params

Name	Description
value	The IndirectObject to be added.
return	The position in which the new element was inserted.

This method adds an [IndirectObject](#) to the Soup.

An IndirectObject can exist in only one ObjectSoup at a time. If the object supplied is already contained in another ObjectSoup then a Clone of the object is inserted.

---

### Notes

When an IndirectObject is inserted the [ID](#) is updated to reflect the position of the object

within the Soup.

Example

None.



# Clear Function

---

Removes all objects from the Soup.

## [C#]

```
void Clear()
```

---

### Syntax

## [Visual Basic]

```
Sub Clear()
```

---

### Params

Name	Description
none	

Removes all IndirectObjects from the Soup.

---

### Notes

Only the Null object at array index zero is preserved.

---

### Example

None.



# Contains Function

---



Determines whether the Soup contains a specific object.

## [C#]

```
bool Contains(IndirectObject  
value)
```

---

## Syntax

## [Visual Basic]

```
Function Contains(value As  
IndirectObject) As Boolean
```

---

## Params

Name	Description
value	The object to locate.
return	True if the object is found, otherwise false.

---

## Notes

Determines whether the Soup contains a specific IndirectObject.

---

## Example

None.





# IndexOf Function

---

Determines the index of a specific object.

## [C#]

```
int IndexOf(IndirectObject value)
```

---

### Syntax

## [Visual Basic]

```
Function IndexOf(value As IndirectObject) As Integer
```

---

### Params

Name	Description
value	The object to locate in the Soup.
return	If found, the index of value, otherwise -1.

---

### Notes

Determines the index of a specific IndirectObject in the Soup.

---

### Example

None.



# Insert Function

---



Inserts an object into the Soup at the specified position.

## [C#]

```
void Insert(int index,  
IndirectObject value)
```

## [Visual Basic]

Syntax

```
Sub Insert(index As Integer,  
value As IndirectObject)
```

- may throw `NotSupportedException()`

Params

Name	Description
index	The zero-based index at which value should be inserted.
return	The Object to insert into the Soup.

Objects within a collection refer to each other by index. This means that once an object is inserted into the Soup it must stay at the same index. If it moves then references may be

broken.

---

**Notes**

Insertion at a particular position might require other objects to be moved. For this reason this method always throws a `NotSupportedException`.

None.

---

**Example**



# Remove Function

---

Removes an object from the Soup.

## [C#]

```
bool Remove(IndirectObject value)
```

---

### Syntax

## [Visual Basic]

```
Function Remove(value As  
IndirectObject) As Boolean
```

---

### Params

Name	Description
value	The IndirectObject to be removed.
return	True if the IndirectObject is removed, otherwise false.

When an object is removed it leaves a gap in the Soup. Other objects do not move to fill the gap.

[RefAtoms](#) pointing to the object which was removed are not updated. This means you can remove one object and replace it with a substitute. Other IndirectObjects in the Soup will then refer to the new object rather than the old one.

---

### Notes

However it also means that you should be careful about removing an object and not replacing it. Because the slot is free it may be re-used and any references which exist may then point to a new - inappropriate - object.

---

**Example**

None.



## RemoveAt Function

---

Removes an object at a specified position from the Soup.

### [C#]

```
void RemoveAt(int index)
```

### [Visual Basic]

```
Sub RemoveAt(index As Integer)
```

Syntax

- may throw `ArgumentOutOfRangeException()`

Params

Name	Description
index	The zero-based index of the item to remove.

If the index is not valid then an `ArgumentOutOfRangeException` will be thrown.

When an object is removed it leaves a gap in the Soup. Other objects do not move to fill the gap.

[RefAtoms](#) pointing to the object which was

---

**Notes**

removed are not updated. This means you can remove one object and replace it with a substitute. Other IndirectObjects in the Soup will then refer to the new object rather than the old one.

However it also means that you should be careful about removing an object and not replacing it. Because the slot is free it may be re-used and any references which exist may then point to a new - inappropriate - object.

---

**Example**

None.



# GetEnumerator Function

---

Gets an enumerator for the Soup.

## [C#]

```
IEnumerator<IndirectObject>  
GetEnumerator()
```

---

### Syntax

## [Visual Basic]

```
Function GetEnumerator() As  
IEnumerator(Of IndirectObject)
```

---

### Params

Name	Description
return	The enumerator for the collection.

---

### Notes

Gets an IndirectObject enumerator for the Soup.

---

### Example

None.

# Count Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of items in the Soup.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The number of items in the Soup.

As objects are added to the collection the Count will increase to accomodate them.

---

## Notes

The collection may contain null values as well as instantiated IndirectObjects.

None.

---

## Example

# Item Property

---



Type	Default	Read Only	Description
<b>[C#]</b> <code>IndirectObject</code> <code>this[int index]</code>			Gets or sets the object at the specified index.
<b>[Visual Basic]</b> Default Property <code>Item(index As Integer) As IndirectObject</code>	n/a	No	

- may throw `ArgumentOutOfRangeException()`

Gets or sets the `IndirectObject` at the specified index. In C# this property is the indexer for the class.

An `IndirectObject` can exist in only one `ObjectSoup` at a time. If the object supplied is already contained in another `ObjectSoup` then a Clone of the object is inserted.

---

## Notes

When an `IndirectObject` is inserted the `ID` is updated to reflect the position of the object within the `Soup`.

---

Example

None.

# Catalog Property

---



Type	Default	Read Only	Description
[C#] Catalog			
[Visual Basic] Catalog	n/a	Yes	The Catalog for the document.

The [Catalog](#) for the document.

---

## Notes

The Catalog is the root of the whole PDF document. It contains information on the root [Pages](#) object and the [Outline](#) object.

---

## Example

None.

# Trailer Property

---



Type	Default	Read Only	Description
[C#] StreamObject	n/a	Yes	The Trailer or XRef for the document.
[Visual Basic] StreamObject			

The Trailer or XRef [StreamObject](#) for the document.

In PDF 1.4 documents have trailer dictionary. In PDF 1.5 documents have XRef streams. Internally ABCpdf only uses XRef streams but - when writing PDF 1.4 documents - it converts the XRef to a standard trailer before output.

---

## Notes

None.

---

## Example

# Revisions Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The number of incremental updates.

The number of incremental updates.

No object in the collection will have an `IndirectObject.Revision` higher than this.

---

## Notes

However it is possible, though unusual, for some revisions to have no objects.

None.

---

## Example



# ObjectSoupSubset Constructor

---

Construct an ObjectSoupSubset.

## [C#]

```
ObjectSoupSubset(ObjectSoup soup)
```

---

## Syntax

## [Visual Basic]

```
Sub New(soup As ObjectSoup)
```

---

## Params

Name	Description
soup	The soup from which objects will be selected.

Create an ObjectSoupSubset.

An ObjectSoupSubset is specific to a particular ObjectSoup and may only contain [IndirectObjects](#) from that soup.

---

## Notes

For this reason you should specify the soup in question at the point of construction. If later you attempt to add objects from a different soup then an exception will be raised.

---

**Example**

None.

# CopyTo Function

---



Copy the objects in this subset to a new soup while preserving the relationships between the items in the selection.

## [C#]

```
void CopyTo(ObjectSoup soup)
```

---

## Syntax

## [Visual Basic]

```
Sub CopyTo(soup As ObjectSoup)
```

Name	Description
soup	The destination soup.

---

## Params

Copy the objects in this subset to a new soup while preserving the relationships between the items in the selection.

Each [IndirectObject](#) in the selection has a unique ID. However the destination soup may already contain items with this ID. For this reason the process of copying the objects requires a remapping of old IDs to new ones in the destination soup.

---

## Notes

Some remap entries may already have been created during addition as the entries in the [RemapTypes](#) property get applied. However most will be created at the point that the CopyTo method is called. The remap table is accessible via the [RemapIDs](#) property.

---

**Example**

None.

# AddFamily Function

---



Add a parent object along with all objects it refers to.

## [C#]

```
int AddFamily(IndirectObject  
parent)  
int AddFamily(DictAtom dict)
```

## [Visual Basic]

Syntax

```
Function AddFamily(parent As  
IndirectObject) As Integer  
Function AddFamily(dict As  
DictAtom) As Integer
```

- may throw Exception()

Params

Name	Description
parent	The IndirectObject to be added.
dict	A dictionary atom to be added.
return	The number of objects that were added.

Add a parent object along with all objects it

refers to.

If a dictionary is specified there is no parent. In this situation only the objects referred to in the dictionary will be added.

---

**Notes**

If the parent or dictionary is not part of the soup specified in the [ObjectSoupSubset constructor](#) then an exception will be raised.

None.

---

**Example**

# AddOnlyOne Function

---



Add just this object ignoring any objects it may refer to.

## [C#]

```
bool AddOnlyOne(IndirectObject  
io)
```

## [Visual Basic]

Syntax

```
Function AddOnlyOne(io As  
IndirectObject) As Boolean
```

- may throw Exception()

Params

Name	Description
io	The IndirectObject to be added.
return	True if the item was added.

Add a parent object ignoring any objects it may refer to.

If the item was added this function returns true. If the item was not added because the subset already contained it or because the

---

**Notes**

**RemapTypes** specified a different remapping then the function returns false.

If the specified object is not part of the soup specified in the **ObjectSoupSubset** constructor then an exception will be raised.

---

**Example**

None.

# Objects Property

---



Type	Default	Read Only	Description
[C#] <code>ICollection&lt;IndirectObject&gt;</code>	n/a	Yes	The collecti of IndirectObj currently contained in the subset.
[Visual Basic] <code>ICollection&lt;IndirectObject&gt;</code>			

Gets the collection of IndirectObjects currently contained in the subset.

---

## Notes

None.

---

## Example

## RemapIDs Property

---



Type	Default	Read Only	Description
<b>[C#]</b> IDictionary<int, int>  <b>[Visual Basic]</b> IDictionary<int, int>	n/a	Yes	Gets the dictionary used to map object IDs from the source soup to new object IDs in the final soup.

Gets the dictionary used to map object IDs from the source soup to new object IDs in the final soup.

---

### Notes

Some mappings may be populated via the [RemapTypes](#) as [IndirectObjects](#) are added. Most will be populated at the point that the [CopyTo](#) method is called.

None.

---

### Example



## RemapTypes Property

---



Type	Default	Read Only	Description
[C#] IDictionary<string, int>	n/a	Yes	A dictionary used to redirect all objects of a specific type to a specific object ID.
[Visual Basic] IDictionary<string, int>			

A dictionary used to redirect all objects of a specific type to a specific object ID.

For example a stamp annotation is an object that describes a rubber stamp image that floats over a page. This object will contain links to other objects which will describe features like the appearance of the stamp. However it also contains a link to the page on which it is located.

Suppose you want to copy this annotation to another document. When you call [AddFamily](#) with this annotation you want to include related items like the annotation appearance but you don't want to include the page on which it is located. In fact you actually want links to this page in the source document to be redirected to a different page in the final destination document.

By specifying a mapping from the "Page" object type to the ID of a page in the final destination you both stop looking for linked objects at the point at which a page is discovered and also allow references to the source page to be linked to the destination page at the point that [CopyTo](#) is called.

To be precise, each time an IndirectObject is encountered it is checked for a "Type" entry. If this entry matches an item in the RemapTypes dictionary then the object is not added and no items to which it refers are added. Instead an item is added to the [RemapIDs](#) property specifying a mapping between the ID of the object in question and the ID specified in the RemapTypes entry.

---

**Example**

None.



## Focus Function

---

Prepare document for drawing at the annotation location

### [C#]

```
bool Focus()
```

---

### Syntax

### [Visual Basic]

```
Function Focus() As Boolean
```

---

### Params

Name	Description
return	True if the focus operation was successful.

Use this method to focus on the Annotation.

This prepares the document for drawing at the Annotation location.

If the operation was successful then the function returns true. If not then it will return false.

---

### Notes

The [Doc.Page](#), [Doc.Rect](#) and [Doc.Transform](#) may all be changed as a result of calling this method.

---

**Example**

None.



## GetFieldOptions Function

---

The field options for any form field associated with this annotation.

### [C#]

```
string[] GetFieldOptions()
```

---

### Syntax

### [Visual Basic]

```
Function GetFieldOptions() As  
String()
```

---

### Params

Name	Description
return	The list of field options.

This function gets the field options for any form field associated with this annotation.

Typically you assign a value using the [FieldValue](#). Some fields such as Text fields will accept any value. Others such as Checkboxes and List Boxes accept only a limited range of options. You can obtain these options using this method.

The unmarked state of a Checkbox or Radio

---

**Notes**

Button is always "Off". The marked state varies and is available via this function. This function will always return a one item array for this type of field.

The set of options for a Combo Box or List Box is available via this property. This function will return as many items as there are possible values.

Pushbuttons, Signatures and Text fields do not have options.

---

**Example**

None.



# Stamp Function

---

Stamp this annotation into the page.

## [C#]

```
void Stamp()
```

---

### Syntax

## [Visual Basic]

```
Sub Stamp()
```

---

### Params

Name	Description
none	

Use this method to permanently stamp an annotation into the page on which it is located.

When this method is called the annotation appearance is stamped permanently into the document and the annotation is deleted.

---

### Notes

The annotation becomes a new layer on the page (see [Doc.LayerCount](#)) so you may wish to call [Doc.Flatten](#) on the affected page.

---

Example

None.

# UpdateAppearance Function

---



Update the Appearance Stream for this annotation.

## [C#]

```
void UpdateAppearance()
```

---

### Syntax

## [Visual Basic]

```
Sub UpdateAppearance()
```

---

### Params

Name	Description
none	n/a.

Update the Appearance Stream for this Annotation.

Annotations often have appearance streams which define how they appear on the page when displayed or printed. ABCpdf understands a variety of annotation types and can create an appropriate stream when requested.

---

### Notes

Calling this function will result in the appearance stream being updated or (if one does not already exist) created.

---

**Example**

None.

# Border Property

---



Type	Default Value	Read Only	Description
[C#] ArrayAtom			
[Visual Basic] ArrayAtom	n/a	No	The border appearance

The border determines the visible appearance of the border around the annotation.

The border is an ArrayAtom containing three or four elements. The first three elements describe the horizontal corner radius, the vertical corner radius and the border width. If the corners are zero then the border has rectangular rather than rounded corners. If the border width is zero then there is no border.

The last optional element is a dash array to be used in drawing dashed borders. This conforms to the standard dash array format which indicates alternately the length of lines and gaps. So [2 1] would indicate two on, one off, two on, one off... Longer arrays can be used for more complex dash patterns.

---

## Notes

If this property is null then a solid rectangular border of width one will be drawn. This is equivalent to the following code.

**[C#]**

---

```
annot.Border =  
(ArrayAtom)Atom.FromString("[0 0 1]");
```

### [Visual Basic]

```
annot.Border =  
DirectCast(Atom.FromString("[0 0  
1]"), ArrayAtom)
```

After changing this property you will need to call [UpdateAppearance](#) to realize the change.

None.

---

**Example**

# Contents Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The visible text of the annotation.

The visible text of the annotation.

---

## Notes

If the annotation subtype does not display text then this field should contain an alternative description of the annotation contents.

---

## Example

None.

# FieldBackgroundColor Property

---



Type	Default Value	Read Only	Description
[C#] XColor			
[Visual Basic] XColor	n/a	No	The background color of the field.

This property is used to access or change the background color of a field.

This property is only valid if this annotation is associated with a form field.

If you assign a color to this property the color supplied must be grayscale, RGB or CMYK. A value of null indicates that the background is transparent.

---

## Notes

When you set this property a clone of the XColor you assign is created to avoid one XColor being shared by multiple annotations.

After changing this property you will need to call [UpdateAppearance](#) to realize the change.

None.

---

## Example



# FieldBorderColor Property

---



Type	Default Value	Read Only	Description
[C#] XColor			
[Visual Basic] XColor	n/a	No	The border color of the field.

This property is used to access or change the border color of a field.

This property is only valid if this annotation is associated with a form field.

If you assign a color to this property the color supplied must be grayscale, RGB or CMYK. A value of null indicates that the border is transparent.

---

## Notes

When you set this property a clone of the XColor you assign is created to avoid one XColor being shared by multiple annotations.

After changing this property you will need to call [UpdateAppearance](#) to realize the change.

None.

---

## Example



# FieldRotation Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>	n/a	No	The rotation of the annotation in degrees counterclockwise to the page.
[Visual Basic] <code>Integer</code>			

This property is used to access or change the rotation of the annotation in degrees counterclockwise to the page. This rotation must be a multiple of 90.

This property is only valid if this annotation is associated with a form field.

If you assign a color to this property the color supplied must be grayscale, RGB or CMYK. A value of null indicates that the border is transparent.

---

## Notes

When you set this property a clone of the XColor you assign is created to avoid one XColor being shared by multiple annotations.

After changing this property you will need to call [UpdateAppearance](#) to realize the change.

---

**Example**

None.

# FieldType Property

---



Type	Default	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	n/a	Yes	The field type for any form field associated with this annotation.

The field type for any form field associated with this annotation.

The following field types are available:

- "Pushbutton"
- "Checkbox"
- "Radio"
- "Text"
- "List"
- "Combo"
- "Signature"
- "Unknown"

---

## Notes

More details of these field types can be found in the Section 8.6.3 of the [Adobe PDF Specification](#).

None.

---

## Example



# FieldValue Property

---



Type	Default	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	n/a	No	The field value for any form field associated with this annotation.

The field value for any form field associated with this annotation.

You may wish to assign or query the form field value using this property.

Checkboxes and Radio Buttons have a value which is either "Off" or the on-state of the control as accessible via [GetFieldOptions](#).

---

## Notes

Text fields have a free-text value.

Combo Boxes and List Boxes have values restricted to a set of selections as accessible via [GetFieldOptions](#).

Pushbuttons and Signatures do not have a value.

None.

---

## Example



# Flags Property

---



Type	Default Value	Read Only	Description
[C#] AnnotationFlags	n/a	No	The Annotation Flags entry
[Visual Basic] AnnotationFlags			

This property is used to access or change the flags for the annotation.

The AnnotationFlags type is a flags type enumeration so the different values can be combined together using bitwise operations. It may take the following values:

- Invisible (Do not display the annotation if it does not belong to one of the standard types and no special handler is available)
- Hidden (Do not display or print the annotation or allow it to interact with the user, regardless of its type or whether an annotation handler is available)
- Print (Print the annotation when the page is printed. Typically used to hide pushbuttons when a page is printed)
- NoZoom (Do not scale the annotation as the zoom level of the page is changed. The top left of the annotation on the page remains fixed)

---

## Notes

- regardless of the page zoom level)
- NoRotate (Do not rotate the annotation as the page is rotated. The top left of the annotation on the page remains fixed regardless of the page rotation)
  - NoView (Do not show the annotation on the screen or allow it to interact with the user. The annotation may be printed if the Print flag is set)
  - ReadOnly (Do not allow the annotation to interact with the user)
  - Locked (Do not allow the annotation to be moved, deleted or otherwise modified. However the content may be changed even if this property is set)
  - ToggleNoView (Invert the interpretation of the NoView flag for certain events)
  - LockedContents (Do not allow the contents of the annotation to be modified. Other annotation properties such as size and location may be modified)

For code showing how to check, set or clear flags see the [FontObject.Flags](#) property.

---

## Example

None.

# FullName Property

---



Type	Default	Read Only	Description
[C#] string			The full name of any form field associated with this annotation.
[Visual Basic] String	n/a	Yes	

The full name of any form field associated with this annotation.

---

## Notes

Note that the PDF specification does not require that full names are unique.

---

## Example

None.

# Page Property

---



Type	Default	Read Only	Description
[C#] Page			
[Visual Basic] Page	n/a	No	The Page on which this annotation is located.

The [Page](#) on which this annotation is located.

---

Notes

None.

---

Example

# Rect Property

---



Type	Default	Read Only	Description
[C#] <code>XRect</code> [Visual Basic] <code>XRect</code>	n/a	No	The rectangle which defines the position and area of the annotation on the page.

- may throw `NullReferenceException()`

The `XRect` which defines the position and area of the annotation on the page.

This rectangle is encoded in PDF coordinates rather than any abstracted coordinate space.

---

## Notes

This property must always have a value. Attempting to assign a null value to this property will result in a `NullReferenceException` being thrown.

None.

---

## Example



# SubType Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The sub-type of Annotation.

The sub-type of annotation.

---

## Notes

This may be 'Text', 'Link', 'Widget' or any of the many supported types listed in Section 8.4.5 of the [Adobe PDF Specification](#).

---

## Example

None.

# TextDirection Property

---



Type	Default Value	Read Only	D
[C#] <code>XTextStyle.DirectionType</code>	DirectionType.Default	No	T te d
[Visual Basic] <code>XTextStyle.DirectionType</code>			

This property specifies the default primary text direction. For the details of each value, please refer to [XTextStyle.Direction](#).

---

## Notes

Its value is not saved to the PDF file. It does not support RightTo and mainly allows you to enable the support for ligatures and right left text (in left-to-right primary text direction) while the appearance stream is updated.

---

## Example

None.



# CopyTo Function

---

Copies the Bookmarks into an array.

## [C#]

```
void CopyTo(Bookmark[] array, int index)
```

---

## Syntax

## [Visual Basic]

```
Sub CopyTo(array As Bookmark(), index As Integer)
```

---

## Params

Name	Description
array	The array that is the destination for the elements.
index	The zero-based index in array at which copying begins.

Copies the elements of the Collection to an array starting at a particular array index.

---

## Notes

The array must be one-dimensional and have zero-based indexing.

---

Example

None.



## Add Function

---

Adds a Bookmark to the end of the list.

### [C#]

```
int Add(Bookmark bookmark)  
int Add(string title)
```

### [Visual Basic]

```
Function Add(bookmark As  
Bookmark) As Integer  
Function Add(title As String) As  
Integer
```

Syntax

Params

Name	Description
bookmark	The bookmark to be added.
title	The title for the bookmark to be added.
return	The position in which the new element was inserted.

This method adds an item to the end of the list.

You can add a Bookmark directly or you can use one of the overloaded operators to add a bookmark with a specified title.

When you add a string this is encapsulated within a new Bookmark which is then inserted.

---

**Notes**

Bookmarks can exist in only one place at a time. If the Bookmark supplied is already contained by another object then a **Clone** of the Bookmark is added.

None.

---

**Example**



# Clear Function

---

Removes all Bookmarks from the list.

## [C#]

```
void Clear()
```

---

### Syntax

## [Visual Basic]

```
Sub Clear()
```

---

### Params

Name	Description
none	

---

### Notes

Removes all Bookmarks from the list.

---

### Example

None.

# Contains Function

---



Determines whether the list contains a specific Bookmark.

## [C#]

```
bool Contains(Bookmark value)
```

---

## Syntax

## [Visual Basic]

```
Function Contains(value As  
Bookmark) As Boolean
```

---

## Params

Name	Description
value	The object to locate.
return	True if the object is found, otherwise false.

---

## Notes

Determines whether the Collection contains a specific Bookmark.

---

## Example

None.





# IndexOf Function

---

Determines the index of a specific Bookmark.

## [C#]

```
int IndexOf(Bookmark value)  
int IndexOf(string value)
```

---

## Syntax

## [Visual Basic]

```
Function IndexOf(value As  
Bookmark) As Integer  
Function IndexOf(value As String)  
As Integer
```

---

## Params

Name	Description
value	The object or the title of the object to locate in the Collection.
return	If found, the index of value, otherwise -1.

---

## Notes

Determines the index of a specific Bookmark in the list.

---

Example

None.

# Insert Function

---



Inserts a Bookmark into the list at the specified position.

## [C#]

```
void Insert(int index, Bookmark  
bookmark)  
void Insert(int index, string  
title)
```

## [Visual Basic]

Syntax

```
Sub Insert(index As Integer,  
bookmark As Bookmark)  
Sub Insert(index As Integer,  
title As String)
```

- may throw `ArgumentOutOfRangeException()`

Params

Name	Description
index	The zero-based index at which value should be inserted.
bookmark	The bookmark to be added.
title	The title for the bookmark to be added.

Inserts an item into the list at the specified position.

You can insert a Bookmark directly or you can use one of the overloaded operators to insert a bookmark with a specified title.

When you insert a string this is encapsulated within a new Bookmark which is then inserted.

---

**Notes**

Bookmarks can exist in only one place at a time. If the Bookmark supplied is already contained by another object then a **Clone** of the Bookmark is added.

If the index equals the number of items in the array then the bookmark is appended to the end.

If the index is not a valid index this method throws an `ArgumentOutOfRangeException`.

---

**Example**

None.



# Remove Function

---

Removes a Bookmark from the list.

## [C#]

```
bool Remove(Bookmark value)
```

---

### Syntax

## [Visual Basic]

```
Function Remove(value As  
Bookmark) As Boolean
```

---

### Params

Name	Description
value	The Bookmark to be removed.
return	True if the Bookmark is removed, otherwise false.

---

### Notes

When a Bookmark is removed the elements that follow the removed element move up to occupy the vacated spot.

---

### Example

None.





## RemoveAt Method

---

Remove the Bookmark at the specified location.

### [C#]

```
void RemoveAt(int index)
```

---

### Syntax

### [Visual Basic]

```
Sub RemoveAt(index As Integer)
```

---

### Params

Name	Description
index	The zero based index specifying the Bookmark to be removed.

Use this method to remove the Bookmark at the specified location.

---

### Notes

The Bookmarks that follow the removed Bookmark move up to occupy the empty spot.

None.

---

### Example





# GetEnumerator Function

---

Gets an enumerator for the Collection.

## [C#]

```
IEnumerator<Bookmark>  
GetEnumerator()
```

---

### Syntax

## [Visual Basic]

```
Function GetEnumerator() As  
IEnumerator(Of Bookmark)
```

---

### Params

Name	Description
return	The enumerator for the collection.

---

### Notes

Gets a Bookmark enumerator for the Collection.

---

### Example

None.



# Adopt Method

---

Adopt a specified Bookmark.

## [C#]

```
void Adopt(Bookmark value)
```

## [Visual Basic]

```
Sub Adopt(value As Bookmark)
```

- may throw Exception()

---

### Syntax

---

### Params

Name	Description
value	The bookmark to be adopted.

Use this method to move bookmarks within the bookmark hierarchy.

The bookmark to be adopted is detached from the bookmark hierarchy. Then it is added at the end of the bookmark collection for the current object.

---

### Notes

Some Adopt operations are not possible. For example a child cannot adopt its parent. In these cases the field structure will be left

unchanged and an exception thrown.

Example

None.



## Refresh Method

---

Refresh and reload the document Bookmarks.

### [C#]

```
void Refresh()
```

---

### Syntax

### [Visual Basic]

```
Sub Refresh()
```

Name	Description
return	n/a.

---

### Params

Use this method to refresh and reload all Bookmarks in the list.

When a bookmark is first referenced the bookmark data from the PDF is cached. This allows a level of optimization which would not otherwise be possible.

---

### Notes

However if you are using the low level functionality to modify the bookmark structure the cache will not reflect your changes. In this situation you can force the bookmarks to be reloaded by calling Refresh.

---

**Example**

None.

# Count Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>	See description.	Yes	The number of Bookmarks in the collection.
[Visual Basic] <code>Integer</code>			

The number of Bookmarks in the Collection.

As with all Collections you can use the Count property to determine the number of items contained and you can iterate through the collection using the standard methods appropriate to the language you are coding in.

---

## Notes

None.

---

## Example

# Item Property

---



Type	Default	Read Only	Description
<b>[C#]</b> Bookmark this[int index]			Get or set the Bookmark at the specified index.
<b>[Visual Basic]</b> Default Property Item(index As Integer) As Bookmark	n/a	No	

- may throw `ArgumentOutOfRangeException()`

Gets or sets the Bookmark at the specified index. In C# this property is the indexer for the class.

---

## Notes

If the index is not a valid index this property throws an `ArgumentOutOfRangeException`.

---

## Example

None.



# Open Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether the bookmark appears open or closed.

This property determines if the bookmark appears open or closed.

When a bookmark is open all its children are visible in a tree below it. When it is closed its children are hidden. Clients can toggle the state of the bookmark by clicking on it.

---

## Notes

This property determines the default state.

None.

---

## Example

# Page Property

---



Type	Default Value	Read Only	Description
[C#] Page			
[Visual Basic] Page	0	No	The destination Page associated with this bookmark.

When activated a bookmark will trigger an action. In most cases this will be a navigation action resulting in a new page being displayed.

The Page object for the destination page is available via this property.

---

## Notes

Occasionally you may come across bookmarks which have more complicated actions. In these cases this property will be null.

None.

---

## Example

# PageID Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	0	No	The destination Page ID associated with this bookmark.

When activated a bookmark will trigger an action. In most cases this will be a navigation action resulting in a new page being displayed.

The Page ID for the destination page is available via this property.

---

## Notes

Occasionally you may come across bookmarks which have more complicated actions. In these cases this property will be zero.

None.

---

## Example

# Parent Property

---



Type	Default Value	Read Only	Description
[C#] Bookmark			
[Visual Basic] Bookmark	n/a	Yes	The parent of this Bookmark.

Bookmarks exist in a hierarchy.

This property allows you to find the bookmark above this one in the hierarchy.

---

## Notes

The top level outline item has no parent so in this case the property will be null.

---

## Example

None.

# Title Property

---



Type	Default Value	Read Only	Description
[C#] string	""	No	The bookmark title to be displayed on screen.
[Visual Basic] String	""	No	The bookmark title to be displayed on screen.

This property determines the title of the bookmark.

---

## Notes

The title is the text that appears in the bookmarks tab.

---

## Example

None.

# AnalyzeContent Function

---



Perform whole document analysis of page contents.

## [C#]

```
void AnalyzeContent()
```

---

### Syntax

## [Visual Basic]

```
Function AnalyzeContent()
```

---

### Params

Name	Description
none	

This function scans the document and performs analysis of the contents.

Scanning the document content involves decompressing and parsing all the content in the document. As such it can be an expensive and time consuming process for large or complex documents.

---

### Notes

For this reason the analysis is cached the first time this function is called. If you later change the content it is your responsibility to call this

function to ensure that the analysis is updated.

Whole document analysis enables certain other optimizations such as [Font.Subset](#).

---

**Example**

None.



## GetEmbeddedFiles Function

---

Gets all the embedded files in this document

### [C#]

```
IDictionary<string,  
FileSpecification> GetEmbeddedFiles()
```

---

### Syntax

### [Visual Basic]

```
Function GetEmbeddedFiles() As  
IDictionary<string,  
FileSpecification>
```

Name	Description
return	A set of names and the file associated with each name.

---

### Params

Gets all the embedded files in this document.

PDF documents can contain a set of named files. These files can then be referenced by name elsewhere in the document.

The most common case in which named files are used is in the case of PDF portfolios. Adobe now uses the name Portfolio for this type of document

but in the PDF specification they are known as Portable Collections. While the names may vary two are the same.

---

#### Notes

PDF Portfolios appear as one PDF document but actually contain many. Typically the first document represents the visible face of the collection when opened. After the collection is open you can switch between the embedded PDFs.

PDF portfolios can be used to represent a folder of PDFs. For example you might find a portable collection being used to display multiple emails within a single PDF Portfolio.

The example below show how to extract all the files embedded in a PDF portfolio. See the [FileSpecification constructor](#) for the creation of portfolios.

#### [C#]

```
using (Doc doc = new Doc()) {
    XReadOptions ro = new XReadOptions();
    ro.OpenPortfolios = false;
    doc.Read("Portfolio1.pdf", ro);
    var files =
doc.ObjectSoup.Catalog.GetEmbeddedFiles();
    foreach (var pair in files) {
        FileSpecification fileSpec =
pair.Value;
        fileSpec.Rationalize();
        EmbeddedFile file =
fileSpec.EmbeddedFile;
        if ((file != null) &&
(file.Decompress())) {
            string name = "Portfolio_" +
```

```

fileSpec.Uri;
    string path = name;
    File.WriteAllBytes(path,
file.GetData());
    }
}
}

```

**Example**

### **[Visual Basic]**

```

Sub...
    Using doc As New Doc()
        Dim ro As New XReadOptions()
        ro.OpenPortfolios = False
        doc.Read("Portfolio1.pdf", ro)
        Dim files =
doc.ObjectSoup.Catalog.GetEmbeddedFiles()
        For Each pair As var In files
            Dim fileSpec As FileSpecification =
pair.Value
                fileSpec.Rationalize()
                Dim file__1 As EmbeddedFile =
fileSpec.EmbeddedFile
                    If (file__1 IsNot Nothing) AndAlso
(file__1.Decompress()) Then
                        Dim name As String = "Portfolio_"
+ fileSpec.Uri
                            Dim path As String = name
                                File.WriteAllBytes(path,
file__1.GetData())
                                    End If
                                Next
                            End Using
                        End Sub

```





## GetFieldNames Function

---

The names of all the eForm fields in the document.

### [C#]

```
string[] GetFieldNames()
```

---

### Syntax

### [Visual Basic]

```
Function GetFieldNames() As  
String()
```

---

### Params

Name	Description
return	The names of all the eForm fields in the document.

---

### Notes

This function scans the field tree and extracts the full name of every field in the document.

---

### Example

None.





# GetFields Function

---

Get all the eForm fields in the document.

## [C#]

```
IndirectObject[] GetFields()
```

---

### Syntax

## [Visual Basic]

```
Function GetFields() As  
IndirectObject()
```

---

### Params

Name	Description
return	All the eForm fields in the document.

This function scans the field tree and builds an array of all the fields in the document.

---

### Notes

Most fields are [Annotations](#) which means they have a visible appearance on the page. All fields are [IndirectObjects](#).

None.

---

### Example





# GetFonts Function

---

Gets all the fonts in this document.

## [C#]

```
FontObject[] GetFonts()
```

---

## Syntax

## [Visual Basic]

```
Function GetFonts() As  
FontObject()
```

---

## Params

Name	Description
return	All the fonts in the document.

---

## Notes

This function scans the document and builds an array of all the fonts.

---

## Example

None.

# Outline Property

---



Type	Default	Read Only	Description
[C#] Outline			
[Visual Basic] Outline	n/a	Yes	The root Outline object.

The root [Outline](#) object.

---

## Notes

The Outline object holds the structure of the document.

---

## Example

None.

# Pages Property

---



Type	Default	Read Only	Description
[C#] Pages			
[Visual Basic] Pages	n/a	Yes	The root Pages object.

The root [Pages](#) object.

---

## Notes

The Pages object holds references to every page in the document.

---

## Example

None.



# ColorSpace Constructor

---

Construct a ColorSpace.

## [C#]

```
ColorSpace(ObjectSoup soup)  
ColorSpace(ObjectSoup soup,  
ColorSpaceType space)
```

---

## Syntax

## [Visual Basic]

```
Sub New(soup As ObjectSoup)  
Sub New(soup As ObjectSoup, space  
As ColorSpaceType)
```

---

## Params

Name	Description
soup	The ObjectSoup to contain the newly created StreamObject.
space	The type of color space required.

Create a ColorSpace.

---

## Notes

If a color space is not specified the default of DeviceRGB will be used.

See the [PixMap.Recolor](#) function.

---

**Example**

## ColorSpaceType Property

---



Type	Default	Read Only	Description
[C#] ColorSpaceType	n/a	No	The type of color space.
[Visual Basic] ColorSpaceType			

The ColorSpaceType enumeration may take the following values:

- None
- DeviceGray
- DeviceRGB
- DeviceCMYK
- CalGray
- CalRGB
- ICCBased
- Lab
- Indexed
- Pattern
- Separation
- DeviceN

Device color spaces are device dependent color spaces. This means that output color will vary depending on the output medium. For example a DeviceRGB value may look different on one monitor than it does on another.

---

**Notes**

Cal, ICC and Lab color spaces are device independent color spaces. They try to define colors in terms of how they should look rather than how they should be produced. The goal is to allow colors to be reproduced accurately on different devices within the capabilities of the destination device.

The Indexed color space is used for palettized color. Each item in the palette is defined in terms of a base color space such as DeviceRGB. Palettes can hold up to 256 entries. The base color space can be determined using the [BaseColorSpaceType](#) property.

Pattern color spaces are special color spaces used for defining repeating patterns.

Separation and DeviceN color spaces define colors in terms of different base colors. Separations specify one color only. DeviceN color spaces define multiple different colors. For example using a DeviceN color space you could define an image to be printed using a combination of Gold, Silver and Black inks.

The None color space represents an unknown or undefined color space. As such you cannot set this property to be None. Attempting to do so will result in a default DeviceRGB color space.

More details of these color space types can be found in Section 4.5 of the [Adobe PDF Specification](#).

---

**Example**

None.



# Components Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of color components in the color space.
[Visual Basic] <code>Integer</code>	n/a	Yes	

Different color spaces have different numbers of components.

For example RGB color spaces have three components and CMYK color spaces have four.

This property allows you to determine the number of color components in the color space.

Indexed color spaces define colors via a palette. So although a color value is determined via a single index into a palette, the number of components is determined by the number of color components in the palette.

---

## Notes

For example the number of components in an indexed RGB color space is three.

Pattern color spaces are special and have no components.

None.

---

**Example**

# IccProfile Property

---



Type	Default	Read Only	Description
[C#] IccProfile	n/a	No	Any ICC Color Profile associated with this color space.
[Visual Basic] IccProfile			

Any [IccProfile](#) describing an ICC Color Profile associated with this color space.

Querying the value of this property will never raise an exception.

Assigning a new value to this property automatically results in the [IccProfile](#) being decompressed, validated, recompressed and (if it is not already there) added to the same [ObjectSoup](#) as the [ColorSpace](#).

---

## Notes

The [ColorSpaceType](#) and [Components](#) properties will change to reflect the color space of the new profile.

An exception will be thrown if the assignment is not possible. This may happen if the [ColorSpace](#) is not in an [ObjectSoup](#) or if the [IccProfile](#) is not valid.

---

**Example**

See the [PixMap.Recolor](#) function.

## BaseColorSpaceType Property

---



Type	Default	Read Only	Description
[C#] ColorSpaceType	n/a	Yes	The base type of color space.
[Visual Basic] ColorSpaceType			

The base color space is relevant for cases in which the [ColorSpaceType](#) property is Indexed.

An Indexed color space contains a palette of colors indexed by number. Each item in the palette is a color in a different color space. As such the Indexed color space has a base color space in which the palette colors are defined.

---

### Notes

This property allows you to determine the base color space for an Indexed color space. If the color space is not Indexed then this property is equal to the [ColorSpaceType](#) property.

None.

---

### Example



# Name Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	Any name associated with the color space.

Any name associated with the color space.

---

## Notes

Names are relevant only for Separation color spaces. They typically refer to the name of a spot color.

---

## Example

None.

# Gamma Property

---



Type	Default Value	Read Only	Description
[C#] <code>ArrayAtom</code>			The gamma correction for the color space
[Visual Basic] <code>ArrayAtom</code>	n/a	No	

The gamma correction for the color space.

This property is only valid for color spaces in which the `ColorSpaceType` is `CalGray` or `CalRGB`.

Gamma is defined for each component of a calibrated color space. So for a `CalGray` color space it has one entry and for a `CalRGB` color space it has three.

---

## Notes

This means that for a `CalGray` color space this property will be a `NumAtom` and for a `CalRGB` color space it will be an `ArrayAtom` with three entries.

Each gamma entry must be positive and is generally greater than or equal to one.

In this example we show how to use the Gamma property with a calibrated grayscale color space.

[C#]

---

```

using (Doc doc = new Doc()) {
    doc.Width = 80;
    doc.Rect.Inset(50, 50);
    ColorSpace cs = new
ColorSpace(doc.ObjectSoup,
ColorSpaceType.CalGray);
    ((NumAtom)cs.Gamma).Real = 1.2;
    doc.ColorSpace = cs.ID;
    doc.Color.SetComponents(0.9); // gray
    doc.AddOval(true);
    doc.Save("examplecalgraycolorspace.pdf");
}

```

### [Visual Basic]

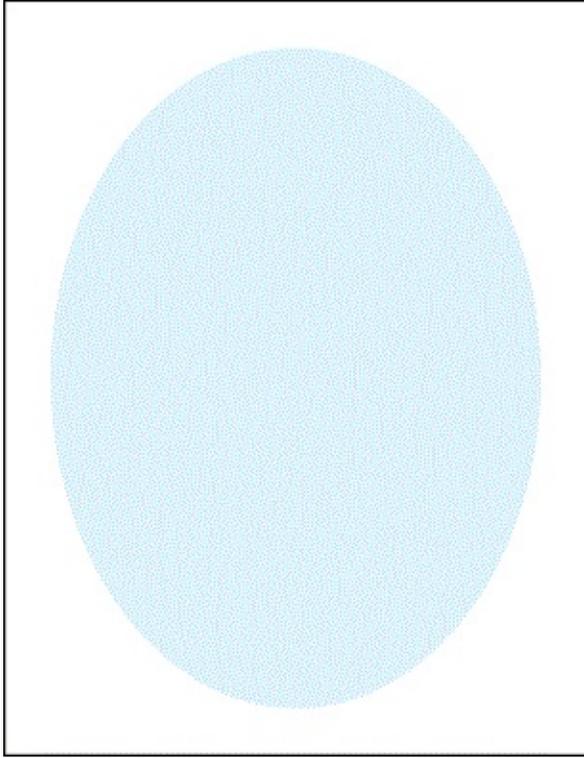
```

Using doc As New Doc()
    doc.Width = 80
    doc.Rect.Inset(50, 50)
    Dim cs As New ColorSpace(doc.ObjectSoup,
ColorSpaceType.CalGray)
    DirectCast(cs.Gamma, NumAtom).Real = 1.2
    doc.ColorSpace = cs.ID
    doc.Color.SetComponents(0.9)
    ' gray
    doc.AddOval(True)
    doc.Save("examplecalgraycolorspace.pdf")
End Using
End Sub

```

---

Example



examplecalgraycolorspace.pdf

# BlackPoint Property

---



Type	Default Value	Read Only	Description
[C#] XColor			
[Visual Basic] XColor	n/a	No	The black point for the color space specified in CIE 1931 XYZ space

The black point for the color space specified in CIE 1931 XYZ space.

This property is only valid for color spaces in which the [ColorSpaceType](#) is CalGray, CalRGB or Lab.

---

## Notes

All components for this valud must be non-negative. The default for this peroperty is 0 0 0.

When you set this property a clone of the XColor you assign is created to avoid one XColor being shared by multiple color spaces.

---

## Example

See the [WhitePoint](#) property.

# WhitePoint Property

---



Type	Default Value	Read Only	Description
[C#] XColor			
[Visual Basic] XColor	n/a	No	The white point for the color space specified in CIE 1931 XYZ space

The white point for the color space specified in CIE 1931 XYZ space.

This property is only valid for color spaces in which the [ColorSpaceType](#) is CallGray, CalRGB or Lab.

---

## Notes

X and Y must be positive and Z must be one. The default for this property is 0 0 0.

When you set this property a clone of the XColor you assign is created to avoid one XColor being shared by multiple color spaces.

In this example we show how to use the WhitePoint and [BlackPoint](#) with a calibrated RGB color space.

### [C#]

```
using (Doc doc = new Doc()) {
```

```

doc.Width = 80;
doc.Rect.Inset(50, 50);
ColorSpace cs = new
ColorSpace(doc.ObjectSoup,
ColorSpaceType.CalRGB);
    cs.WhitePoint.SetComponents(0.9, 1.0,
1.1);
    cs.BlackPoint =
XColor.FromComponents(0.1, 0.1, 0.1);
    doc.ColorSpace = cs.ID;
    doc.Color.SetComponents(0.9, 0.1, 0.1);
// red
doc.AddOval(true);
doc.Save("examplecalrgbcolorspace.pdf");
}

```

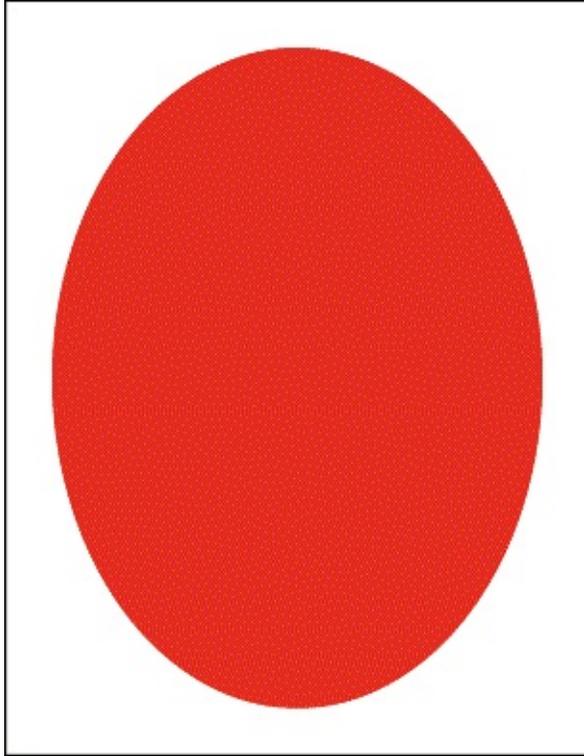
### [Visual Basic]

```

Using doc As New Doc()
    doc.Width = 80
    doc.Rect.Inset(50, 50)
    Dim cs As New
ColorSpace(doc.ObjectSoup,
ColorSpaceType.CalRGB)
    cs.WhitePoint.SetComponents(0.9, 1.0,
1.1)
    cs.BlackPoint =
XColor.FromComponents(0.1, 0.1, 0.1)
    doc.ColorSpace = cs.ID
    doc.Color.SetComponents(0.9, 0.1, 0.1)
    ' red
    doc.AddOval(True)
    doc.Save("examplecalrgbcolorspace.pdf")
End Using
End Sub

```

Example



examplecalrgbcolorspace.pdf



# EmbeddedFile Function

---

EmbeddedFile Constructor

## [C#]

```
EmbeddedFile(ObjectSoup soup)
EmbeddedFile(ObjectSoup soup,
string path)
EmbeddedFile(ObjectSoup soup,
byte[] data)
```

---

## Syntax

## [Visual Basic]

```
Sub New(soup As ObjectSoup)
Sub New(soup As ObjectSoup, path
As String)
Sub New(soup As ObjectSoup,
data() As Byte)
```

---

## Params

Name	Description
soup	The ObjectSoup to contain the newly created EmbeddedFile.
path	A path to a file containing data which should be placed in the EmbeddedFile.
data	An array of bytes which should be placed in the EmbeddedFile.

Create an EmbeddedFile.

This constructor which takes a path to a file will embed and compress the file data using using the [CompressFlate](#) function. It will also insert appropriate metadata using the [UpdateMetadata](#) function.

---

**Notes**

The constructor which takes an array of data will similarly embed and compress the data. However since there is no file path metadata will not be assigned.

---

**Example**

None.



# UpdateMetadata Function

---

Update metadata for the embedded file.

## [C#]

```
void UpdateMetadata(string path)
```

## [Visual Basic]

```
Sub UpdateMetadata(path As  
String)
```

### Syntax

- may throw `Exception()`

### Params

Name	Description
path	A path to a file containing data. If null is passed the data will be cleared.

Update metadata like the [ModificationDate](#), [CreationDate](#) and [Size](#) to reflect the file values.

### Notes

This `EmbeddedFile` must be part of an `ObjectSoup` or an exception will be thrown.

---

**Example**

None.

# Checksum Property

---



Type	Default Value	Read Only	Description
[C#] string	null	No	The 16 byte checksum for the embedded file
[Visual Basic] String			

The 16 byte checksum for the embedded file.

---

## Notes

If there is no such checksum the value will be null.

---

## Example

None.

# CreationDate Property

---



Type	Default Value	Read Only	Description
[C#] DateTime?			
[Visual Basic] DateTime?	null	No	The creation date of the embedded file

The creation date of the embedded file.

---

## Notes

If there is no such date the value will be null.

---

## Example

None.

# MacCreator Property

---



Type	Default Value	Read Only	Description
<code>[C#] int?</code> <code>[Visual Basic] Nullable(Of Integer)</code>	null	No	The Macintosh file creator - a four char code represented as an integer

The Macintosh file creator - a four char code represented as an integer.

---

## Notes

If there is no creator the value will be null.

---

## Example

None.

# MacResFork Property

---



Type	Default Value	Read Only	Description
[C#] StreamObject	null	No	The Macintosh resource forks stream for this file
[Visual Basic] StreamObject			

The Macintosh resource forks stream for this file.

---

## Notes

If there is no resource fork the value will be null.

---

## Example

None.

# MacType Property

---



Type	Default Value	Read Only	Description
[C#] int? [Visual Basic] Nullable(Of Integer)	null	No	The Macintosh file type - a four char code represented as an integer

The Macintosh file type - a four char code represented as an integer.

---

## Notes

If there is no type the value will be null.

---

## Example

None.

# ModificationDate Property

---



Type	Default Value	Read Only	Description
[C#] DateTime?			
[Visual Basic] DateTime?	null	No	The modification date of the embedded file

The modification date of the embedded file.

---

## Notes

If there is no such date the value will be null.

---

## Example

None.

# Size Property

---



Type	Default Value	Read Only	Description
[C#] <code>int?</code> [Visual Basic] <code>Nullable(Of Integer)</code>	null	No	The size of the embedded file

The size of the embedded file.

---

## Notes

If there is no size the value will be null.

---

## Example

None.

# Subtype Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	null	No	The subtype of the embedded file

The subtype of the embedded file.

---

## Notes

This property is represented via a [NameAtom](#) and hence it cannot take an empty string. As such, setting this property to an empty string will set it to null.

---

## Example

None.



## Focus Function

---

Prepare document for drawing at the field location.

### [C#]

```
bool Focus()
```

---

### Syntax

### [Visual Basic]

```
Function Focus() As Boolean
```

---

### Params

Name	Description
return	True if the focus operation was successful.

Use this method to focus on the field.

This prepares the document for drawing at the field location.

If the operation was successful then the function returns true. If the field has no visible location it will return false.

---

### Notes

The [Doc.Page](#), [Doc.Rect](#) and [Doc.Transform](#) may all be changed as a result of calling this method.

---

**Example**

None.



## GetAnnotations Function

---

Gets all the Annotations referenced by this field or its children

### [C#]

```
Annotation[] GetAnnotations()
```

---

### Syntax

### [Visual Basic]

```
Overridable GetAnnotations() As  
Annotation()
```

---

### Params

Name	Description
none	

---

### Notes

Gets all the [Annotations](#) referenced by this field or its children.

---

### Example

None.





## GetKids Function

---

Gets a set of Fields that are descendents of this one.

### [C#]

```
Field[] GetKids(int maximumDepth)
```

---

### Syntax

### [Visual Basic]

```
Function GetKids(maximumDepth As Integer) As Field()
```

---

### Params

Name	Description
maximumDepth	The maximum depth to search.

Gets a set of Fields that are descendents of this one.

The maximum depth parameter allows you to control the depth to which the field tree is searched. One for immediate children, two for children and grandchildren, etc.

---

### Notes

Set the maximumDepth to `Int32.MaxValue` if you wish to retrieve all the fields under this one.

---

**Example**

None.



## SetFont Function

---

Sets the font and font size to be used for text

### [C#]

```
virtual void SetFont(FontObject  
font, double size)
```

---

### Syntax

### [Visual Basic]

```
Sub SetFont(font As FontObject,  
size As Double)
```

---

### Params

Name	Description
font	The font to be assigned to the field.
size	The font size to be assigned to the field.

Sets the font and font size to be used for text.

This function works by changing the [DefaultAppearance](#) for the field using the specified [FontObject](#) and size.

Properties such as the [TextFont](#) and [TextSize](#) get the effective value which may be inherited. Calling this function sets the value on this item

itself which will override any inherited value.

---

**Notes**

Passing a font value of null will remove both the font and size from the item itself though a value may be still be present if it is inherited from a parent field or from document defaults.

A zero size font indicates that the font should scale so that it fits the size of the field. Negative font heights appear the same as positive ones but are best avoided as they are likely to cause confusion.

The appearance of the field is not updated until you change the [Value](#) or call [UpdateAppearance](#).

---

**Example**

None.



## Stamp Method

---

Stamp this field into the document.

### [C#]

```
void Stamp()
```

---

### Syntax

### [Visual Basic]

```
Sub Stamp()
```

Name	Description
return	n/a.

---

### Params

Use this method to permanently stamp a field into the document.

When this method is called the field appearance is stamped permanently into the document and the field is deleted.

---

### Notes

The field becomes a new layer on the page (see [Doc.LayerCount](#)) so you may wish to call [Doc.Flatten](#) on the affected page.

You can use the [XForm.Stamp](#) method to stamp all fields into the document.

---

**Example**

None.



## UpdateAppearance Function

---

Update the appearance of all the Annotations associated with this field

### [C#]

```
void UpdateAppearance()
```

---

#### Syntax

### [Visual Basic]

```
Sub UpdateAppearance()
```

---

#### Params

Name	Description
none	n/a

Update the appearance of all the [Annotations](#) associated with this field.

A field may have more than one appearance on one or more pages. Each appearance is represented as an Annotation. When a field is changed in a way which may impact the visual rendition on the page, the appearance of all the Annotations associated with that field need to be updated.

---

#### Notes

When you update the [Value](#) this happens

automatically. However if you change other properties such as the `TextColor` you will need to make an explicit call to ensure the appearance streams are updated.

---

**Example**

None.

# DefaultAppearance Property

---



Type	Default Value	Read Only	Description
[C#] ArrayAtom			
[Visual Basic] ArrayAtom	n/a	No	The default appearance (DA) used for the text

The default appearance (DA) used for the text.

The ArrayAtom is a sequence of PDF parameters and operators which makes up a complete drawing sequence.

This property controls the base set of drawing instructions used for controlling the appearance of variable text in a field. Getting this property gets the effective value which may be inherited. Setting this property sets the value on this item itself.

---

## Notes

Similarly setting this property to null will remove the property on the item itself but the value may continue to be inherited. The appearance of the field is not updated until you change the [Value](#) or call [UpdateAppearance](#).

---

None.

**Example**

# FieldType Property

---



Type	Default Value	Read Only	Description
[C#] FieldType			
[Visual Basic] FieldType	See description.	Yes	The field type.

The FieldType enumeration may take the following values:

- Pushbutton
- Checkbox
- Radio
- Text
- List
- Combo
- Signature
- Unknown

---

## Notes

More details of these field types can be found in Section 8.6.3 of the [Adobe PDF Specification](#).

None.

---

## Example



# Flags Property

---



Type	Default Value	Read Only	Description
[C#] FieldFlags	n/a	No	The Field Flags (Ff) entry
[Visual Basic] FieldFlags			

This property is used to access or change the flags (Ff entry) for the field.

The FieldFlags type is a flags type enumeration so the different values can be combined together using bitwise operations. It may take the following values:

- ReadOnly (Do not let the user change the value of the field)
- Required (Require that the field has a value at the time it is exported by a form submission action)
- NoExport (Do not export the value of the field as part of a form submission action)
- NoToggleToOff (This is used for radio buttons and specifies that exactly one button shall be selected at all times)
- Radio (Indicates whether the field is a radio button or a checkbox. This field is ignored if the Pushbutton flag is set)
- Pushbutton (Whether this field is a pushbutton)

that stores no permanent value)

- RadiosInUnison (Allow radio buttons with the same 'on' value to turn on and off together)
- Multiline (Allow a text field to contain multiple lines of text)
- Password (Display text field as a password box in which the text contents are obscured)
- FileSelect (Display a text field that allows a file to be selected. When the form is submitted the contents of the file will be used as the value of the field)
- DoNotSpellCheck (Text in this text or choice field shall not be spell checked)
- DoNotScroll (Text in this text field shall not scroll. Once text has been entered which fills the field it shall accept no further characters)
- Comb (Use the MaxLen property to equally space out text field characters into a comb-like slot position)
- RichText (The value of this text field shall be a rich text string)
- Combo (Whether this field is a combo or list field)
- Edit (Whether field shall present an editable text box in addition to a list)
- Sort (Whether entries in the list shall be put into alphabetical order)
- MultiSelect (Whether more than one entry in the list can be selected at the same time)
- CommitOnSelChange (A change to the value of this field will be committed as soon as it is made)

---

Notes

The ReadOnly, Required and NoExport flags are used by all field types.

The NoToggleToOff, Radio, Pushbutton and RadiosInUnison flags are used by button fields.

The Multiline, Password, FileSelect, DoNotSpellCheck, DoNotScroll, Comb and RichText flags are used by text fields.

The Combo, Edit, Sort, MultiSelect, DoNotSpellCheck and CommitOnSelChange are used by choice fields.

For code showing how to check, set or clear flags see the [FontObject.Flags](#) property.

---

**Example**

None.

# Format Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>	See description.	Yes	The field format.
[Visual Basic] <code>String</code>			

The format for this field.

The PDF specification does not define how fields should be formatted. Acrobat uses a set of standard JavaScripts to perform field formatting.

This property reflects the JavaScript formatting function which is being used and the values which are being passed to it. A typical value might be:

```
AFNumber_Format(2, 3, 0, 0, "", true);
```

---

## Notes

This defines a number format with two decimal places. For a precise understanding of the purposes of the other parameters you will need to see the JavaScripts which come installed with Acrobat.

Other typical formatting functions are `AFPercent_Format`, `AFSpecial_Format`, `AFDate_Format` and `AFTime_Format`. Again for full details you should see the JavaScripts which come

installed with Acrobat.

---

**Example**

None.

# Kids Property

---



Type	Default Value	Read Only	Description
[C#] Fields			
[Visual Basic] Fields	See description.	Yes	All the immediate children of this field.

This property holds the immediate children of this field.

Items in this collection can be referenced by partial field name or by zero based index.

---

## Notes

As with all collections you can use the Count property to determine the number of items contained and you can iterate through the collection using the standard methods appropriate to the language you are coding in.

---

## Example

None.

# MultiSelect Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	See description.	Yes	Whether the field supports multiple selections.

This property allows you to determine if the the field supports multiple values.

Most fields have a single value. A text box contains one item of text. A radio button group has one selected button. A checkbox can be checked or clear.

---

## Notes

However some kinds of fields can hold multiple values. For example a list box can be used to select multiple items. If this is the case then the MultiSelect property will be set to true.

If the field supports multiple selection then the **Value** of the field will be a comma delimited list of the items which are selected.

None.

---

## Example



# Name Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>	See description.	Yes	The fully qualified field name.
[Visual Basic] <code>String</code>			

The fully qualified field name for this field.

---

## Notes

See the [XForm](#) object for details of how fully qualified names are constructed.

---

## Example

None.

# Options Property

---



Type	Default Value	Read Only	Description
[C#] <code>string[]</code>	See description.	Yes	The field options.
[Visual Basic] <code>String()</code>			

The options available for the value of this field.

Typically you assign a value using the [Value](#) property. Some fields such as Text fields will accept any value. Others such as Checkboxes and List Boxes accept only a limited range of options. The field options tell you what options are available for a particular field.

---

## Notes

The unmarked state of a Checkbox or Radio Button is always "Off". The marked state varies and is available via item zero of this property.

The set of options for a Combo Box or List Box can be obtained directly via this property.

Pushbuttons, Signatures and Text fields do not have options.

None.

---

## Example



# Page Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">Page</a>	See description.	No	The Page on which the field appears.
[Visual Basic] <a href="#">Page</a>			

- may throw `ArgumentOutOfRangeException()`

This property reflects the page on which this field is located.

This property is only valid if the field has one single visible appearance on the page. If you are unsure whether your field will always map to one visible appearance it is better to call [GetAnnotations](#) and work with the [Annotation.Page](#) of each of the items in the array. If there is only one item in the array then this indicates that the field has one single visible appearance on the page.

The most common case in which a field does not have a single visible appearance is in the case of radio buttons. In this case the field describes the radio button group and the children of the field are the visible buttons. Be aware that it is possible to spread radio buttons across more than one page.

---

## Notes

This value may be null if the field does not have a visible appearance.

If this field does not have one single visible appearance on the page, attempting to assign a value to this property will result in an `ArgumentOutOfRangeException` being thrown.

---

**Example**

None.

## PageID Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	See description.	No	The ID of the page on which the field appears.

- may throw `ArgumentOutOfRangeException()`

This property reflects the page on which this field is located.

This property is only valid if the field has one single visible appearance on the page. If you are unsure whether your field will always map to one visible appearance it is better to call [GetAnnotations](#) and work with the [Annotation.Page](#) of each of the items in the array. If there is only one item in the array then this indicates that the field has one single visible appearance on the page.

---

### Notes

The most common case in which a field does not have a single visible appearance is in the case of radio buttons. In this case the field describes the radio button group and the children of the field are the visible buttons. Be aware that it is possible to spread radio buttons across more than one page.

This value may be zero if the field does not have a visible appearance.

If this field does not have one single visible appearance on the page, attempting to assign a value to this property will result in an `ArgumentOutOfRangeException` being thrown.

---

**Example**

None.

# Parent Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">Field</a>	See description.	Yes	The parent of this field.
[Visual Basic] <a href="#">Field</a>			

This property holds the parent of this field.

Each field may have a number of children which you can access using the [Kids](#) property. Each child will have one parent which you can access using the Parent property. Fields at the top of the hierarchy will have a null parent.

---

## Notes

None.

---

## Example

# PartialName Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	See description.	Yes	The partial field name.

The partial name for this field.

---

## Notes

See the [XForm](#) object for details of how partial and fully qualified names are related.

---

## Example

None.

# Rect Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">XRect</a>	See description.	No	The location and size of the field.
[Visual Basic] <a href="#">XRect</a>			

- may throw `NullReferenceException()`

- may throw `ArgumentOutOfRangeException()`

This property reflects the position and area of the field on the page.

This property is only valid if the fields has one single visible appearance on the page. If you are unsure whether your field will always map to one visible appearance it is better to call [GetAnnotations](#) and work with the [Annotation.Rect](#) of each of the items in the array. If there is only one item in the array then this indicates that the field has one one single visible appearance on the page.

The most common case in which a field does not have a single visible appearance is in the case of radio buttons. In this case the field describes the radio button group and the children of the field are

the visible buttons. Be aware that it is possible to spread radio buttons across more than one page.

This rectangle will be empty if the field does not have one single visible appearance on the page.

This property must always have a value. Attempting to assign a null value to this property will result in a `NullReferenceException` being thrown.

If this field has no visible appearance, attempting to assign a value will result in an `ArgumentOutOfRangeException` being thrown.

---

**Example**

None.

# TextAlignment Property

---



Type	Default Value	Read Only	Description
[C#] AlignmentType	n/a	No	The alignment for the text
[Visual Basic] AlignmentType			

The alignment for the text.

The AlignmentType enumeration may take the following values:

- Left
- Center
- Right

---

## Notes

Getting this property gets the effective value which may be inherited. Setting this property sets the alignment on this item itself.

This feature is referred to as quadding within the Adobe PDF Specification. We call it alignment because quadding is not normal terminology.

The appearance of the field is not updated until you change the [Value](#) or call [UpdateAppearance](#).

---

**Example**

None.

# TextColor Property

---



Type	Default Value	Read Only	Description
[C#] XColor			
[Visual Basic] XColor	n/a	No	The color used for the text

The color used for the text.

This is derived from the [DefaultAppearance](#) for the field.

Getting this property gets the effective value which may be inherited. Setting this property sets the value on this item itself. When you set this property a clone of the XColor you assign is created to avoid one XColor being shared by multiple fields.

---

## Notes

The appearance of the field is not updated until you change the [Value](#) or call [UpdateAppearance](#).

None.

---

## Example

# TextFont Property

---



Type	Default Value	Read Only	Description
[C#] FontObject	n/a	Yes	The font used for the text
[Visual Basic] FontObject			

The font used for the text.

The [FontObject](#) that comprises this property is derived from the [DefaultAppearance](#) for the field.

---

## Notes

Getting this property gets the effective value which may be inherited from a parent field or from document defaults. To change this property use the [SetFont](#) function.

---

## Example

None.

# TextSize Property

---



Type	Default Value	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The font size used for the text

The font size used for the text.

This is derived from the [DefaultAppearance](#) for the field.

Getting this property gets the effective value which may be inherited from a parent field or from document defaults. To change this property use the [SetFont](#) function.

---

## Notes

The value zero has a special significance and indicates that the text in the field should be scaled to fit the area available.

None.

---

## Example

# Value Property

---



Type	Default Value	Read Only	Description
[C#] string	""	No	The field value.
[Visual Basic] String	""	No	The field value.

This property allows you access to the value of the field.

You may wish to assign or query the form field value using this property.

Text fields have a free-text value.

Checkboxes and Radio Buttons have a value which is either "Off" or the on-state of the control as specified in the [Options](#).

Combo Boxes and List Boxes have values restricted to a set of selections as specified in the Options property.

Pushbuttons and Signatures do not have a value.

## Common Operations.

Set the value of a text field:

**[C#]**

```
theField.Value = "Mr Jones";
```

**[Visual Basic]**

```
theField.Value = "Mr Jones"
```

Set a checkbox:

**[C#]**

```
theField.Value =  
theField.Options[0];
```

**[Visual Basic]**

```
theField.Value =  
theField.Options(0)
```

---

Notes

Clear a checkbox:

**[C#]**

```
theField.Value = "Off";
```

**[Visual Basic]**

```
theField.Value = "Off"
```

Set the fifth Radio Button in a group:

**[C#]**

```
theField.Value =  
theField.Options[4];
```

### **[Visual Basic]**

```
theField.Value =  
theField.Options(4)
```

### **Multiline Form Fields Tip.**

When inserting carriage returns into multiline form fields use carriage returns (\r) only.

Using line feeds (\n) or a carriage return linefeed combination (\r\n) is less compatible with older versions of Acrobat.

None.

---

**Example**



# CopyTo Function

---

Copies the items into a collection.

## [C#]

```
void CopyTo(Field[] array, int index)
```

---

## Syntax

## [Visual Basic]

```
Sub CopyTo(array As Field(), index As Integer)
```

---

## Params

Name	Description
array	The array that is the destination for the elements.
index	The zero-based index in array at which copying begins.

Copies the elements of the collection to an array starting at a particular array index.

---

## Notes

The array must be one-dimensional and have zero-based indexing.

---

Example

None.



## Add Function

---

Add an item to the end of the collection.

### [C#]

```
int Add(Field field)
```

### [Visual Basic]

```
Function Add(field As Field) As Integer
```

---

#### Syntax

- may throw `NotSupportedException()`

Name	Description
field	The item to be added.

---

#### Params

This method adds an item to the collection.

All Fields collections are read only so calling this function will generate a `NotSupportedException`.

---

#### Notes

---

Example

None.



# Clear Function

---

Removes all items from the collection.

## [C#]

```
void Clear()
```

## [Visual Basic]

```
Sub Clear()
```

- may throw `NotSupportedException()`

---

### Syntax

---

### Params

Name	Description
none	

Removes all items from the collection.

All Fields collections are read only so calling this function will generate a `NotSupportedException`.

---

### Notes

None.

---

**Example**

# Contains Function

---



Determines whether the collection contains a specific item.

## [C#]

```
bool Contains(Field value)
```

---

### Syntax

## [Visual Basic]

```
Function Contains(value As Field)  
As Boolean
```

---

### Params

Name	Description
value	The object to locate.
return	True if the object is found, otherwise false.

---

### Notes

Determines whether the collection contains a specific item.

---

### Example

None.





# IndexOf Function

---

Determines the index of a specific item.

## [C#]

```
int IndexOf(Field value)
```

---

## Syntax

## [Visual Basic]

```
Function IndexOf(value As Field)  
As Integer
```

---

## Params

Name	Description
value	The object to locate in the collection.
return	If found, the index of value, otherwise -1.

---

## Notes

Determines the index of a specific item in the collection.

---

## Example

None.



# Insert Function

---



Inserts an item into the collection at the specified position.

## [C#]

```
void Insert(int index, Field  
value)
```

## [Visual Basic]

```
Sub Insert(index As Integer,  
value As Field)
```

Syntax

- may throw `NotSupportedException()`

Params

Name	Description
index	The zero-based index at which value should be inserted.
value	The item to insert into the array.

Inserts an item into the collection at the specified position.

All Fields collections are read only so calling

---

**Notes**

this function will generate a `NotSupportedException`.

---

**Example**

None.



# Remove Function

---

Removes an item from the collection.

## [C#]

```
bool Remove(Field value)
```

## [Visual Basic]

```
Function Remove(value As Field)  
As Boolean
```

---

### Syntax

- may throw `NotSupportedException()`

---

### Params

Name	Description
value	The item to be removed.
return	True if the item is removed, otherwise false.

Removes an item from the collection.

All Fields collections are read only so calling this function will generate a `NotSupportedException`.

---

### Notes

---

**Example**

None.



## RemoveAt Function

---

Removes an item at a specified position from the collection.

### [C#]

```
void RemoveAt(int index)
```

### [Visual Basic]

```
Sub RemoveAt(index As Integer)
```

---

#### Syntax

- may throw `NotSupportedException()`

---

#### Params

Name	Description
index	The zero-based index of the item to remove.

Removes an item from the collection.

---

#### Notes

All Fields collections are read only so calling this function will generate a `NotSupportedException..`

---

**Example**

None.



# GetEnumerator Function

---

Gets an enumerator for the Collection.

## [C#]

```
IEnumerator<Field>  
GetEnumerator()
```

---

### Syntax

## [Visual Basic]

```
Function GetEnumerator() As  
IEnumerator(Of Field)
```

---

### Params

Name	Description
return	The enumerator for the collection.

---

### Notes

Gets a Field enumerator for the Collection.

---

### Example

None.

# Count Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of items in the collection.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The number of items in the collection.

---

## Notes

All Fields collections are read only so this value will be constant.

---

## Example

None.

# Item Property

---



Type	Default	Read Only	Description
<b>[C#]</b> <code>Field this[int index]</code> <code>Field this[string index]</code>			Get or set the item at the specified index.
<b>[Visual Basic]</b> Default Property <code>Item(index As Integer) As Field</code> Default Property <code>Item(index As String) As Field</code>	n/a	No	

- may throw `ArgumentOutOfRangeException()`

- may throw `NotSupportedException()`

Gets or sets the item at the specified location. In C# this property is the indexer for the class.

The index specified may be a zero based numeric index. If the index is not a valid index this property throws an `ArgumentOutOfRangeException`.

---

**Notes**

Alternatively it may be a string specifying a field name. If the name is not one which matches a Field in the collection then a null value will be returned.

All Fields collections are read only so attempting to assign an item using this function will generate a NotSupportedException.

---

**Example**

None.



# FileSpecification Function

---

## FileSpecification Constructor

### [C#]

```
FileSpecification(ObjectSoup  
soup)  
FileSpecification(ObjectSoup  
soup, string path)
```

---

### Syntax

### [Visual Basic]

```
Sub New(soup As ObjectSoup)  
Sub New(soup As ObjectSoup, path  
As String)
```

---

### Params

Name	Description
soup	The ObjectSoup to contain the newly created FileSpecification.
path	A file to embed within the file specification.

Create a FileSpecification.

This constructor which takes a path to a file will embed and compress the file data using using the [CompressFlate](#) function. It will also insert

appropriate metadata using the [EmbeddedFileUpdateMetadata](#) function.

The constructor which takes an array of data will similarly embed and compress the data. However since there is no file path metadata will not be assigned.

The example below shows how to combine a set of PDF documents into a portfolio. See the [Catalog.GetEmbeddedFiles](#) function for how to extract files from a portfolio.

### [C#]

```
string[] files = {
    "Bentley.pdf",
    "Acrobat.pdf",
    "Noise.pdf",
    "ChecksModified.pdf",
};
using (Doc doc = new Doc()) {
    List<Tuple<string,
FileSpecification>> fileSpecs = new
List<Tuple<string,
FileSpecification>>();
    foreach (string file in files) {
        byte[] data = null;
        using (Doc subDoc = new Doc())
        {
            subDoc.Read(file);
            data = subDoc.GetData();
        }
        EmbeddedFile embedFile = new
EmbeddedFile(doc.ObjectSoup, data);
        embedFile.CompressFlate();
    }
}
```

```

        FileSpecification fileSpec =
new
FileSpecification(doc.ObjectSoup);
    fileSpec.EmbeddedFile =
embedFile;
    fileSpec.Uri = file;
    string name =
Path.GetFileName(file);
    fileSpecs.Add(new Tuple<string,
FileSpecification>(name,
fileSpec));
    }
    fileSpecs.Sort((a, b) =>
a.Item1.CompareTo(b.Item1));
    // we don't really need to delete
existing files but we do so as an
example
    doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Names:Del",
"");
    foreach (Tuple<string,
FileSpecification> fileSpec in
fileSpecs) {
        doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Names*
[]:Text", fileSpec.Item1);
        doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Names*
[]:Ref", fileSpec.Item2.ID);
    }
    doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Limits*
[]:Ref", fileSpecs[0].Item1);
    doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Limits*
[]:Ref", fileSpecs[fileSpecs.Count
- 1].Item1);

```

---

**Example**

```
doc.SetInfo(doc.Root,  
"/Collection*/D:Text", files[0]);  
doc.Save("createportfolio.pdf");  
}
```

**[Visual Basic]**

```
Sub ...  
    Dim files As String() =  
    {"Bentley.pdf", "Acrobat.pdf",  
    "Noise.pdf", "ChecksModified.pdf"}  
    Using doc As New Doc()  
        Dim fileSpecs As New List(Of  
    Tuple(Of String,  
    FileSpecification))()  
        For Each file As String In  
    files  
            Dim data As Byte() = Nothing  
            Using subDoc As New Doc()  
                subDoc.Read(file)  
                data = subDoc.GetData()  
            End Using  
            Dim embedFile As New  
    EmbeddedFile(doc.ObjectSoup, data)  
            embedFile.CompressFlate()  
            Dim fileSpec As New  
    FileSpecification(doc.ObjectSoup)  
            fileSpec.EmbeddedFile =  
    embedFile  
            fileSpec.Uri = file  
            Dim name As String =  
    Path.GetFileName(file)  
            fileSpecs.Add(New Tuple(Of  
    String, FileSpecification)(name,  
    fileSpec))  
        Next  
        fileSpecs.Sort(Function(a, b)
```

```

a.Item1.CompareTo(b.Item1))
    ' we don't really need to
delete existing files but we do so
as an example
    doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Names:Del",
"")
    For Each fileSpec As Tuple(Of
String, FileSpecification) In
fileSpecs
        doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Names*
[]:Text", fileSpec.Item1)
        doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Names*
[]:Ref", fileSpec.Item2.ID)
    Next
    doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Limits*
[]:Ref", fileSpecs(0).Item1)
    doc.SetInfo(doc.Root,
"/Names*/EmbeddedFiles*/Limits*
[]:Ref", fileSpecs(fileSpecs.Count
- 1).Item1)
    doc.SetInfo(doc.Root,
"/Collection*/D:Text", files(0))
    doc.Save("createportfolio.pdf")
End Using
End Sub

```



## GetPath Function

---

Get the path to the file for the specified platform

### [C#]

```
string GetPath(string platform)
```

---

### Syntax

### [Visual Basic]

```
Function GetPath(platform As String) As String
```

---

### Params

Name	Description
platform	The platform name.
return	The file path.

This method is provided for low level control over obsolete entries contained in old PDF documents.

So in general you should be using the [Rationalize](#) method and the [Uri](#) property rather than this method.

A FileSpecification may hold multiple paths to a file.

---

**Notes** Each path is specific to a platform such as Mac, Windows or Unix. For further details see the [Platform](#) property.

This function allows you to retrieve the path for the specified platform.

If there is no matching path then null will be returned.

---

**Example**

None.



# Rationalize Function

---

Removes any obsolescent and redundant entries.

## [C#]

```
void Rationalize()
```

---

### Syntax

## [Visual Basic]

```
Sub Rationalize()
```

---

### Params

Name	Description
none	n/a

Removes any obsolescent and redundant entries.

---

### Notes

This will convert the object to be compliant with the changes recommended in the PDF 1.7 standard.

---

### Example

None.





## SetPath Function

---

Set the path to the file for the specified platform

### [C#]

```
string SetPath(string platform,  
string path)
```

---

### Syntax

### [Visual Basic]

```
Function SetPath(platform As  
String, path As String) As String
```

---

### Params

Name	Description
platform	The platform name.
path	The file path.

This method is provided for low level control over obsolete entries contained in old PDF documents.

So in general you should be using the [Rationalize](#) method and the [Uri](#) property rather than this method.

A FileSpecification may hold multiple paths to a file.

---

**Notes**

Each path is specific to a platform such as Mac, Windows or Unix. For further details see the [Platform](#) property.

This function allows you to set the path for the specified platform.

Setting to null will remove the path.

---

**Example**

None.

# Description Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	null	No	A description of the file for use in a user interface

A description of the file for use in a user interface.

---

## Notes

None.

---

## Example

# EmbeddedFile Property

---



Type	Default Value	Read Only	Description
[C#] EmbeddedFile	null	No	The embedded file if one has been embedded in the PDF
[Visual Basic] EmbeddedFile			

The embedded file if one has been embedded in the PDF.

If a file has not been embedded then the URL needs to be interpreted in the context of the current PDF to locate the external file.

---

## Notes

Setting this property will clear the EmbeddedFiles property if one is present.

This property requires reference to other objects in the ObjectSoup. If this object is not part of an ObjectSoup this property will always be null.

None.

---

## Example



# EmbeddedFiles Property

---



Type	Default Value	Read Only	Description
<b>[C#]</b> Tuple<string, EmbeddedFile> []	null	No	The set of embedded files if a set has been embedded in the PDF
<b>[Visual Basic]</b> Tuple<string, EmbeddedFile> []			

The set of embedded files if a set has been embedded in the PDF.

Normally a file specification will refer to one file. However in some situations it may be necessary to allow it to refer to multiple. For example if a CMYK image is available as four color separations then it may be useful to be able to embed four different files to make up the whole data set. Each file will need a name so this property reflects both the name and the content of each file.

---

## Notes

If the EmbeddedFiles property has a value, then the first file in the array should always be the same as the EmbeddedFile property. For this reason setting this property will also set the EmbeddedFile property to be equal to the first file in the array. If there are no

items in the array then the EmbeddedFile property will be cleared.

This property requires reference to other objects in the ObjectSoup. If this object is not part of an ObjectSoup this property will always be null.

---

**Example**

None.

# Platform Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	null	No	The default platform being used for the file specification

The default platform being used for the file specification.

For historic reasons PDF has supported multiple different files for multiple different platforms. This was appropriate when files were often non-interoperable and would not port easily between machines.

Fortunately this state of affairs is now unusual and only the default "UF" platform is commonly used. Adobe do not specify what the initials stand for, but it is likely to be Unicode File, though it could be Universal File or URL File.

---

## Notes

While you are unlikely to wish to create documents which use these obsolescent platforms, you may find old documents that use them. This method allows you to access information specified by platform. Appropriate selectors are "UF", "F", "DOS", "Mac" or "Unix" as per the descriptions in the Adobe PDF Specification. You can use the Platforms property to determine which are being used.

The default of "UF" provides access via the standard settings as specified in the PDF 1.7 specification.

---

**Example**

None.

# Uri Property

---



Type	Default Value	Read Only	Description
[C#] string	null	No	The URI to the file
[Visual Basic] String			

- may throw Exception()

The URI to the file.

For the default platform, attempting to set this property to a value which is not a valid URI will result in an exception being thrown. To convert a standard Windows path to a URI use `(new System.Uri(path)).AbsoluteUri`.

---

## Notes

For some values of the Platform property (e.g. Mac, DOS, Unix) raw file paths can be used. However these are obsolete and while you may see PDFs which contain them, you would be inadvised to generate new documents containing these structures.

None.

---

## Example



# Volatile Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether the file changes frequently.

This property indicates whether the file changes frequently and needs to be reloaded each time it is used, or if it is not and may be cached.

---

## Notes

None.

---

## Example

# GetEncoding Function

---



Obtain a standard encoding dictionary for use with PDF text operators.

## [C#]

```
static Dictionary<char, char>  
GetEncoding(LanguageType type,  
bool vertical)
```

## [Visual Basic]

### Syntax

```
Shared Function GetEncoding(type  
As LanguageType, vertical As  
Boolean) As Dictionary(Of Char,  
Char)
```

- may throw Exception()

### Params

Name	Description
type	The language type for the desired encoding.
vertical	Whether the writing direction should be horizontal or vertical.
return	A dictionary mapping Unicode values to native character IDs.

Use this method to obtain a standard encoding dictionary for use with PDF text operators.

Note that the encoding dictionary is defined purely by the language and direction provided as arguments to this function. It is your responsibility to ensure that the returned encoding dictionary is only used with fonts that already have this encoding. If instead you need to encode text using the font encoding specific to this particular font see the [EncodeText](#) method.

---

**Notes**

Since the Unicode LanguageType is already Unicode the encoding for this type is identity. For this reason an exception will be thrown if an encoding for this type is requested.

See the [Fonts and Languages](#) section for details on language types.

None.

---

**Example**



## EmbedFont Function

---

Search for and embed a font file into this font object

### [C#]

```
bool EmbedFont()
```

---

### Syntax

### [Visual Basic]

```
Function EmbedFont() As Boolean
```

---

### Params

Name	Description
return	Whether the font was embedded successfully.

Search for and embed a font file into this font object.

If an existing font file is already embedded it will be replaced. If you wish to subset the font you will need to call the [Subset](#) method after embedding the font.

---

### Notes

Calling this function on [Identity](#) encoded fonts should be done with great caution as these types of fonts rely on glyph indexes and are

thus specific to a particular font file.

---

**Example**

None.



## EncodeText Function

---

Encode text for use with PDF text operators.

### [C#]

```
string EncodeText(string text,  
EncodingSupport support)  
string EncodeText(string text,  
EncodingSupport support, out int  
outCount)
```

### [Visual Basic]

```
Function EncodeText(text As  
String, support As  
EncodingSupport)  
Function EncodeText(text As  
String, support As  
EncodingSupport, <Out> ByRef  
outCount As Integer)
```

Syntax

- may throw Exception()

Name	Description
text	The text to be encoded.
	The way in which characters not

---

**Params**

support	included in the font are to be encoded.
outCount	The number of characters in (parameter) text that are encoded in the return value.
return	The PDF string containing the encoded text.

Use this method to convert Unicode text into PDF string for use with the Tj or the TJ operator.

The EncodingSupport enumeration can take any of the following values:

- All – encode all text.
- NonelfAnyNotSupported – encode nothing if any character is not included in the font.
- SupportedPrefix – encode up to and excluding the first character not included in the font.

---

**Notes**

Note that the encoding for a font is constructed and cached at the time the font is first created. So changing the encoding for the font will not result in a corresponding change of behavior from this method. If this is something you need to do then you will need to create an entirely new font.

---

**Example**

None.



# RegenerateToUnicode Function

---

Attempt to regenerate a ToUnicode map.

## [C#]

```
bool RegenerateToUnicode()
```

---

## Syntax

## [Visual Basic]

```
Function RegenerateToUnicode() As Boolean
```

---

## Params

Name	Description
return	Whether a new ToUnicode map was inserted.

Attempt to regenerate a ToUnicode map using embedded font data as a key.

Complex fonts are generally embedded using an identity encoding which references characters by glyph index rather than character code. Because glyph indexes are specific to a particular font, to extract text from the PDF you need a table mapping the glyph indexed to character codes. This map is called the ToUnicode map.

---

**Notes**

Well behaved PDF producers will insert a ToUnicode map for all fonts that they embed. However some font producers may fail to do so. This means that copying or extracting text from such a PDF will result in gibberish.

In some cases it is possible to regenerate the ToUnicode map from the embedded font data. This function attempts to do so and returns true if a new ToUnicode map was inserted. This function only works on **composite** fonts and calling it on other types of fonts will not do anything.

---

**Example**

None.



## Subset Function

---

Subset a previously embedded font.

### [C#]

```
bool Subset()  
bool Subset(string characters)
```

---

### Syntax

### [Visual Basic]

```
Function Subset() As Boolean  
Function Subset(characters as  
String) As Boolean
```

---

### Params

Name	Description
characters	The character set to be used for the subsetting.
return	Whether subsetting was performed successfully.

Use this method to subset a previously embedded font.

Many PDF producers are not efficient at subsetting fonts. Subsetting previously embedded fonts can be a time consuming

operation but can also result in a significant reduction in file size.

---

**Notes**

If you provide a character set string then this is a relatively fast operation. However if you do not pass a character set then one has to be constructed and to do this it is necessary to analyze the entire document to determine what characters are being used. In this case if a call to [Catalog.AnalyzeContent](#) has not already been made then one will be triggered.

This call will subset embedded TrueType fonts. If the font is referenced or if the font is not TrueType then this function will return false.

---

**Example**

None.



## UnembedFont Function

---

Unembed the font if this is a simple operation.

### [C#]

```
int UnembedSimpleFont()
```

---

#### Syntax

### [Visual Basic]

```
UnembedSimpleFont() As Integer
```

---

#### Params

Name	Description
return	The number of bytes of font data that were removed as a result of this call.

Unembed the font if this is a simple operation.

Latin based fonts can often be unembedded as a simple and fast operation. However Unicode and symbolic fonts are more complex to unembed and cannot be simply removed from the document. To unembed a composite font you need to use the [ReduceSizeOperation](#) class.

---

#### Notes

This function returns the number of bytes of font data that were removed as a result of this call. Note that it is possible for multiple font objects

to reference the same embedded font. In this case each call to unembed will return the same number and the actual font will only actually be removed from the document if all references to it are released.

---

**Example**

None.

# BaseFont Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] string	n/a	Yes	The PostScript name of the font.

---

## Notes

The PostScript name of the font.

---

## Example

None.

# EmbeddedFont Property

---



Type	Default	Read Only	Description
[C#] StreamObject	n/a	Yes	The embedded font file.
[Visual Basic] StreamObject			

The embedded font file.

---

## Notes

Font files may be in a number of different formats such as TrueType, OpenType or Type 1 - PostScript.

---

## Example

None.

# IsComposite Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] bool	n/a	Yes	Whether the font is a complex composite font.

Whether the font is a complex composite font.

---

## Notes

Composite fonts are typically used for non-ASCII characters and complex writing systems.

---

## Example

None.

# IsIdentity Property

---



Type	Default Value	Read Only	Description
[C#] bool			Whether the font is a composite glyph encoded identity font
[Visual Basic] Boolean	n/a	Yes	

Whether the font is a composite glyph encoded identity font.

The normal way to identify a character in a font is to use a standard such as ASCII. Language encodings such as [Latin](#) work this way.

For characters outside this range the PDF format allows you to reference font glyphs directly using the glyph index within the font. This is known as an identity encoding. Language encodings such as [Unicode](#) work this way.

---

## Notes

Identity encodings are very flexible but they do hardwire the document to a particular font file and as such the font must be embedded. Care should be taken when operating on identity encoded embedded font files.

None.

---

**Example**

# IsSubset Property

---



Type	Default	Read Only	Description
[C#] bool			Whether the font contains an embedded subset.
[Visual Basic] bool	n/a	Yes	

Whether the font contains an embedded subset.

---

## Notes

None.

---

## Example

# IsVertical Property

---



Type	Default	Read Only	Description
[C#] <code>bool</code>			
[Visual Basic] <code>bool</code>	n/a	Yes	Whether the font is for vertical writing.

Whether the font is for vertical writing.

Some languages may be written vertically instead of horizontally. As well as the direction of flow often there are often specific forms of characters for the different directions.

---

## Notes

None.

---

## Example

# WritingMode Property

---



Type	Default	Read Only	Description
[C#]WritingModeType			
[Visual Basic]WritingModeType	n/a	Yes	Gets the font writing mode.

Gets the font writing mode.

The WritingModeType enum has two possible values:

---

## Notes

- Horizontal
- Vertical

---

## Example

None.

# Widths Property

---



Type	Default	Read Only	Description
[C#] IDictionary<char, int>	n/a	Yes	The widths of the characters in the font.
[Visual Basic] IDictionary<char, int>			

The character widths for all the characters in the font.

The array is indexable by Unicode value. For example to find the width of a space (ASCII 32) you would simply reference item 32 in the collection. The values are measured in 1000ths of a PDF unit.

---

## Notes

Font subsetting can result in characters being partially or completely removed from a font. ABCpdf will attempt to ensure that the Widths collection only contains entries for valid characters. However sometimes it is difficult to tell and as such you cannot rely on a character being present simply because it has a width.

The example below shows how to add text on a curve

and text flowing round a circle.

### [C#]

```
void Example() {
    string theFont = "Comic Sans MS";
    string theText = "Gallia est omnis
divisa in partes tres, quarum unam
incolunt Belgae...";
    string theTitle = "Commentarii de
Bello Gallico";
    using (Doc doc = new Doc()) {
        doc.FontSize = 36;
        doc.TextStyle.Kerning =
XTextStyle.KerningType.None;
        doc.Font = doc.EmbedFont(theFont,
LanguageType.Latin, false, true, false);
        // add some radial text in the top
middle of the page
        double cx = doc.MediaBox.Width *
0.5;
        double cy = doc.MediaBox.Height *
0.6;
        double r = doc.MediaBox.Width * 0.3;
        CurvedText.AddRadial(doc, theText,
cx, cy, r, 225, true, false);
        // add some curved text to a
rectangle
        double width =
doc.MeasureText(theTitle);
        doc.Rect.SetRect(100, 100, width,
doc.FontSize * 1.5);
        doc.FrameRect();
        CurvedText.AddCurved(doc, theTitle);
        // save
        doc.Save("ExampleCurvedText.pdf");
    }
}
```

```

class CurvedText {
    public static void AddCurved(Doc doc,
string text) {
        double halfWidth = doc.Rect.Width /
2;
        double height = doc.Rect.Height -
doc.TextStyle.Size;
        double radius = ((halfWidth *
halfWidth) + (height * height)) / (2 *
height);
        double centerX = doc.Rect.Left +
halfWidth;
        double centerY = doc.Rect.Bottom +
radius + doc.TextStyle.Size;
        double alpha = Math.Asin(halfWidth /
radius) - Math.PI;
        AddRadial(doc, text, centerX,
centerY, radius,
RadiansToDegrees(alpha), true, false);
    }

    public static void AddRadial(Doc doc,
string text, double centerX, double
centerY, double radius, double
startAngleDegrees, bool inside, bool
clockwise) {
        FontObject font =
doc.ObjectSoup[doc.Font] as FontObject;
        IDictionary<char, int> widths =
font.Widths;
        int n = text.Length;
        double a =
DegreesToRadians(startAngleDegrees);
        double fontWidthToRadians =
doc.TextStyle.Size / (radius * 1000);
        doc.Rect.String =

```

```

doc.MediaBox.String;
    for (int i = 0; i < n; i++) {
        // work out position
        double x = centerX + (Math.Sin(a)
* radius);
        double y = centerY + (Math.Cos(a)
* radius);
        // add a character
        doc.Pos.X = x;
        doc.Pos.Y = y;
        doc.Transform.Reset();
        double charRotation = inside ?
RadiansToDegrees(-a) + 180 :
RadiansToDegrees(-a);
        doc.Transform.Rotate(charRotation,
x, y);
        doc.AddText(text[i].ToString());
        // increment angle
        double da =
Convert.ToDouble(widths[text[i]]) *
fontWidthToRadians;
        a += clockwise ? da : -da;
    }
    doc.Transform.Reset();
}

private static double
DegreesToRadians(double degrees) {
    return degrees * Math.PI / 180;
}

private static double
RadiansToDegrees(double radians) {
    return radians * 180 / Math.PI;
}
}

```

## [Visual Basic]

Example

```
Sub ...
    Dim theFont As String = "Comic Sans
MS"
    Dim theText As String = "Gallia est
omnis divisa in partes tres, quarum unam
incolunt Belgae..."
    Dim theTitle As String = "Commentarii
de Bello Gallico"
    Using doc As New Doc()
        doc.FontSize = 36
        doc.TextStyle.Kerning =
XTextStyle.KerningType.None
        doc.Font = doc.EmbedFont(theFont,
LanguageType.Latin, False, True, False)
        ' add some radial text in the top
middle of the page
        Dim cx As Double =
doc.MediaBox.Width * 0.5
        Dim cy As Double =
doc.MediaBox.Height * 0.6
        Dim r As Double = doc.MediaBox.Width
* 0.3
        CurvedText.AddRadial(doc, theText,
cx, cy, r, 225, _
        True, False)
        ' add some curved text to a
rectangle
        Dim width As Double =
doc.MeasureText(theTitle)
        doc.Rect.SetRect(100, 100, width,
doc.FontSize * 1.5)
        doc.FrameRect()
        CurvedText.AddCurved(doc, theTitle)
        ' save
        doc.Save("ExampleCurvedText.pdf")
    End Using
```

```
End Sub
```

```
Private Class CurvedText
```

```
    Public Shared Sub AddCurved(doc As  
Doc, text As String)
```

```
        Dim halfWidth As Double =  
doc.Rect.Width / 2
```

```
        Dim height As Double =  
doc.Rect.Height - doc.TextStyle.Size
```

```
        Dim radius As Double = ((halfWidth *  
halfWidth) + (height * height)) / (2 *  
height)
```

```
        Dim centerX As Double =  
doc.Rect.Left + halfWidth
```

```
        Dim centerY As Double =  
doc.Rect.Bottom + radius +
```

```
doc.TextStyle.Size
```

```
        Dim alpha As Double =  
Math.Asin(halfWidth / radius) - Math.PI
```

```
        AddRadial(doc, text, centerX,  
centerY, radius,
```

```
RadiansToDegrees(alpha), _  
True, False)
```

```
    End Sub
```

```
    Public Shared Sub AddRadial(doc As  
Doc, text As String, centerX As Double,
```

```
centerY As Double, radius As Double,  
startAngleDegrees As Double, _
```

```
inside As Boolean, clockwise As  
Boolean)
```

```
    Dim font As FontObject =  
TryCast(doc.ObjectSoup(doc.Font),  
FontObject)
```

```
    Dim widths As IDictionary(Of Char,  
Integer) = font.Widths
```

```
    Dim n As Integer = text.Length
```

```

    Dim a As Double =
DegreesToRadians(startAngleDegrees)
    Dim fontWidthToRadians As Double =
doc.TextStyle.Size / (radius * 1000)
    doc.Rect.[String] = doc.MediaBox.
[String]
    For i As Integer = 0 To n - 1
        ' work out position
        Dim x As Double = centerX +
(Math.Sin(a) * radius)
        Dim y As Double = centerY +
(Math.Cos(a) * radius)
        ' add a character
        doc.Pos.X = x
        doc.Pos.Y = y
        doc.Transform.Reset()
        Dim charRotation As Double =
If(inside, RadiansToDegrees(-a) + 180,
RadiansToDegrees(-a))
        doc.Transform.Rotate(charRotation,
x, y)
        doc.AddText(text(i).ToString())
        ' increment angle
        Dim da As Double =
Convert.ToDouble(widths(text(i))) *
fontWidthToRadians
        a += If(clockwise, da, -da)
    Next
    doc.Transform.Reset()
End Sub

```

```

Private Shared Function
DegreesToRadians(degrees As Double) As
Double
    Return degrees * Math.PI / 180
End Function

```

```
Private Shared Function  
RadiansToDegrees(radians As Double) As  
Double  
    Return radians * 180 / Math.PI  
End Function  
End Class
```



ExampleCurvedText.pdf

# CharToEncoding Property

---



Type	Default Value	Read Only	Description
[C#] IDictionary<char, char>	n/a	Yes	The Unicode to glyph mapping table for all the characters in the font
[Visual Basic] IDictionary<char, char>			

The Unicode to glyph mapping table for all the characters in the font.

In this context, the glyph value is the value which occurs in the content stream when using this font. While the encoding used is often similar to ASCII, it may vary considerably depending on the way the font has been included in the PDF.

---

## Notes

To map from the glyph encoding to a Unicode character see [EncodingToChar](#).

None.

---

## Example



# CheckGlyphs Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to exclude invalid glyphs from our lookup tables

Whether to exclude invalid glyphs from our lookup tables.

When fonts are subsetted the glyphs in the tables may be rendered invalid because they are not used. The encoding continues to exist but the information required to draw the glyphs has been lost. As such it is not a good idea to attempt to use these glyphs even though they may appear to be part of the font.

---

## Notes

By default we attempt to detect such glyphs and exclude them from any operations or font properties such as the [EncodingToChar](#) and [CharToEncoding](#) mappings. This prevents them being used inadvertently.

None.

---

## Example



# EncodingToChar Property

---



Type	Default Value	Read Only	Description
[C#] IDictionary<char, char>	0?	No?	The glyph to Unicode mapping table for all the characters in the font
[Visual Basic] IDictionary<char, char>			

The glyph to Unicode mapping table for all the characters in the font.

In this context, the glyph value is the value which occurs in the content stream when using this font. While the encoding used is often similar to ASCII it may vary considerably depending on the way the font has been included in the PDF.

---

## Notes

Typically you would use the StringAtom [Decode](#) or [DecodeDoubleByte](#) methods to allow text operator parameters to be decoded into the base text encoding. These can then be passed through the FontObject [EncodingToChar](#) and [EncodingToString](#) properties to allow mapping from the text encoding through to Unicode values.

To map from a Unicode character to the glyph encoding see [CharToEncoding](#).

---

**Example**

None.

# EncodingToString Property

---



Type	Default Value	Read Only	Description
[C#] IDictionary<char, string>	n/a	Yes	The glyph to Unicode string mapping table.
[Visual Basic] IDictionary<Char, String>			

The glyph to Unicode string mapping table for all the characters in the font that contain one.

Most glyphs map to a single Unicode character. However in some circumstances it may be necessary to map one glyph to a sequence of characters. For example the sequence "ffl" may be represented as one glyph in a font. If there is no Unicode encoding for this type of ligature then it will need to map back to a string rather than a single Unicode character.

Typically you would use the StringAtom [Decode](#) or [DecodeDoubleByte](#) methods to allow text operator parameters to be decoded into the base text encoding. These can then be passed through the FontObject [EncodingToChar](#) and EncodingToString properties to allow mapping from the text encoding through to Unicode values.

---

Notes

String encodings are unusual and this property is normally empty. If it is populated it should be viewed as a supplement and an override for the [EncodingToChar](#) and [CharToEncoding](#) mappings.

---

**Example**

None.

# Flags Property

---



Type	Default Value	Read Only	Description
[C#] FontFlags	n/a	No	The Font Descriptor entry
[Visual Basic] FontFlags			

The Font Descriptor Flags entry.

The FontFlags type is a flags type enumeration so the different values are combined together using bitwise operations. It may take the following values:

- FixedPitch (All glyphs have the same width; as opposed to proportional or variable-pitch fonts, which have different widths)
- Serif (Glyphs have serifs - short strokes drawn at an angle to the bottom of glyph stems. Sans serif fonts do not have serifs)
- Symbolic (Font contains glyphs outside the Adobe standard character set. This flag and the Nonsymbolic flag are mutually exclusive)
- Script (Glyphs look like cursive handwriting)
- Nonsymbolic (Font uses the Adobe standard Latin character set or a subset of it)
- Italic (Glyphs have slanted dominant vertical strokes)
- AllCap (Font contains no lowercase letters)
- SmallCap (Lower case letters in this font look like smaller versions of their upper case equivalents)
- ForceBold (Whether bold glyphs shall be painted with extra thick strokes, even at very small text sizes)

The flags detailed here are descriptive. So reading them from an

object will provide information on the style. However changing the style does not necessarily change it. If the font is referenced then the viewer will need to find an appropriate system font and it should try to take into account these flags. However if the font is embedded then changing the style will change the underlying embedded font. Make sure you understand these concepts before you change these values.

---

#### Notes

### How do I manipulate flags enumerations?

To check a flag you can write code of the following form:

```
bool isSymbolic =  
font.Flags.HasFlags(FontObject.FontFlags.Symbolic);
```

To set the flag:

```
font.Flags |= FontObject.FontFlags.Symbolic;
```

To clear the flag:

```
font.Flags &= ~FontObject.FontFlags.Symbolic;
```

None.

---

#### Example



# FontAscender Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			The ascender for the glyphs in this font
[Visual Basic] <code>Integer</code>		Yes	

The ascender for the glyphs in this font.

---

## Notes

This corresponds to the `sTypoAscender` entry in the 'OS/2' TrueType table.

---

## Example

None.

# FontAscent Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The ascent for the glyphs in this font

The ascent for the glyphs in this font.

---

## Notes

This corresponds to the `usWinAscent` entry in the 'OS/2' TrueType table.

---

## Example

None.

# FontBBox Property

---



Type	Default Value	Read Only	Description
[C#] XRect			
[Visual Basic] XRect	n/a	Yes	The bounding box for the glyphs in this font

The bounding box for the glyphs in this font.

This [XRect](#) corresponds to the xMin, xMax, yMin and yMax entries in the 'head' TrueType table.

---

## Notes

The PDF format only supports a limited range of metrics so FontObjects read from PDF may contain some approximations compared with 'live' FontObjects which have been added using the Doc.AddFont or Doc.EmbedFont methods.

---

## Example

None.

# FontDescender Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The descender for the glyphs in this font

The descender for the glyphs in this font.

---

## Notes

This corresponds to the `sTypoDescender` entry in the 'OS/2' TrueType table.

---

## Example

None.

# FontDescent Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The descent for the glyphs in this font

The descent for the glyphs in this font.

---

## Notes

This corresponds to the `usWinDescent` entry in the 'OS/2' TrueType table.

---

## Example

None.

# FontLineGap Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The line gap for the glyphs in this font

The line gap for the glyphs in this font.

---

## Notes

This corresponds to the `sTypoLineGap` entry in the 'OS/2' TrueType table.

---

## Example

None.

# FontLineSpacing Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The line spacing for the glyphs in this font

The line spacing for the glyphs in this font.

This broadly reflects the distance in 1000ths between the bottom of the lowest descender and the top of the highest ascender on two subsequent lines. We follow Microsoft best practice on this metric and vary the calculations depending on whether bit 7 of the `fsSelection` entry in the 'OS/2' TrueType table is set or not. However it is worth noting that many items of software including Microsoft flagship products do not.

---

## Notes

The PDF format only supports a limited range of metrics so `FontObjects` read from PDF may contain some approximations compared with 'live' `FontObjects` which have been added using the `Doc.AddFont` or `Doc.EmbedFont` methods.

None.

---

## Example





# AddResource Function

---

Add a particular type of resource

## [C#]

```
string AddResource(IndirectObject  
resource, string type, string  
name)
```

---

## Syntax

## [Visual Basic]

```
Function AddResource(resource As  
IndirectObject, type As String,  
name As String) As String
```

---

## Params

Name	Description
resource	The resource to be added.
type	The type of resource.
name	The format of the name that should be used.

Add a particular type of resource to the Form XObject.

Form XObjects may contain default resources usable by the page content stream. The most common resource types are "Font", "XObject"

and "ColorSpace". For further details see the PDF Specification.

---

**Notes**

This method allows you to add new resources to the Form XObject. You may supply your own name for the resource but if the name is already in use, it may need to be modified. For this reason the function returns the value which was actually used for the addition.

None.

---

**Example**

# GetResourceMap Function

---



Get a dictionary mapping the names of a particular type of resource to Atoms

## [C#]

```
IDictionary<string, Atom>  
GetResourceMap(string type)  
IDictionary<string, Atom>  
GetResourceMap(string type, bool  
includeXObjects, bool  
includePatterns, ISet<int> skip,  
ISet<Atom> set)
```

## [Visual Basic]

Syntax

```
Function GetResourceMap(type As  
String) As IDictionary(Of string,  
Atom)  
Function GetResourceMap(type As  
String, includeXObjects As  
Boolean, includePatterns As  
Boolean, skip As ISet{Of Integer},  
set As  
System.Collections.Generic.ISet{Of  
Atom}) As IDictionary(Of string,  
Atom)
```

Name	Description
type	The type of resource to be

	indexed.
includeXObjects	Whether to include resources from other Form XObjects referenced from this one.
includePatterns	Whether to include resources from Patterns referenced from this Form XObject.
skip	A set of IDs indicating IndirectObjects that should be skipped. This value may be null if you do not want to skip any items.
set	A set of IDs of all the resources that have been encountered. The set of resources is generally the same as the set of items in the returned dictionary, but may not be if resource names have clashed. This value may be null if you are not interested in the set.

---

Params

Get a dictionary mapping the names of a particular type of resource to Atoms. The Atoms returned are typically RefAtom objects referring to an IndirectObject. However some resource types (notably color spaces) may be referred to direct.

---

Notes

It is unusual to look up just one resource name. Typically one has a sequence of resource

names from a content stream which all need resolving. For this reason it is best to use this function to obtain a map which can then be cached.

---

**Example**

None.

# BBox Property

---



Type	Default Value	Read Only	Description
[C#] XRect			
[Visual Basic] XRect	n/a	No	The rect defining the bounding box of the graphic

The rect defining the bounding box of the graphic.

---

## Notes

Content will be clipped to this area.

---

## Example

None.

# Matrix Property

---



Type	Default Value	Read Only	Description
[C#] XTransform	n/a	No	The transformation matrix used for mapping the Form XObject space into user space
[Visual Basic] XTransform			

The transformation matrix used for mapping the Form XObject space into user space.

---

## Notes

This entry may be null which signifies the identity matrix.

---

## Example

None.



# IccProfile Constructor

---

IccProfile Constructor.

## [C#]

```
IccProfile(ObjectSoup soup, byte[] data)
IccProfile(ObjectSoup soup, string path)
```

## [Visual Basic]

### Syntax

```
Sub New(soup As ObjectSoup, data() As Byte)
Sub New(soup As ObjectSoup, path As String)
```

- may throw Exception()

### Params

Name	Description
soup	The ObjectSoup to contain the newly created IccProfile.
data	An ICC profile stored in an array of bytes.
path	A path to an ICC file.

Create an IccProfile object.

---

**Notes**

If the content provided is not a valid ICC profile then an exception will be thrown.

---

**Example**

See the [PixMap.Recolor](#) function.



## SetData Function

---

Set the raw binary content of the stream.

### [C#]

```
override void SetData(byte[]  
value)
```

### [Visual Basic]

#### Syntax

```
Overrides Sub SetData(value() As  
Byte)
```

- may throw `Exception()`

#### Params

Name	Description
value	The raw binary content of the stream.

Set the raw binary content of the stream.

This function overrides the [StreamObject.SetData](#) function.

#### Notes

If the `StreamObject` currently contains compressed data or if the content provided is not a valid ICC profile then an exception will be thrown.

---

**Example**

None.

# UpdateProfile Function

---



Updates this object to reflect the values contained within the embedded ICC color profile data

## [C#]

```
bool UpdateProfile()
```

---

## Syntax

## [Visual Basic]

```
Overridable UpdateProfile() As Boolean
```

---

## Params

Name	Description
------	-------------

return	Whether the operation was successful.
--------	---------------------------------------

Updates this object to reflect the values contained within the embedded ICC color profile data.

---

## Notes

If the ICC profile was absent or corrupt then the return value will be false.

---

Example

None.

# AlternateColorSpaceType Property

---



Type	Default Value	Read Only	Description
[C#] ColorSpaceType [Visual Basic] ColorSpaceType	n/a	Yes	The alternate color space for this ICC color profile

The alternate color space for this ICC color profile.

The alternate color space generally indicates the base color space type such as DeviceRGB or DeviceCMYK. This is useful to know if you are attempting to find out the core type of color space referenced by an ICC color profile.

---

## Notes

In the situation that no alternate color space is available, one can be generated using the [UpdateProfile](#) function.

None.

---

## Example

# ImageName Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The name used to identify the <a href="#">Pixmap</a> in the <a href="#">Page</a> resources

The name used to identify the [Pixmap](#) in the [Page](#) resources.

---

## Notes

None.

---

## Example

# PixelFormat Property

---



Type	Default	Read Only	Description
[C#] <code>PixelFormat</code>			
[Visual Basic] <code>PixelFormat</code>	n/a	Yes	The <code>PixelFormat</code> containing the image data.

The `PixelFormat` containing the image data.

---

Notes

None.

---

Example

# BaseRect Property

---



Type	Default Value	Read Only	Description
[C#] XRect	n/a	Yes	The untransformed rect defining the bounds of the visible content on the page
[Visual Basic] XRect			

The untransformed [XRect](#) defining the bounds of the visible content on the page.

---

## Notes

This rectangle is encoded in PDF coordinates rather than any abstracted coordinate space.

---

## Example

None.

# Page Property

---



Type	Default Value	Read Only	Description
[C#] Page			
[Visual Basic] Page	n/a	Yes	The Page on which the Layer is located

The [Page](#) on which the Layer is located.

---

## Notes

While Layers are normally only referenced from one page, it is possible to reference them from multiple. This property reflects the original Page for which this layer was created.

---

## Example

None.

# Rect Property

---



Type	Default	Read Only	Description
[C#] <code>XRect</code>			
[Visual Basic] <code>XRect</code>	n/a	Yes	The transformed rect defining the bounds of the visible content.

The transformed `XRect` defining the bounds of the visible content.

---

## Notes

This rectangle is encoded in PDF coordinates rather than any abstracted coordinate space.

---

## Example

None.

# Transform Property

---



Type	Default Value	Read Only	Description
[C#] XTransform	n/a	Yes	The transform which has been applied to the visible content
[Visual Basic] XTransform			

The transform which has been applied to the visible content.

---

Notes

None.

---

Example



# DeInline Function

---

Makes any inline images into external images.

## [C#]

```
void DeInline(bool  
convertAnnotations)  
void DeInline(bool  
convertAnnotations, out PixMap[]  
detachedPixMaps)
```

## [Visual Basic]

### Syntax

```
Sub DeInline(convertAnnotations  
As Boolean)  
Sub DeInline(convertAnnotations  
As Boolean, <Out> ByRef  
detachedPixMaps As PixMap())
```

- may throw Exception()

### Params

Name	Description
convertAnnotations	Whether to convert the color space of annotation appearance streams.
detachedPixMaps	All the PixMaps which were extracted as a result of the call.

Makes any inline images into external images.

Images in PDF documents are generally held as external object and are then referenced from the page content stream. However it is also possible to inline small images and embed the binary image data directly into the PDF content stream. This is not always desirable as it complicates certain types of PDF processing.

---

**Notes**

This method allows you to extract such inline images and convert them into standard external QPixmap objects.

None.

---

**Example**



# Detach Function

---

Detaches a content stream from the page.

## [C#]

```
void Detach(StreamObject layer)
```

---

### Syntax

## [Visual Basic]

```
Sub Detach(layer As StreamObject)
```

---

### Params

Name	Description
layer	The stream to detach from the page.

Detaches a content stream from the page.

---

### Notes

The layer is not deleted it is simply that any reference to it is removed from the page.

---

### Example

None.





# Flatten Function

---

Flatten and compress the page content stream.

## [C#]

```
void Flatten()  
void Flatten(bool force)
```

---

## Syntax

## [Visual Basic]

```
Sub Flatten()  
Sub Flatten(force As Boolean)
```

---

## Params

Name	Description
force	Whether to force the flattening even if the streams contain extra information which might be lost - default false.
return	Whether the page is flat.

Objects added to a page are stored as individual layers. Calling this method combines all the layers on the current page and then re-compresses the layer data.

This method is the same as the [Doc.Flatten](#) method. However the 'force' parameter can be used to force the page to be flattened even

when the `Doc.Flatten` method would be more cautious.

The `Doc.Flatten` method is cautious because different layers can be tagged with different information. Flattening the page will result in this type of information being lost. This type of tagging is not something that is defined in the PDF specification but it is possible that various third party applications may make use of proprietary techniques like this to add extra information to the document.

However for some types of operations a failure to flatten the page can be very inconvenient. For example it is possible to split PDF drawing operations over more than one content stream. If you are analyzing and updating a page using the `Doc.GetText` method to identify the locations of PDF operators, it can be very tiresome to deal with this type of construct.

None.

---

**Notes**

---

**Example**



# GetAnnotations Function

---

Gets all the Annotations on the page.

## [C#]

```
Annotation[] GetAnnotations()
```

---

## Syntax

## [Visual Basic]

```
Function GetAnnotations() As  
Annotation()
```

---

## Params

Name	Description
return	An array of all the Annotations on the page.

---

## Notes

Gets all the [Annotations](#) on the page.

---

## Example

None.



# GetLayers Function

---

Gets all the content Layers for the page.

## [C#]

```
StreamObject[] GetLayers()
```

---

### Syntax

## [Visual Basic]

```
Function GetLayers() As  
StreamObject()
```

---

### Params

Name	Description
return	An array of all the content Layers on the page.

---

### Notes

Gets all the content [Layers](#) for the page.

---

### Example

None.



# GetNamedSeparations Function

---

Gets all the named separations referenced by the page.

## [C#]

```
string[] GetNamedSeparations()
```

---

## Syntax

## [Visual Basic]

```
Function GetNamedSeparations() As String()
```

---

## Params

Name	Description
return	An array of the names of the separations.

Gets the names of all the separations referenced by the page.

In general named separations are referenced on a page because they are used on that page.

---

## Notes

However this is not a necessary truth. Separations can be referenced but never used.

---

**Example**

None.

# GetResourcesByType Function

---



Get all the resources of a named type, optionally including any used by referenced objects.

## [C#]

```
ISet<Atom>  
GetResourcesByType(string type,  
bool annotations, bool patterns,  
bool xobjects, bool parents,  
ISet<int> skip)
```

---

## Syntax

## [Visual Basic]

```
Function GetResourcesByType(type  
As String, annotations As  
Boolean, patterns As Boolean,  
xobjects As Boolean, parents As  
Boolean, skip As ISet(Of  
Integer)) As ISet(Of Atom)
```

Name	Description
type	The type of resource to be found.
annotations	Whether to include resources in annotations.
patterns	Whether to include resources in pattern dictionaries.
	Whether to include resources in

---

**Params**

xobjects	referenced Form XObjects.
parents	Whether to include resources in parent objects.
skip	A set of IndirectObject IDs that should be skipped when performing the scan. May be null.
return	A set of matching resource Atoms.

Get all the resources of a named type, optionally including any used by referenced objects.

This can, for example, be used to find all the FontObjects referenced in the "Font" section of the page. Optionally you can include annotations, patterns, Form XObjects and parent objects in the scan and generally this is something you will want to do.

---

**Notes**

If you are performing multiple operations of this type you may wish to include a set of [IndirectObject IDs](#) that should be skipped. On return the set will have been updated and all the new IndirectObjects that have been scanned will have been added.

None.

---

**Example**



# GetResourceMap Function

---



Get a dictionary mapping the names of a particular type of resource to Atoms.

## [C#]

```
IDictionary<string, Atom>  
GetResourceMap(int id, string  
type)
```

---

## Syntax

## [Visual Basic]

```
Function GetResourceMap(int id,  
string type) As IDictionary(Of  
Atom)
```

---

## Params

Name	Description
id	The type of resource to be found.
type	The ID of the resource containing object. This is typically the Page ID but you may also use a Form XObject ID.
return	A dictionary mapping resource names to Atoms.

Get a dictionary mapping the names of a particular type of resource to Atoms. The Atoms returned are typically RefAtom objects referring

to an IndirectObject. However some resource types (notably color spaces) may be referred to direct.

It is unusual to look up just one resource name. Typically one has a sequence of resource names from a content stream which all need resolving. For this reason it is best to use this function to obtain a map which can then be cached.

---

**Notes**

For resources referenced from the content stream of a page you should pass the ID of the page as this is the containing object. For resources referenced from a FormXObject stream you should pass the ID of the Form XObject as this is the containing object.

None.

---

**Example**



# GetText Function

---

Extract content from the current page in a specified format.

## [C#]

```
string GetText(TextType type, bool
includeAnnotations)
string GetText(TextType type, bool
includeAnnotations, bool includePaths, bool
includeText, bool includeImages, bool
includeColors)
```

Syntax

## [Visual Basic]

```
Function GetText(type As TextType,
includeAnnotations As Boolean) As String
Function GetText(type As TextType,
includeAnnotations As Boolean, includePaths
Boolean, includeText As Boolean, includeIma
As Boolean, includeColors As Boolean) As
String
```

Name	Description
type	The format in which to return the content.
includeAnnotations	Whether to include field and annotation text.
includePaths	Whether to include graphics paths and path operators in the output (ignored for Text and RawText).

---

**Params**

includeText	Whether to include text and text operators in the output (ignored for Text and RawText).
includeImages	Whether to include image placeholders in the output (ignored for Text and RawText).
includeColors	Whether to include colors and color operators in the output (ignored for Text and RawText).
return	An array of the names of the separation

The TextType enumeration may take the following values:

- Text
- RawText
- Svg
- SvgPlus
- SvgPlus2

Text is in layout order, which may not be the same as reading order. ABCpdf will make sensible assumptions on how items of text are combined, but some situations are ambiguous. The TextType format provides sophisticated text analysis and interpolation to produce human readable output. The TextType.RawText format is far simpler and provides a more literal interpretation of the text document.

SVG is an XML based format for representing vector graphics. Because SVG is standard XML, it's easy to parse and gives precise position of each item of text on the page. The way text is constructed in the SVG should make it easy to extract any information you require. ABCpdf currently supports SVG text, paths and image placeholders.

For example, a simple "Hello World" PDF might produce the following content:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg
<svg width="612" height="792" x="0" y="0">
<text x="0" y="76.8" font-size="96" font-
family="Times-Roman" >Hello World</text>
</svg>
```

SVG+ and SVG+2 are annotated forms of SVG which include the PDF operators and how they relate to the items of content in SVG. They can be very useful if you are trying to deconstruct and determine how objects in the PDF relate to objects in the SVG+. In SVG+, SVG elements appear before the pdf elements for their generating operators, and the pdf elements for the Do operator on Form XObjects are not generated. In SVG+2, SVG elements appear after the pdf elements of their generating operators, and the pdf elements for the Do operator on Form XObjects are generated.

For example, you could use SVG+ to identify the section of a stream that relates to a particular word on a page. You could replace the text show operator for that word with another operator. Effectively, you'd be performing a low-level Search/Replace on the document. However, you should note that this would not mean your layout would automatically adjust if - for example - you replace a short word with a long one.

There is no official standard for SVG+, but if you are familiar with the PDF specification, it should be easy enough to understand.

---

**Notes**

For example, a simple "Hello World" PDF might produce the following content:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg
<svg width="612" height="792" x="0" y="0">
<pdf pdf_op="q" pdf_StreamID="5"
pdf_StreamOffset="0" pdf_StreamLength="1" /
```

```

<pdf pdf_0p="BT" pdf_StreamID="5"
pdf_StreamOffset="3" pdf_StreamLength="2" /
<pdf pdf_0p="0 Tr" pdf_StreamID="5"
pdf_StreamOffset="7" pdf_StreamLength="4" /
<pdf pdf_0p="/Fabc6 96 Tf" pdf_StreamID="5"
pdf_StreamOffset="13" pdf_StreamLength="12"
<pdf pdf_0p="0 0 0 rg" pdf_StreamID="5"
pdf_StreamOffset="27" pdf_StreamLength="8"
<pdf pdf_0p="1 0 0 1 0 715.2 Tm" pdf_Stream
pdf_StreamOffset="37" pdf_StreamLength="18"
<pdf pdf_0p="0 Ts" pdf_StreamID="5"
pdf_StreamOffset="57" pdf_StreamLength="4"
<text x="0" y="76.8" font-size="96" font-
family="Times-Roman" pdf_CTM="1 0 0 1 0 0"
pdf_TM="1 0 0 1 0 715.2" pdf_Trm="96 0 0 96
715.2" pdf_Tf="Fabc6" pdf_Tz="100" pdf_Ts="
pdf_w1000="5027" pdf_0p="(Hello World) Tj"
pdf_StreamID="5" pdf_StreamOffset="63"
pdf_StreamLength="16" >Hello World</text>
<pdf />
<pdf pdf_0p="ET" pdf_StreamID="5"
pdf_StreamOffset="81" pdf_StreamLength="2"
<pdf pdf_0p="Q" pdf_StreamID="5"
pdf_StreamOffset="85" pdf_StreamLength="1"
</svg>

```

The operators within the PDF stream are detailed in the SV example, the first 'q' operator is located in Object ID 5 at offset 63 and has a length of 1 byte. The 'Tj' operator which shows "Hello World" is located at offset 63 and has length 16. The Current Transformation Matrix (CTM), the Text Matrix (TM), and other important PDF state values are also detailed in the example.

Unfortunately, the XML specification was designed in such a way that it does not allow all ASCII values to be represented. There are certain ranges of characters that are completely banned, and this is true even if you attempt to use entity references to include them. Given that the PDF specification allows a broader range of values, you need to be careful when using XML to represent PDF content.

consider how you represent characters outside the XML rar pdf\_Op attribute, all such characters are moved into the U+ Unicode private use area. So to convert this string to a PDF string, you just need to coerce any such character to a byte only relevant to the pdf\_Op attribute.

---

**Example**

None.

# Recolor Function

---



Converts the page from one color space to another.

## [C#]

```
void Recolor(ColorSpace space,
bool convertAnnotations)
void Recolor(ColorSpace space,
bool convertAnnotations, out
Pixmap[] recoloredPixMaps)
```

## [Visual Basic]

Syntax

```
Sub Recolor(space As ColorSpace,
convertAnnotations As Boolean)
Sub Recolor(space As ColorSpace,
convertAnnotations As Boolean,
<Out> ByRef recoloredPixMaps As
Pixmap())
```

- may throw Exception()

Name	Description
space	The destination color space.
convertAnnotations	Whether to convert the color space of annotation appearance streams.

Params

recoloredPixMaps

All the PixMaps which were recolored as a result of the call.

Converts the page from one color space to another.

All images used on the page are converted to the new color space. All color operators used in the page content stream are converted to the new color space.

Annotations and fields are not part of the page but instead float over the page. You can choose whether to convert the appearance stream of any annotations or leave them in their native color space.

Colors can only be sensibly mapped from one color space to another if we know something about the characteristics of the color space. If your color spaces do not contain this information (e.g. if they are device color spaces) then ABCpdf will use a default color profile.

---

Notes

An exception will be thrown if the operation is not possible. This may happen if the Page is not in an ObjectSoup or if the ColorSpace object is in some way invalid.

As part of the Recolor process it is necessary to call [Pixmap.Recolor](#) on all the images used on the page. After the Recolor process has been completed these [Pixmap](#) objects will no longer be compressed. You may wish to compress them using the [StreamObject.Compress](#)

method.

None.

---

**Example**

# VectorizeText Function

---



Replaces the text on the page with glyph outlines.

## [C#]

```
void VectorizeText()
```

## [Visual Basic]

### Syntax

```
Sub VectorizeText()
```

- may throw `Exception()`

Name	Description
none	

### Params

Use this method to vectorize the text (i.e. replace the text with equivalent glyph polygon outlines).

### Notes

## [C#]

```
Doc theDoc = new Doc();  
theDoc.FontSize = 96;  
theDoc.AddText("Hello World");
```

```

List<Page> theList = new List<Page>();
foreach (IndirectObject obj in theDoc.ObjectSou
{
    Page page = obj as Page;
    if (page != null) {
        theList.Add(page);
    }
}
for (int i = 0; i < theList.Count; i++) {
    Page page = theList[i];
    page.VectorizeText();
}
theDoc.Save(Server.MapPath("VectorizedText.pdf")
theDoc.Clear();

```

**Example**

### **[Visual Basic]**

```

Dim theDoc As Doc = New Doc()
theDoc.FontSize = 96
theDoc.AddText("Hello World")
Dim obj As IndirectObject
Dim thePage As Page
Dim theList As New List(Of Page)
For Each obj in theDoc.ObjectSoup
    thePage = TryCast(obj, Page)
    If thePage IsNot Nothing Then
        theList.Add(thePage)
    End If
Next
For Each thePage As Page in theList
    thePage.VectorizeText()
Next
theDoc.Save(Server.MapPath("VectorizedText.pdf")
theDoc.Clear()

```





# AddLayer Function

---

Add a content layer at the front of the page

## [C#]

```
void AddLayer(StreamObject layer)
```

---

### Syntax

## [Visual Basic]

```
Sub AddLayer(layer As  
StreamObject)
```

---

### Params

Name	Description
layer	The layer to be added.

Add a content layer at the front of the page.

The [StreamObject](#) that is supplied should contain PDF drawing operations. You will need to ensure that any resources that are referenced in this content stream have been added to the resources of the page using a method such as [AddResource](#).

---

### Notes

---

**Example**

None.



# AddResource Function

---

Add a particular type of resource to the page

## [C#]

```
string AddResource(IndirectObject  
resource, string type, string  
name)
```

---

## Syntax

## [Visual Basic]

```
Function AddResource(resource As  
IndirectObject, type As String,  
name As String) As String
```

---

## Params

Name	Description
resource	The resource to be added.
type	The type of resource.
name	The format of the name that should be used.

Add a particular type of resource to the page.

Pages may contain default resources usable by the page content stream. The most common resource types are "Font", "XObject" and "ColorSpace". For further details see the PDF

Specification.

---

**Notes**

This method allows you to add new resources to the page. You may supply your own name for the resource but if the name is already in use, it may need to be modified. For this reason the function returns the value which was actually used for the addition.

---

**Example**

None.



# GetBitmap Function

---

Render one or more layers on the current page

## [C#]

```
System.Drawing.Bitmap  
GetBitmap(Layer[] layers)
```

## [Visual Basic]

Syntax

```
Function GetBitmap(layers() As  
Layer) As System.Drawing.Bitmap
```

- may throw Exception()

Params

Name	Description
layers	The layers to be rendered.
return	The System.Drawing.Bitmap containing the image of the layers.

Render one or more layers on the current page.

This function renders a set of layers and returns the result as a System.Drawing.Bitmap. The Bitmap covers the smallest possible area which encompasses all the layers provided.

If this Page is not in a Doc then an exception will be raised. The function will return null if the layers argument is null or empty or the output Bitmap would be smaller than one pixel in area.

Rendering options such as resolution are taken directly from the current [Doc.Rendering](#) settings.

---

**Notes**

However automatic page rotation (XRendering.AutoRotate) is disabled so that a Layer can be rendered and then re-inserted into the page as a raster copy of itself.

Note that a layer can exist on multiple pages and indeed can be rendered in the context of a page that does not currently contain it. However layers typically contain references to named resources which are only available in the context of a specific page. So rendering a layer in the context of a page which does not normally contain it is prone to error unless the page and layer have been specifically constructed to allow this.

The following example shows how to use this method to generate various types of drop shadows.

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        // light blue background
        doc.Color.SetCmyk(50, 0, 0, 0);
        doc.FillRect();
        doc.Color.SetRgb(0, 0, 0);
        doc.Rect.Inset(20, 20);
        doc.Rect.Pin = XRect.Corner.TopLeft;
        doc.Rect.Height = doc.Rect.Height /
```

```

5;
    // set up styles
    doc.TextStyle.Size = 72;
    double shift = doc.TextStyle.Size *
0.1;
    XColor pink = new XColor();
    pink.SetRgb(255, 128, 128);
    XColor gray = new XColor();
    gray.SetRgb(128, 128, 128);
    XColor blue = new XColor();
    blue.SetRgb(0, 0, 255);
    // add text content
    AddDropShadow(doc, doc.AddText("Sharp
Shadow"), 0, shift, -shift, gray);
    doc.Rect.Move(0, -doc.Rect.Height);
    AddDropShadow(doc,
doc.AddText("Blurred Shadow"), 1, shift,
-shift, gray);
    doc.Rect.Move(0, -doc.Rect.Height);
    AddDropShadow(doc, doc.AddText("Pink
Shadow"), 1, shift, -shift, pink);
    doc.Rect.Move(0, -doc.Rect.Height);
    // add drawn content
    doc.Transform.Magnify(0.5, 0.5, 0,
0);
    doc.Transform.Translate(50, 0);
    string star = "124 158 300 700 476
158 15 493 585 493 124 158";
    doc.Width = 20;
    doc.Color.String = "255 0 0";
    AddDropShadow(doc, doc.AddPoly(star,
false), 3, shift, -shift, blue);
    doc.Save("dropshadows.pdf");
}
}

```

```

void AddDropShadow(Doc doc, int id,

```

```

double gaussianBlurRadius, double
shadowHorizontalShift, double
shadowVerticalShift, XColor shadowColor)
{
    string rect = doc.Rect.String;
    string transform =
doc.Transform.String;
    string color = doc.Color.String;
    double dpiX =
doc.Rendering.DotsPerInchX;
    double dpiY =
doc.Rendering.DotsPerInchY;
    bool saveAlpha =
doc.Rendering.SaveAlpha;
    int docLayer = doc.Layer;
    try {
        doc.Rendering.DotsPerInch = 72;
        doc.Rendering.SaveAlpha = true;
        Layer layer = doc.ObjectSoup[id] as
Layer;
        Page page = doc.ObjectSoup[doc.Page]
as Page;
        Bitmap bm = page.GetBitmap(new
Layer[] { layer });
        doc.Transform.Reset();
        doc.Rect.String = layer.Rect.String;
        doc.Rect.Move(shadowHorizontalShift,
shadowVerticalShift);
        doc.Layer = docLayer + 1;
        int pid = doc.AddImageBitmap(bm,
true);
        // Here we set the base image to be
one pixel of an appropriate color.
        // This is what will determine the
shadow color.
        ImageLayer img = doc.ObjectSoup[pid]
as ImageLayer;

```

```

        PixMap pm = img.PixMap;
        pm.ClearData(); // this will remove
any compression settings
        pm.SetData(new byte[] {
(byte)shadowColor.Red,
(byte)shadowColor.Green,
(byte)shadowColor.Blue });
        pm.Width = 1;
        pm.Height = 1;
        // The alpha channel is held as a
separate soft mask and this is what will
        // determine the shape of the shadow.
If required we blur it to give it
        // soft edges.
        PixMap alpha = pm.SMask;
        if (gaussianBlurRadius > 0) {
            using (EffectOperation effect = new
EffectOperation("Gaussian Blur")) {
                effect.Parameters["Radius"].Value
= gaussianBlurRadius;
                effect.Apply(alpha);
            }
        }
    }
finally {
    doc.Rect.String = rect;
    doc.Transform.String = transform;
    doc.Color.String = color;
    doc.Rendering.DotsPerInchX = dpiX;
    doc.Rendering.DotsPerInchY = dpiY;
    doc.Rendering.SaveAlpha = saveAlpha;
    doc.Layer = docLayer;
}
}
}

```

**[Visual Basic]**

---

Example

```
Sub ...
  Using doc As New Doc()
    ' light blue background
    doc.Color.SetCmyk(50, 0, 0, 0)
    doc.FillRect()
    doc.Color.SetRgb(0, 0, 0)
    doc.Rect.Inset(20, 20)
    doc.Rect.Pin = XRect.Corner.TopLeft
    doc.Rect.Height = doc.Rect.Height / 5
    ' set up styles
    doc.TextStyle.Size = 72
    Dim shift As Double =
doc.TextStyle.Size * 0.1
    Dim pink As New XColor()
    pink.SetRgb(255, 128, 128)
    Dim gray As New XColor()
    gray.SetRgb(128, 128, 128)
    Dim blue As New XColor()
    blue.SetRgb(0, 0, 255)
    ' add text content
    AddDropShadow(doc, doc.AddText("Sharp
Shadow"), 0, shift, -shift, gray)
    doc.Rect.Move(0, -doc.Rect.Height)
    AddDropShadow(doc,
doc.AddText("Blurred Shadow"), 1, shift,
-shift, gray)
    doc.Rect.Move(0, -doc.Rect.Height)
    AddDropShadow(doc, doc.AddText("Pink
Shadow"), 1, shift, -shift, pink)
    doc.Rect.Move(0, -doc.Rect.Height)
    ' add drawn content
    doc.Transform.Magnify(0.5, 0.5, 0, 0)
    doc.Transform.Translate(50, 0)
    Dim star As String = "124 158 300 700
476 158 15 493 585 493 124 158"
    doc.Width = 20
    doc.Color.[String] = "255 0 0"
```

```
    AddDropShadow(doc, doc.AddPoly(star,
False), 3, shift, -shift, blue)
    doc.Save("dropshadows.pdf")
End Using
End Sub
```

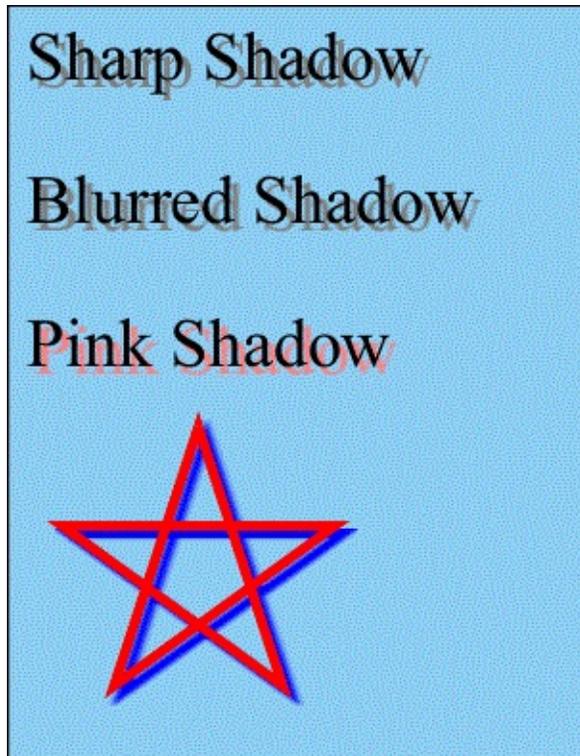
```
Private Sub AddDropShadow(doc As Doc, id
As Integer, gaussianBlurRadius As Double,
shadowHorizontalShift As Double,
shadowVerticalShift As Double,
shadowColor As XColor)
    Dim rect As String = doc.Rect.[String]
    Dim transform As String =
doc.Transform.[String]
    Dim color As String = doc.Color.
[String]
    Dim dpiX As Double =
doc.Rendering.DotsPerInchX
    Dim dpiY As Double =
doc.Rendering.DotsPerInchY
    Dim saveAlpha As Boolean =
doc.Rendering.SaveAlpha
    Dim docLayer As Integer = doc.Layer
Try
    doc.Rendering.DotsPerInch = 72
    doc.Rendering.SaveAlpha = True
    Dim layer As Layer =
TryCast(doc.ObjectSoup(id), Layer)
    Dim page As Page =
TryCast(doc.ObjectSoup(doc.Page), Page)
    Dim bm As Bitmap = page.GetBitmap(New
Layer() {layer})
    doc.Transform.Reset()
    doc.Rect.[String] = layer.Rect.
[String]
    doc.Rect.Move(shadowHorizontalShift,
shadowVerticalShift)
```

```

        doc.Layer = docLayer + 1
        Dim pid As Integer =
doc.AddImageBitmap(bm, True)
        ' Here we set the base image to be
one pixel of an appropriate color.
        ' This is what will determine the
shadow color.
        Dim img As ImageLayer =
TryCast(doc.ObjectSoup(pid), ImageLayer)
        Dim pm As PixMap = img.PixMap
        pm.ClearData()
        ' this will remove any compression
settings
        pm.SetData(New Byte()
{CByte(shadowColor.Red),
CByte(shadowColor.Green),
CByte(shadowColor.Blue)})
        pm.Width = 1
        pm.Height = 1
        ' The alpha channel is held as a
separate soft mask and this is what will
        ' determine the shape of the shadow.
If required we blur it to give it
        ' soft edges.
        Dim alpha As PixMap = pm.SMask
        If gaussianBlurRadius > 0 Then
            Using effect As New
EffectOperation("Gaussian Blur")
                effect.Parameters("Radius").Value
= gaussianBlurRadius
                effect.Apply(alpha)
            End Using
        End If
    Finally
        doc.Rect.[String] = rect
        doc.Transform.[String] = transform
        doc.Color.[String] = color

```

```
doc.Rendering.DotsPerInchX = dpiX  
doc.Rendering.DotsPerInchY = dpiY  
doc.Rendering.SaveAlpha = saveAlpha  
doc.Layer = docLayer  
End Try  
End Sub
```



dropshadows.pdf



# MakeFormXObject Function

---

Makes a FormXObject out of the page.

## [C#]

```
FormXObject MakeFormXObject()
```

---

## Syntax

## [Visual Basic]

```
Function MakeFormXObject() As  
FormXObject
```

---

## Params

Name	Description
return	The newly created FormXObject.

Makes a [FormXObject](#) out of the page.

This process involves copying the page content stream to a new Form XObject and referencing any required resources. The current page is left unaltered.

---

## Notes

The returned FormXObject will have been added to the same ObjectSoup as contains the Page. The returned Form XObject can then be added to other pages using methods such as [AddXObject](#).

This example shows how to take a page, convert it into a separate drawing object and then draw it, scaled, onto the page it came from.

### [C#]

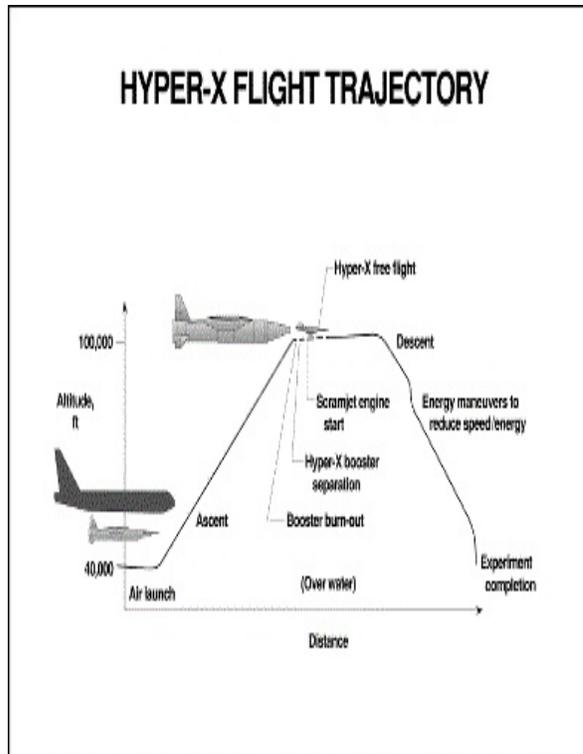
```
using (Doc doc = new Doc()) {
    doc.Read("HyperX.pdf");
    Page page1 =
doc.ObjectSoup[doc.Page] as Page;
    FormXObject form =
page1.MakeFormXObject();
    doc.Transform.Magnify(0.5, 0.5, 0,
0);
    doc.Page = doc.AddPage();
    Page page2 =
doc.ObjectSoup[doc.Page] as Page;
    string name =
page2.AddResource(form, "XObject",
"Iabc");
    // Here we create our own layer for
the purposes of the demonstration.
    // However a simpler approach would
be to use Doc.AddXObject.
    StreamObject layer = new
StreamObject(doc.ObjectSoup);
    layer.SetText(String.Format("q {0}
cm /{1} Do Q ",
doc.Transform.ToString(), name));
    page2.AddLayer(layer);
    doc.Save("exampleformxobject.pdf");
}
```

## [Visual Basic]

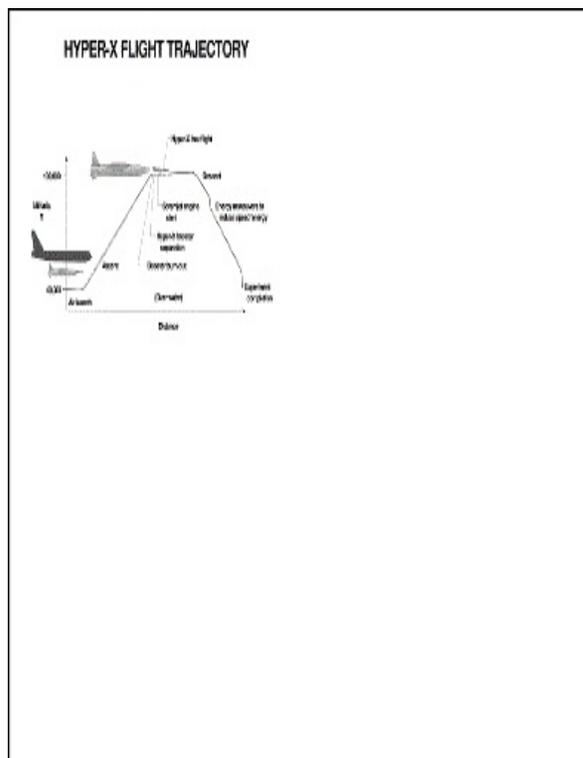
```
Sub ...
    Using doc As New Doc()
        doc.Read("HyperX.pdf")
        Dim page1 As Page =
TryCast(doc.ObjectSoup(doc.Page),
Page)
        Dim form As FormXObject =
page1.MakeFormXObject()
        doc.Transform.Magnify(0.5, 0.5, 0,
0)
        doc.Page = doc.AddPage()
        Dim page2 As Page =
TryCast(doc.ObjectSoup(doc.Page),
Page)
        Dim name As String =
page2.AddResource(form, "XObject",
"Iabc")
        ' Here we create our own layer for
the purposes of the demonstration.
        ' However a simpler approach would
be to use Doc.AddXObject.
        Dim layer As New
StreamObject(doc.ObjectSoup)
        layer.SetText([String].Format("q
{0} cm /{1} Do Q ",
doc.Transform.ToString(), name))
        page2.AddLayer(layer)
        doc.Save("exampleformxobject.pdf")
    End Using
End Sub
```

---

Example



exampleformxobject.pdf [Page 1]



exampleformxobject.pdf [Page 2]





# StampFormXObjects Function

---

Removes all Form XObjects from the page by embedding them into the page content

## [C#]

```
int StampFormXObjects(bool force)
```

---

## Syntax

## [Visual Basic]

```
Function StampFormXObjects(force  
As Boolean) As Integer
```

---

## Params

Name	Description
force	Whether to stamp FormXObjects that may cause a change in the page appearance.
return	The number of FormXObjects removed from the page.

Removes all [Form XObjects](#) from the page by embedding them into the content.

Most Form XObjects can be embedded in the page with no difference in page appearance. However Form XObjects such as transparency groups contain structures which cannot be

represented any other way. These structures are uncommon in most documents but they do occur.

---

**Notes**

In this situation you need to decide whether a change in page appearance is acceptable and that the objects should be forcibly embedded. The alternative is to skip these objects and embed only those that will not affect the page appearance.

None.

---

**Example**

# ArtBox Property

---



Type	Default Value	Read Only	Description
[C#] XRect			
[Visual Basic] XRect	n/a	No	The ArtBox for the page

The ArtBox defines the extent of meaningful content on this page.

These boundaries are always defined directly on the Page object. If no property has been defined then this property will be null and the effective value is assumed to be that of the CropBox.

---

## Notes

Note that, as with all objects in this namespace, this property is measured in points in the native PDF coordinate space. It is unaffected by the [Doc.Units](#) or [Doc.TopDown](#) properties.

None.

---

## Example

# BleedBox Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">XRect</a>			
[Visual Basic] <a href="#">XRect</a>	n/a	No	The BleedBox for the page

The BleedBox defines the region to which the contents of the page shall be clipped when rendered in a production environment.

These boundaries are always defined directly on the Page object. If no property has been defined then this property will be null and the effective value is assumed to be that of the CropBox.

---

## Notes

Note that, as with all objects in this namespace, this property is measured in points in the native PDF coordinate space. It is unaffected by the [Doc.Units](#) or [Doc.TopDown](#) properties.

None.

---

## Example



# CropBox Property

---



Type	Default Value	Read Only	Description
[C#] XRect	n/a	No	The CropBox for the page
[Visual Basic] XRect			

- may throw `NullReferenceException()`

The CropBox for the page.

The CropBox defines the visible part of the physical medium on which the page may be displayed or printed. This is what you see as the boundary of the page when you view a document using Acrobat Reader.

These boundaries may be defined directly on the Page object or they may be inherited from a parent Pages object.

The CropBox is optional and if this property is null it is implicitly assumed to be the same as the MediaBox.

Assigning a new value to this property will change the property for the current Page rather than any Pages object from which the value may have been inherited. In this way the property exhibits a copy-on-

write behavior.

Attempting to assign a null value to this property will result in a `NullReferenceException` being thrown. This is because the `CropBox` may be inherited from a parent page and thus removing the `CropBox` from the current page may simply result in it being inherited from another. This is unusual and counterintuitive behavior and can result in subtle bugs related to specific documents. As such, if you want to clear this property you should assign it the value of the `MediaBox`.

Note that, as with all objects in this namespace, this property is measured in points in the native PDF coordinate space. It is unaffected by the `Doc.Units` or `Doc.TopDown` properties.

---

**Example**

None.

# MediaBox Property

---



Type	Default Value	Read Only	Description
[C#] XRect			
[Visual Basic] XRect	n/a	No	The MediaBox for the page

- may throw `NullReferenceException()`

The MediaBox defines the boundaries of the physical medium on which the page may be displayed or printed. This may include bleed areas or printing marks and also parts of the medium on which printing cannot take place because of physical limitations of the output technology. Any PDF drawing outside the MediaBox can be safely ignored.

These boundaries may be defined directly on the Page object or they may be inherited from a parent Pages object.

A MediaBox is mandatory for all pages so this property should never be null.

---

## Notes

Assigning a new value to this property will change the property for the current Page rather than any Pages object from which the value may have been inherited. In this way the property exhibits a copy-on-write behavior.

Attempting to assign a null value to this property will result in a `NullReferenceException` being thrown.

Note that, as with all objects in this namespace, this property is measured in points in the native PDF coordinate space. It is unaffected by the `Doc.Units` or `Doc.TopDown` properties.

---

**Example**

None.

# PageNumber Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The number of the page in the document.

This property holds the current Page Number.

---

## Notes

The PageNumber indicates the page using an index ranging between one and the [Doc.PageCount](#).

---

## Example

None.

# Parent Property

---



Type	Default	Read Only	Description
[C#] Pages			
[Visual Basic] Pages	n/a	Yes	The parent of this page.

The parent [Pages](#) object for this page.

---

Notes

None.

---

Example

# Rotation Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	No	The number of degrees to rotate the page before display

The number of degrees clockwise to rotate the page before display

The values 0 and 180 indicate portrait orientation. The values 90 and 270 indicate landscape orientation.

---

## Notes

This value should be a multiple of 90.

This example shows how to use the Rotation property to determine if a PDF page - which may be rotated - is portrait or landscape.

### [C#]

```
using (Doc doc = new Doc(), src = new Doc()) {
    src.Read("landscape.pdf");
    int rotation =
        ((Page)src.ObjectSoup[src.Page]).Rotation;
    bool landscape = src.MediaBox.Width >
        src.MediaBox.Height;
    doc.Page = doc.AddPage(); // output is always
    portrait
}
```

```

if(landscape){
    switch(rotation) {
    default:
    case 90:
        doc.Transform.Rotate(270, 0, 0);
        doc.Transform.Translate(0,
doc.MediaBox.Height);
        break;
    case 180:
    case 270:
        doc.Transform.Rotate(90, 0, 0);
        doc.Transform.Translate(doc.MediaBox.Width
        break;
    }
    doc.Rect.SetRect(0, 0, doc.MediaBox.Height,
doc.MediaBox.Width);
    } else {
    switch(rotation) {
    case 90:
    case 180:
        doc.Transform.Rotate(180, 0, 0);
        doc.Transform.Translate(doc.MediaBox.Width
doc.MediaBox.Height);
        break;
    }
    }
    doc.AddImageDoc(src, 1, null);
    doc.Save("addtoportrait.pdf");
}

```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc(), src As New Doc()
        src.Read("landscape.pdf")
        Dim rotation As Integer =
((Page)src.ObjectSoup(src.Page)).Rotation

```

---

Example

```
' output is always in portrait
If landscape Then
  Select Case rotate
    Case Else, 90
      doc.Transform.Rotate(270, 0, 0)
      doc.Transform.Translate(0,
doc.MediaBox.Height)
    Exit Select
    Case 180, 270
      doc.Transform.Rotate(90, 0, 0)
      doc.Transform.Translate(doc.MediaBox.
0)

    Exit Select
  End Select
  doc.Rect.SetRect(0, 0, doc.MediaBox.Heigh
doc.MediaBox.Width)
Else
  Select Case rotate
    Case 90, 180
      doc.Transform.Rotate(180, 0, 0)
      doc.Transform.Translate(doc.MediaBox.
doc.MediaBox.Height)
    Exit Select
  End Select
End If
doc.AddImageDoc(src, 1, Nothing)
doc.Save("addtoportrait.pdf")
End Using
End Sub
```

# Landscape Orientation

landscape.pdf

Landscape  
Orientation

addtoportrait.pdf

# Thumbnail Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">Pixmap</a>			The Thumbnail for the page
[Visual Basic] <a href="#">Pixmap</a>	n/a	No	

Each page can have a thumbnail image for quick preview purposes. This property allows to access such a thumbnail if it exists, or to assign such a thumbnail to a page if it does not.

---

## Notes

This example shows how to create and embed thumbnails in a PDF document.

### [C#]

```
using (Doc doc = new Doc(), srcDoc = new Doc())
{
    doc.Read("spaceshuttle.pdf");
    doc.Rendering.DotsPerInch = 18;
    Page[] pages =
doc.ObjectSoup.Catalog.Pages.GetPageArrayAll();
    foreach (Page page in pages) {
        doc.Page = page.ID;
        using (XImage xi =
```

```
XImage.FromData(doc.Rendering.GetData(".jpg"),
null))
    page.Thumbnail =
Pixmap.FromXImage(doc.ObjectSoup, xi);
}
doc.Save("embedthumbnails.pdf");
}
```

---

Example

### [Visual Basic]

```
Sub ...
    Using doc As New Doc(), srcDoc As New Doc()
        doc.Read("spaceshuttle.pdf")
        doc.Rendering.DotsPerInch = 18
        Dim pages As Page() =
doc.ObjectSoup.Catalog.Pages.GetPageArrayAll()
        For Each page As Page In pages
            doc.Page = page.ID
            Using xi As XImage =
XImage.FromData(doc.Rendering.GetData(".jpg"),
Nothing)
                page.Thumbnail =
Pixmap.FromXImage(doc.ObjectSoup, xi)
            End Using
        Next
        doc.Save("embedthumbnails.pdf")
    End Using
End Sub
```

# TrimBox Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">XRect</a>			
[Visual Basic] <a href="#">XRect</a>	n/a	No	The TrimBox for the page

The TrimBox defines the intended boundaries of the page after trimming.

These boundaries are always defined directly on the Page object. If no property has been defined then this property will be null and the effective value is assumed to be that of the CropBox.

---

## Notes

Note that, as with all objects in this namespace, this property is measured in points in the native PDF coordinate space. It is unaffected by the [Doc.Units](#) or [Doc.TopDown](#) properties.

None.

---

## Example



## GetPage Function

---

Performs a fast lookup to retrieve a particular Page from this node tree

### [C#]

```
Page GetPage(int page)
```

---

### Syntax

### [Visual Basic]

```
Function GetPage(Integer page) As Page
```

---

### Params

Name	Description
page	The page number - a one based index
returns	The specified Page object or null if one could not be found.

Performs a fast lookup to retrieve a particular Page from this node tree.

The page is specified in terms of the index of the page within the tree. This is a one-based index within the current Pages node. Since a Pages node relates to only a portion of a PDF document, the page number within the Pages node is offset from the page number within the

---

### Notes

document.

Example

None.

# GetPageArray Function

---



Gets all the Page objects immediately under this node.

## [C#]

```
Page[] GetPageArray()
```

---

### Syntax

## [Visual Basic]

```
Function GetPageArray() As Page()
```

---

### Params

Name	Description
return	An array of Page objects.

---

### Notes

Gets all the [Page](#) objects under this node.

Only immediate children are returned. Any Pages objects are returned as null Page objects.

For an array that includes descendents see [GetPageArrayAll](#).

---

**Example**

None.



## GetPageArrayAll Function

---

Gets all the Page objects under this node and descendents of this node.

### [C#]

```
Page[] GetPageArrayAll()
```

---

### Syntax

### [Visual Basic]

```
Function GetPageArrayAll() As  
Page()
```

---

### Params

Name	Description
return	An array of Page objects.

---

### Notes

Gets all the [Page](#) objects under this node and descendents of this node. The pages are returned in order of page number. The [Page.ID](#) can be assigned directly to the [Doc.Page](#) property to set the focus to a particular page. For large or badly structured documents it is likely to be faster to use this type of operation than it is to repeatedly access the [Doc.PageNumber](#) property.

For an array of the immediate children of this

node see [GetPageArray](#).

---

**Example**

None.

# Recolor Function

---



Converts the pages from one color space to another.

## [C#]

```
void Recolor(ColorSpace space,
bool convertAnnotations)
void Recolor(ColorSpace space,
bool convertAnnotations, out
Pixmap[] recoloredPixMaps)
```

## [Visual Basic]

Syntax

```
Sub Recolor(space As ColorSpace,
convertAnnotations As Boolean)
Sub Recolor(space As ColorSpace,
convertAnnotations As Boolean,
<Out> ByRef recoloredPixMaps As
Pixmap())
```

- may throw Exception()

Name	Description
space	The destination color space.
convertAnnotations	Whether to convert the color space of annotation appearance streams.

Params

recoloredPixMaps	All the PixMaps which were recolored as a result of the call.
------------------	---

Converts the pages from one color space to another.

---

**Notes**

This method calls the [Page.Recolor](#) method on all Page objects within this group of pages.

None.

---

**Example**

# Count Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The number of visible pages under this node.

The number of visible pages under this node.

---

## Notes

None.

---

## Example

# Parent Property

---



Type	Default	Read Only	Description
[C#] Pages			
[Visual Basic] Pages	n/a	Yes	The parent of this node.

The parent [Pages](#) object for this node.

---

Notes

None.

---

Example



# FromXImage Function

---

Pixmap static constructor

## [C#]

```
static Pixmap  
FromXImage(ObjectSoup soup,  
XImage image)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromXImage(soup  
As ObjectSoup, image As XImage)  
As Pixmap
```

---

## Params

Name	Description
soup	The ObjectSoup to contain the newly created Pixmap.
image	The XImage from which the Pixmap should be created.

This method allows you to create a Pixmap directly from an XImage object.

The Pixmap that is created exists within the current Doc.Soup but is not linked into any pages.

---

## Notes

To link it into a page t needs to be added as a resource and then drawn from the content stream of the page.

---

**Example**

None.



## CompressCcitt Function

---

Compresses the image using CCITT compression.

### [C#]

```
void CompressCcitt()
```

### [Visual Basic]

```
Sub CompressCcitt()
```

- may throw `Exception()`

---

#### Syntax

---

#### Params

Name	Description
none	

Compresses the image using CCITT compression.

CCITT compression can only be used on black and white images. It is optimized for the lossless compression of scanned documents and faxes.

---

#### Notes

If the values of both the `BitsPerComponent` and the `Components` is one then the `PixMap` can be

compressed using this method. If not then calling this method will generate an exception.

---

**Example**

None.



## CompressJbig2 Function

---

Compresses the image using JBIG2 compression.

### [C#]

```
void CompressJbig2()
```

### [Visual Basic]

---

#### Syntax

```
Sub CompressJbig2()
```

- may throw `Exception()`

---

#### Params

Name	Description
none	

Compresses the image using JBIG2 compression.

---

#### Notes

JBIG2 compression can only be used on monochrome - typically black and white - images. It offers high compression levels for images like scanned documents which contain repeating patterns.

---

**Example**

None.



# CompressJpeg Function

---

Compresses the image using JPEG compression.

## [C#]

```
void CompressJpeg(int quality)
```

## [Visual Basic]

```
Sub CompressJpeg(quality As Integer)
```

- may throw Exception()

Syntax

Params

Name	Description
quality	The quality of compression to use (0 to 100).

Compresses the image using JPEG compression.

JPEG compression can be used on RGB, Grayscale or CMYK images. It offers high compression levels for continuous tone images like photographs. However it is a lossy method which means that the quality of the compressed

---

**Notes**

image will not be as good as the uncompressed image.

The quality of the compression can range from zero (lowest quality, highest compression) to 100 (highest quality, lowest compression). A good generic value is 75.

---

**Example**

See the [Recolor](#) function.



## CompressJpx Function

---

Compresses the image using JPEG 2000 compression.

### [C#]

```
void CompressJpx(int quality)
```

### [Visual Basic]

```
Sub CompressJpx(quality As Integer)
```

- may throw Exception()

Syntax

Params

Name	Description
quality	The quality of compression to use (0 to 100).

Compresses the image using JPEG 2000 compression.

JPEG 2000 compression can be used on RGB, Grayscale or CMYK images. It offers high compression levels for continuous tone images like photographs. However it is a lossy method which means that the quality of the compressed

---

**Notes**

image will not be as good as the uncompressed image.

The quality of the compression can range from zero (lowest quality, highest compression) to 100 (lossless quality, lowest compression). A good generic value is 75.

---

**Example**

None.

# Decompress Function

---



Decompress the data in the stream using on-the-fly resizing

## [C#]

```
bool Decompress(int width, int height)
```

---

## Syntax

## [Visual Basic]

```
Function Decompress(width As Integer, height As Integer) As Boolean
```

---

## Params

Name	Description
width	The desired final width.
height	The desired final height
return	Whether the data was decompressed correctly.

Decompress the data in the stream using on-the-fly resizing of the image. This type of scaling is fast and memory efficient but may change the depth of the image. Only some compression types support on-the-fly resizing.

---

**Notes**

As such you should always check the dimensions of the image after calling this method. Do not assume that simply because the image has decompressed correctly that it has also been resized.

---

**Example**

None.



# Flip Function

---

Flip the image horizontally or vertically

## [C#]

```
void Flip(bool horizontal, bool vertical)
```

---

## Syntax

## [Visual Basic]

```
Sub Flip(horizontal As Boolean, vertical As Boolean)
```

---

## Params

Name	Description
horizontal	Whether to flip the image horizontally.
vertical	Whether to flip the image vertically

This function flips the image horizontally or vertically or both. Flipping both horizontally and vertically is equivalent to a 180 degree rotation.

An image may have an associated bit mask or soft (alpha) mask. If this is the case the associated mask will be transformed as well.

Flipping horizontally will mean that pixels that

were on the far right of the image will be moved to the far left and pixels on the left will be moved to the right.

---

**Notes**

Flipping horizontally will mean that pixels that were on the top of the image will be moved to the bottom and pixels that were on the bottom will be moved to the top.

After the Bitmap has been set the PixMap will be uncompressed. You may wish to compress it using a call like [CompressJpeg](#) or [CompressFlate](#).

None.

---

**Example**



# GetBitmap Function

---

Get the PixMap image as a System.Drawing.Bitmap.

## [C#]

```
System.Drawing.Bitmap GetBitmap()
```

## [Visual Basic]

```
Function GetBitmap() As  
System.Drawing.Bitmap
```

Syntax

- may throw Exception()

Name	Description
return	The Bitmap containing the image.

Params

Use this method to get the PixMap image as a System.Drawing.Bitmap.

You can then use this Bitmap for drawing to screen or for manipulation using System.Drawing routines.

ABCpdf tries to make a literal copy of the image contained in the PixMap, both to minimize color shifts and also for speed.

However because PDF images can contain many color spaces and bit depths that are unsupported by System.Drawing it may be necessary to change the color space or bit depth of the image.

---

#### Notes

In addition there are certain parameters such as decode arrays which may be used to modify the image before display. This method returns the image content as literally as possible which means decode arrays and alpha channels are not included.

Please note that some formats of Bitmaps are not usable for certain operations. In particular, indexed color bitmaps cannot be drawn on using the Graphics.FromImage method. For this reason if you are going to wish to draw on the returned Bitmap you should check the pixel format before use.

This example shows how to extract all the page thumbnails they exist) from a document. See also the ContentExtract example project for another example.

#### [C#]

```
using (Doc doc = new Doc(), srcDoc = new Doc()
{
    doc.Read("embedthumbnails.pdf");
    doc.Rendering.DotsPerInch = 18;
    Page[] pages =
doc.ObjectSoup.Catalog.Pages.GetPageArrayAl
    foreach (Page page in pages) {
        if (page.Thumbnail == null)
            continue;
        string path = "embedthumbnails" +
page.Thumbnail.ID.ToString( + ".jpg");
        using (Bitmap bm =
```

```

page.Thumbnail.GetBitmap() {
    bm.Save(path);
}
}
}
}

```

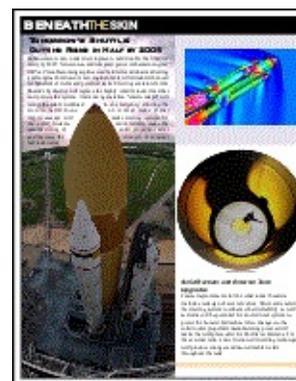
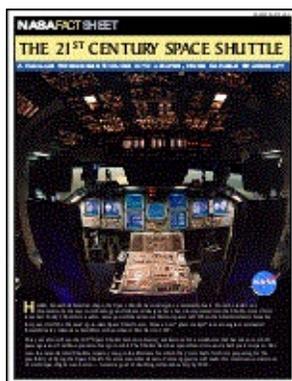
## [Visual Basic]

```

Sub ...
    Using doc As New Doc(), srcDoc As New Doc
        doc.Read("embedthumbnails.pdf")
        doc.Rendering.DotsPerInch = 18
        Dim pages As Page() =
doc.ObjectSoup.Catalog.Pages.GetPageArrayAll
        For Each page As Page In pages
            If page.Thumbnail Is Nothing Then
                Continue For
            End If
            Dim path As String = "embedthumbnails
page.Thumbnail.ID.ToString( + ".jpg")
            Using bm As Bitmap =
page.Thumbnail.GetBitmap()
                bm.Save(path)
            End Using
        Next
    End Using
End Sub

```

Example



embedthumbnails.pdf - [Page 1] embedthumbnails.pdf - [P



embedthumbnails.pdf - [Page 3] embedthumbnails.pdf - [P



# Realize Function

---

Converts the image to component color.

## [C#]

```
void Realize()
```

## [Visual Basic]

```
Sub Realize()
```

- may throw Exception()

Syntax

Params

Name	Description
none	

Converts the image from indexed color to component color.

The process of converting an indexed color image into a component color image, will result in any chromakeys being converted to masks and the elimination of any decode arrays.

The Indexed color space is used for palletized color. Each item in the palette is defined in terms of a base color space such as

Notes

DeviceRGB. Palettes can hold up to 256 entries.

After an indexed color image has been realized it is no longer compressed. You may wish to compress it using the [StreamObject.Compress](#) method.

---

**Example**

See the [Resize](#) function.



# Recolor Function

---

Converts the image from one color space to another.

## [C#]

```
void Recolor(ColorSpace space)
void Recolor(ColorSpace space, RenderingInt
intent)
```

## [Visual Basic]

### Syntax

```
Sub Recolor(space As ColorSpace)
Sub Recolor(space As ColorSpace, intent As
RenderingIntent)
```

- may throw Exception()

### Params

Name	Description
space	The destination color space.
intent	The rendering intent enumeration. If no intent is provided, the default for the image (typically RelativeColorimetric)

Converts the image from one color space to another.

Calling this method results in the pixels within the PixMap image being mapped from the current color space to the new color space.

The new `ColorSpace` is then assigned to the `Pixmap` and (if already there) added to the `Pixmap ObjectSoup`.

Colors can only be sensibly mapped from one color space to another if we know something about the characteristics of the color spaces. If your color spaces do not contain this information (e.g. if the device color spaces) then ABCpdf will use a default color profile.

An exception will be thrown if the operation is not possible. This can happen if the `Pixmap` is not in an `ObjectSoup` or if the `ColorSpace` object is in some way invalid.

If the `ImageMask` property is true the image has no color space and is implicitly black and white. Image masks cannot be anything other than black and white so trying to convert an image mask to another color space will result in an exception being raised.

The rendering intent determines how out of gamut colors are handled. The following options are available:

---

Notes

- Perceptual
- RelativeColorimetric
- Saturation
- AbsoluteColorimetric

The perceptual model maps the entire gamut of the source color space into the destination one and is good for photographic images. The saturation model is good for diagrams, cartoon, posterized images where distinctiveness of color is more important than precise color fidelity. The colorimetric methods remap colors which are out of gamut. The relative one keeps the color hue the same while allowing the brightness to vary. The absolute colorimetric method uses the closest color at the gamut boundary.

In general you will want to use a perceptual intent when mapping a large gamut color space (e.g. RGB) to a narrow one (e.g. CMYK). In general you will want to use a relative colorimetric intent when mapping between similar color spaces (e.g. RGB to RGB or CMYK).

After the PixMap has been Recolored it is no longer compressed and will have a [BitsPerComponent](#) of eight or sixteen. You may compress it using the [StreamObject.Compress](#) method.

Here we change all the images in a document to CMYK.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../Rez/spaceshu
List<PixMap> theList = new List<PixMap>();
// find all the PixMap objects in the soup
foreach (IndirectObject obj in theDoc.Objec
    PixMap p = obj as PixMap;
    if (p != null)
        theList.Add(p);
}
// add our destination color space
ColorSpace cs = new ColorSpace(theDoc.Objec
cs.IccProfile = new IccProfile(theDoc.Objec
Server.MapPath("../Rez/abccmyk.icc"));
// convert images to our color space
for (int i = 0; i < theList.Count; i++) {
    PixMap p = theList[i];
    p.Recolor(cs, RenderingIntent.Perceptual)
    p.CompressJpeg(75);
}
theDoc.Save(Server.MapPath("pixmaprecolor.p
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As New Doc()
theDoc.Read(Server.MapPath("../Rez/spaceshu
Dim theList As New List(Of PixMap)
```

```

' find all the PixMap objects in the soup
Dim obj As IndirectObject
Dim p As PixMap
For Each obj In theDoc.ObjectSoup
    p = TryCast(obj, PixMap)
    If p IsNot Nothing Then
        theList.Add(p)
    End If
Next ' add our destination color space
Dim cs As New ColorSpace(theDoc.ObjectSoup)
cs.IccProfile = New IccProfile(theDoc.Object
Server.MapPath("../Rez/abccmyk.icc"))
' convert images to our color space
For i As Integer = 0 To theList.Count - 1
    p = theList(i)
    p.Recolor(cs, RenderingIntent.Perceptual)
    p.CompressJpeg(75)
Next
theDoc.Save(Server.MapPath("pixmaprecolor.p
theDoc.Clear()

```

---

### Example



spaceshuttle.pdf [page 1]



pixmaprecolor.pdf [page 1]



# Resample Function

---



Changes the number of bits per color component.

## [C#]

```
void Resample(int  
bitsPerComponent)
```

## [Visual Basic]

Syntax

```
Sub Resample(bitsPerComponent As  
Integer)
```

- may throw Exception()

Params

Name	Description
bitsPerComponent	The number of bits per color component.

Change the number of bits per color component.

The bit depths you should use are 1, 2, 4, 8 or 16. Other values are not supported in the PDF Specification.

Changing this property for Indexed color images changes the precision of the index rather than of the color components. If you want to change the precision of the components you need to [Realize](#) the QPixmap first.

---

**Notes**

Operations like [Resize](#) operate more naturally on eight and sixteen bit images - producing better results, faster, using less memory. So if you will be resizing and also moving to eight or sixteen bit depth it is typically better to Resample before calling Resize.

After this method is called the image may no longer be longer compressed. You may wish to compress it using the [StreamObject.Compress](#) method.

None.

---

**Example**



# Resize Function

---

Resizes the image.

## [C#]

```
void Resize(int width, int height)
void Resize(int width, int height, Interpol
interpolation)
```

## [Visual Basic]

### Syntax

```
Sub Resize(width As Integer, height As Inte
Sub Resize(width As Integer, height As Inte
interpolation As Interpolation)
```

- may throw Exception()

### Params

Name	Description
width	The destination width for the image.
height	The destination height for the image.
interpolation	The type of interpolation to be used when res image. The default value is Auto.

Resizes the image to a specified width and height using a s interpolation method.

The interpolation parameter should be viewed as a request than a command because not all types of interpolation can all types of image. If the method you specify is not appropriate ABCpdf will automatically select a suitable alternative.

After an image has been resized it is no longer compressed. If you wish to compress it using the [StreamObject.Compress](#) method.

The Interpolation enumeration may take the following values:

- Auto
- NearestNeighbor
- Linear
- Cubic
- Super
- Lanczos

The Auto setting allows ABCpdf to automatically choose an appropriate interpolation algorithm given the type of image and type of scaling. If you are aiming to maximize quality this is the one you should use.

---

Notes

The nearest neighbor algorithm is the fastest but also the lowest quality method. It simply finds the nearest pixel in the source image and maps it through to the destination image. However it is the only method which can be used for Indexed color images.

The Linear, Cubic and Lanczos methods are progressively higher quality methods based around a weighted average of pixels in the source image. However they are also progressively slower. The Lanczos function used is the three lobed variety.

The Linear, Cubic and Lanczos methods use weighted averages of a limited number of pixels in the source image. For large size reductions this may result in information from some pixels in the source image being completely discarded. The super method works around this issue by increasing coverage to all the pixels in the source image. It only works if an image is being reduced in height and width.

This function is optimized for resizing one, three and four color images at precisions of eight or sixteen bits per component

Here we resize all the images in a document to a quarter of resolution.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../Rez/spaceship.jpg"));
foreach (IndirectObject io in theDoc.ObjectSoup)
    if (io is PixMap) {
        PixMap pm = (PixMap)io;
        pm.Realize(); // eliminate indexed color
        pm.Resize(pm.Width / 4, pm.Height / 4);
    }
}
theDoc.Save(Server.MapPath("pixmapresize.pcx"));
theDoc.Clear();
```

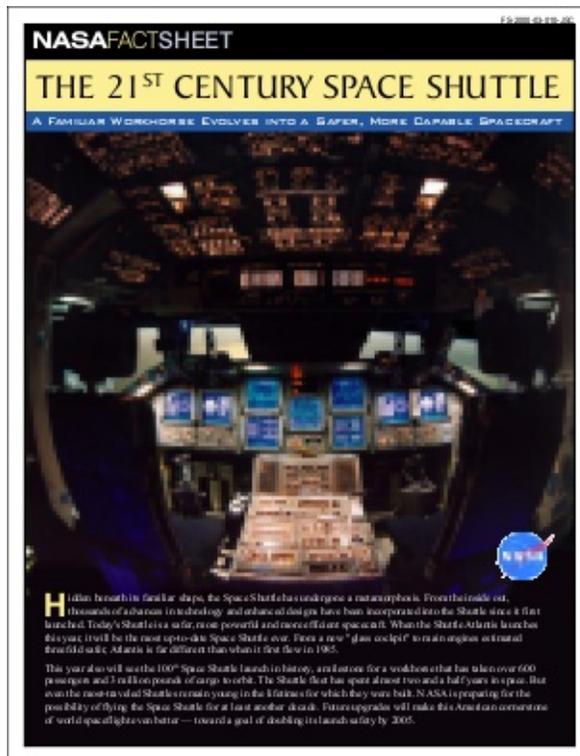
### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../Rez/spaceship.jpg"))
Dim io As IndirectObject
For Each io in theDoc.ObjectSoup
    If TypeOf io Is PixMap Then
        Dim pm As PixMap = CType(io, PixMap)
        pm.Realize() ' eliminate indexed color
        pm.Resize(pm.Width / 4, pm.Height / 4)
    End If
Next
theDoc.Save(Server.MapPath("pixmapresize.pcx"))
theDoc.Clear()
```

Example



spaceshuttle.pdf [page 1]



pixmapresize.pdf [page 1]





# Rotate Function

---

Rotate the image clockwise

## [C#]

```
void Rotate(int degrees)
```

## [Visual Basic]

```
Sub Rotate(degrees As Integer)
```

- may throw `ArgumentOutOfRangeException()`

Syntax

Params

Name	Description
degrees	The number of degrees clockwise that the image should be rotated. This must be a multiple of 90.

This function rotates the `PixelFormat` image clockwise. The rotation angle must be a multiple of 90. If an invalid number of degrees is specified, then an `ArgumentOutOfRangeException` will be thrown.

An image may have an associated bit mask or soft (alpha) mask. If this is the case, the

associated mask will be transformed as well.

---

**Notes**

When a PixMap is drawn on a page, it is scaled to fit into a particular location. The scaling is separate from the PixMap itself. Rotating by 90 or 270 degrees will swap the width and height of the PixMap. Because the scaling of drawn copies is separate from the PixMap itself, any previously drawn instances of the PixMap may appear to have an incorrect aspect ratio.

After the Bitmap has been set, the PixMap will be uncompressed. You may wish to compress it using a call like [CompressJpeg](#) or [CompressFlate](#).

---

**Example**

None.

# Save Function

---



Saves the PixMap to stream attempting to preserve resolution, color space and depth as far as the output format allows

## [C#]

```
void Save(string path)
void Save(Stream stream, string
name)
```

---

## Syntax

## [Visual Basic]

```
Sub Save(path As String)
Sub Save(stream As Stream, name
As String)
```

---

## Params

Name	Description
path	The destination file path.
stream	The destination output stream.
name	A dummy file name used to determine the type of image required.

Saves the PixMap to stream attempting to preserve resolution, color space and depth as far as the output format allows.

---

**Notes**

The output format is specified using a file extension. In the case of saving to file, the extension is taken from the file path. In the case of saving to stream, a dummy name should be provided.

For details of the capabilities of output formats, see the [XRendering.Save](#) method.

---

**Example**

None.



# SetAlpha Function

---

Sets a constant alpha value (0-255) for this image.

## [C#]

```
void SetAlpha(double alpha)
```

---

### Syntax

## [Visual Basic]

```
Sub SetAlpha(alpha As Double)
```

Name	Description
alpha	A constant alpha value to assign to this image.

---

### Params

Assigns a constant alpha transparency to the PixMap.

---

### Notes

The alpha value should range from 0 (fully transparent) to 255 (fully opaque). Any values outside this range will result in the alpha channel being removed.

Here we add an image without transparency and then, at a distance down and to the right, with 50% transparency.

## [C#]

```

Doc theDoc = new Doc();
theDoc.Rect.Pin = XRect.Corner.TopLeft;
theDoc.Rect.Magnify(0.5, 0.5);
string thePath =
Server.MapPath("../mypics/mypic.tif");
theDoc.AddImageFile(thePath, 1);
theDoc.Rect.Move(theDoc.Rect.Width, -
theDoc.Rect.Height);
int i = theDoc.AddImageFile(thePath, 1);
ImageLayer im = (ImageLayer)theDoc.ObjectSc
im.PixMap.SetAlpha(128);
theDoc.Save(Server.MapPath("pixmapsetalpha.
theDoc.Clear());

```

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
theDoc.Rect.Pin = XRect.Corner.TopLeft
theDoc.Rect.Magnify(0.5, 0.5)
Dim thePath As String =
Server.MapPath("../mypics/mypic.tif")
theDoc.AddImageFile(thePath, 1)
theDoc.Rect.Move(theDoc.Rect.Width, -
theDoc.Rect.Height)
Dim i As Integer = theDoc.AddImageFile(theF
Dim im As ImageLayer = CType(theDoc.ObjectS
ImageLayer)
im.PixMap.SetAlpha(128)
theDoc.Save(Server.MapPath("pixmapsetalpha.
theDoc.Clear()

```

---

Example



pixmapsetalpha.pdf



# SetBitmap Function

---

Set the content of the object as a Bitmap

## [C#]

```
void  
SetBitmap(System.Drawing.Bitmap  
bitmap, bool transparent)
```

---

## Syntax

## [Visual Basic]

```
Sub SetBitmap(bitmap As  
System.Drawing.Bitmap,  
transparent As Boolean)
```

---

## Params

Name	Description
bitmap	The Bitmap containing the image.
transparent	Whether any transparency information should be preserved.

Set the content of the object as a System.Drawing.Bitmap.

If transparency is required, the PixMap must be contained within an [ObjectSoup](#).

---

## Notes

After the Bitmap has been set, the PixMap will

be uncompressed. You may wish to compress it using a call like [CompressJpeg](#) or [CompressFlate](#).

---

**Example** See the [Doc.AddXObject](#) method..



## SetChromakey Function

---

Sets a chromakey transparent color for this image.

### [C#]

```
void SetChromakey(string chromakey)
```

---

### Syntax

### [Visual Basic]

```
Sub SetChromakey(chromakey As String)
```

---

### Params

Name	Description
chromakey	A chromakey string to assign to this image.

This allows you to assign a transparent color or color range image.

You need to specify two values for each component of the color. For a particular pixel, if all the components of the color fall within the specified ranges, then the pixel will not be displayed.

For example, to make pure white elements of an RGB transparent image, you might specify,

---

### Notes

```
doc.SetChromakey("255 255 255 255 255 255")
```

If you wanted to include colors which were off-white you might specify,

```
doc.SetChromakey("250 255 250 255 250 255")
```

This function is equivalent to setting the PixMap "/Mask" en array of color values.

Here we add an image over the top of a green background. chromakey to make black and near-black colors transparen

### [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Inset(20, 20);
theDoc.Color.String = "0 255 0";
theDoc.FillRect();
string thePath =
Server.MapPath("../mypics/mypic.tif");
int i = theDoc.AddImageFile(thePath, 1);
ImageLayer im = (ImageLayer)theDoc.ObjectSc
im.PixMap.SetChromakey("0 50 0 50 0 50");
theDoc.Save(Server.MapPath("pixmapsetchroma
theDoc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Inset(20, 20)
theDoc.Color.String = "0 255 0"
theDoc.FillRect()
Dim thePath As String =
Server.MapPath("../mypics/mypic.tif")
Dim i As Integer = theDoc.AddImageFile(theF
Dim im As ImageLayer = CType(theDoc.ObjectS
ImageLayer)
im.PixMap.SetChromakey("0 50 0 50 0 50")
```

---

Example

```
theDoc.Save(Server.MapPath("pixmapsetchroma  
theDoc.Clear()
```



pixmapsetchromakey.pdf



# ToGrayscale Function

---

Converts the image to grayscale.

## [C#]

```
void ToGrayscale()
```

---

### Syntax

## [Visual Basic]

```
Sub ToGrayscale()
```

---

### Params

Name	Description
none	

This allows you to convert an image to grayscale. It can be used for preparing soft masks.

---

### Notes

This function is a convenience method for this common operation. A practically identical effect can be achieved using the [Rectangle](#) method followed by [Compress](#).

Here we add an image in its natural color space and then, as shown in the image below, converted to grayscale.

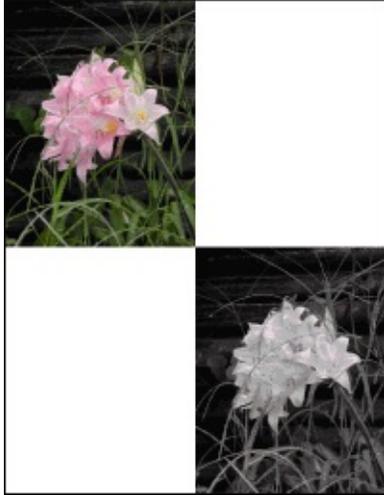
## [C#]

```
Doc theDoc = new Doc();
theDoc.Rect.Pin = XRect.Corner.TopLeft;
theDoc.Rect.Magnify(0.5, 0.5);
string thePath =
Server.MapPath("../mypics/mypic.tif");
theDoc.AddImageFile(thePath, 1);
theDoc.Rect.Move(theDoc.Rect.Width, -
theDoc.Rect.Height);
int i = theDoc.AddImageFile(thePath, 1);
ImageLayer im = (ImageLayer)theDoc.ObjectSc
im.PixMap.ToGrayscale();
theDoc.Save(Server.MapPath("pixmaptograyscale
theDoc.Clear();
```

## [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Rect.Pin = XRect.Corner.TopLeft
theDoc.Rect.Magnify(0.5, 0.5)
Dim thePath As String =
Server.MapPath("../mypics/mypic.tif")
theDoc.AddImageFile(thePath, 1)
theDoc.Rect.Move(theDoc.Rect.Width, -
theDoc.Rect.Height)
Dim i As Integer = theDoc.AddImageFile(theF
Dim im As ImageLayer = CType(theDoc.ObjectS
ImageLayer)
im.PixMap.ToGrayscale()
theDoc.Save(Server.MapPath("pixmaptograyscale
theDoc.Clear()
```

**Example**



pixmaptograyscale.pdf

# AutoFix Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to automatically fix corrupt images.

Some corrupt documents can contain corrupt images. The most common type of corruption is the truncation of the image data. However most of the time it is better to use the image data that exists rather than throw an error.

If the AutoFix property is set, then when the image is decompressed it will be checked. If it is found that the data has been truncated then extra blank data will be added onto the end.

---

## Notes

Typically such an image will appear containing black content at the bottom. Depending on the amount of truncation in the original data there will be varying amounts of blank space.

None.

---

## Example



# BitsPerComponent Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The number of bits per color component.

The number of bits per color component.

This property can take the value 1, 2, 4, 8 or 16. Other values are not supported in the PDF Specification.

---

## Notes

To change the bits per component of an image use the [Resample](#) method.

None.

---

## Example

# ColorSpace Property

---



Type	Default	Read Only	Description
[C#] ColorSpace	n/a	No	The ColorSpace for this image.
[Visual Basic] ColorSpace			

The [ColorSpace](#) for this image.

Note that referencing this property may result in the creation of a color space if one does not already exist. So you should avoid querying this property while iterating through the [ObjectSoup](#). If you do so and an object is created then it will invalidate the enumerator and cause an exception to be raised.

Not all PixMaps have color spaces. For example image masks do not and must not contain a color space atom; they are implicitly one bit grayscale. In this case the ColorSpace property will be null.

In most cases properties which you might like to reference via the ColorSpace can be referenced directly via the PixMap. This is a less intrusive way of obtaining the same information.

---

## Notes

Querying the value of this property will never raise an exception.

Assigning a value to this property changes the color space without changing the raw pixel values of the image. You might wish to do this if you wanted to convert - say - a DeviceRGB to a CalRGB color space or a DeviceGray to a Separation color space.

The number of components in the two color spaces should be the same. If they are not then an exception will be thrown.

If you wish to convert the image pixel values from one color space to another then you need to use the [Recolor](#) method.

---

**Example**

None.

# ColorSpaceType Property

---



Type	Default	Read Only	Description
[C#] ColorSpaceType	n/a	Yes	The ColorSpace for this image.
[Visual Basic] ColorSpaceType			

The [ColorSpaceType](#) for this image.

Referencing this property has a number of advantages over referencing the `PixelFormat.ColorSpace.ColorSpaceType` property.

Firstly it avoids the possibility that a new `ColorSpace` object will need to be created.

---

## Notes

Secondly it takes account of the fact that certain types of `PixelFormat` (e.g. `ImageMasks`) do not have a `ColorSpace`. In these cases the `ColorSpaceType` is implicit in the definition of the `PixelFormat` and needs to be inferred by the `PixelFormat` itself.

None.

---

## Example



# Components Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of color components for each pixel.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The number of color components for each pixel.

For example Grayscale images contain one component, RGB images contain three and CMYK images contain four.

Referencing this property has a number of advantages over referencing the [PixelFormat.ColorSpace.Components](#) property.

---

## Notes

Firstly it avoids the possibility that a new `ColorSpace` object will need to be created.

Secondly it takes account of the fact that certain types of `PixelFormat` (e.g. `ImageMasks`) do not have a `ColorSpace`. In these cases the number of components is implicit in the definition of the `PixelFormat` and needs to be inferred by the `PixelFormat` itself.

None.

---

**Example**

# Height Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The height of the image in pixels.

The height of the image in pixels.

---

Notes

None.

---

Example

# ImageMask Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether this image is a one bit image mask.

Whether this image is a one bit image mask.

---

Notes

None.

---

Example

# Mask Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">PixelFormat</a>			Any one bit image mask associated with this image
[Visual Basic] <a href="#">PixelFormat</a>	n/a	No	

One-bit masks do not have the quality of soft masks but they are more compatible with older viewing software, printer RIPs and formats like PDF/A.

---

## Notes

If both a [SMask](#) and Mask entry are specified, then the [SMask](#) will take priority.

---

## Example

None.

# Matte Property

---



Type	Default Value	Read Only	Description
[C#] <code>ArrayAtom</code>			Any matte associated with this soft mask
[Visual Basic] <code>ArrayAtom</code>	n/a	No	

The matte is array of components specifying the color with which the image has been preblended. The number of items in the array is equal to the number of components.

---

## Notes

None.

---

## Example

# SMask Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">PixelFormat</a>			Any soft image mask associated with this image
[Visual Basic] <a href="#">PixelFormat</a>	n/a	No	

Soft masks are higher quality than one bit masks but are less compatible with older viewing software, printer RIPs and formats like PDF/A.

---

## Notes

If both a SMask and [Mask](#) entry are specified, then the SMask will take priority.

None.

---

## Example

# Width Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The width of the image in pixels.

The width of the image in pixels.

---

Notes

None.

---

Example



## Commit Function

---

Commit a previously signed signature to the document.

### [C#]

```
void Commit()
```

### [Visual Basic]

```
Sub Commit()
```

Syntax

- may throw `Exception()`

Params

Name	Description
none	

After a signature is signed, it needs to be committed to the document.

Normally, this is done when you save the document. It happens invisibly without you needing to do anything. However, sometimes, you may wish to commit before the document is saved. This most commonly occurs if you are working with documents containing more than one signature field.

The PDF architecture requires that a document be incrementally updated each time a signature is signed. It requires this so that a PDF viewer can show what changes were made to the document between the times it was signed. For this reason, if you are signing multiple fields, each signature (bar the last) needs to be signed and then committed in turn. The last signature does not need to be committed because this is implicitly done by the final save.

---

Notes

A Commit operation involves an implicit [Doc.Save](#) event. For this reason it is important that the [Doc.SaveOptions](#) are appropriately set up before the signature is Committed. If you intending to commit to a document containing multiple signatures you will want to ensure that the [XSaveOptions.Incremental](#) property is set.

Similarly after each commit, all previous references to form fields are invalidated. You need to obtain updated references to form fields from the [Doc.Form.Fields](#) property.

None.

---

Example

# GetCertificates Function

---



Extract the encoded X.509 data of the certificate(s).

## [C#]

```
IEnumerable<byte[]>  
GetCertificates()  
IEnumerable<byte[]>  
GetCertificates(out int  
outCount)
```

## [Visual Basic]

### Syntax

```
Function GetCertificates()  
As IEnumerable(Of Byte())  
Function  
GetCertificates(<Out> ByRef  
outCount As Integer) As  
IEnumerable(Of Byte())
```

- may throw Exception()

### Params

Name	Description
outCount	The number of certificates.
return	The encoded X.509 data for the certificate(s).

Use this method to extract the encoded X.509 data of the certificate that was used to sign the document. Normally, there is only one certificate returned, but for some documents, you may receive additional certificates that can be used to create a certificate chain. In such cases, the first certificate is always the certificate that was used for signing.

---

**Notes**

You can pass the data returned by this function to the `X509Certificate2` class constructor in `System.Security.Cryptography.X509Certificates` and then extract certificate details such as the subject, the issuer, the serial number, the version, etc. See the Annotations example project for details.

---

**Example**

None.



# Sign Method

---

Sign the digital signature using a private key and password

## [C#]

```
void Sign(string path, string password)
void Sign(byte[] data, string password)
void Sign(Stream stream, string password)
void Sign(X509Certificate2 cert, bool siler
```

## [Visual Basic]

Syntax

```
Sub Sign(path As String, password As String)
Sub Sign(data() As Byte, password As String)
Sub Sign(stream As Stream, password As Stri
Sub Sign(cert As X509Certificate2, silent A
Boolean)
```

- may throw Exception()

Name	Description
path	The path to the PFX/PKCS #12 (.pfx or .p12) file signing.
data	The data for the PFX/PKCS #12 (.pfx or .p12) file signing.
stream	The stream for the PFX/PKCS #12 (.pfx or .p12) signing.

	password The password used to authorize use of the private key.
<b>Params</b>	cert The System.Security.Cryptography.X509Certificates.X509Certificate2 object for the PFX/PKCS #12 (.pfx or .p12) certificate used for signing.
	silent Whether to suppress prompting the user to use the private key. While using the private keys of some certificates imported with the check box "Enable strong private key protection" checked, you will be prompted every time the private key is used by an application if you enable this option. If this option is true, no dialog box is displayed and must be closed if this option is false, and the signing fails if this parameter is true in an interactive/unattended operation, this parameter must be true.

Use this method to sign a signature field.

In order to sign a signature, you need to use your private key. To authorize the use of this key, you need to provide your password. If you are not using a X509Certificate2 certificate.

Typically, this password protected private key is held in an PFX/PKCS #12 (.pfx or .p12) file. So to perform the signing operation, you need a path to this file and a password to allow use of the private key.

**Notes** Time-stamped signatures can be produced by using the set of Time Stamping Authority (TSA). See [TimestampServiceUrl](#).

If you are signing multiple signature fields in the same PDF document, you should call [Commit](#) manually after each signing operation.

If the file is not available, if the file is invalid or if the password is incorrect, then this function will throw an exception.

Read a document and sign a signature field embedded with it. Before signing, we specify a location and a reason why the document was digitally signed.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../Rez/Authorization.pdf"));
Signature theSig = (Signature)theDoc.Form["Signature"];
theSig.Location = "Washington";
theSig.Reason = "Schedule Agreed";
theSig.Sign(Server.MapPath("../Rez/JohnSmith1234"));
theDoc.Save(Server.MapPath("Signed.pdf"));
```

---

Example

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../Rez/Authorization.pdf"))
Dim theSig As Signature = theDoc.Form("Signature")
theSig.Location = "Washington"
theSig.Reason = "Schedule Agreed"
theSig.Sign(Server.MapPath("../Rez/JohnSmith1234"))
theDoc.Save(Server.MapPath("Signed.pdf"))
```

# Validate Function

---



Check and validate the status of this signature.

## [C#]

```
bool Validate()  
bool Validate(string[] certificatePaths)  
bool Validate(System.Collections.IEnumerable  
certificates)
```

## [Visual Basic]

---

### Syntax

```
Function Validate() As Boolean  
Function Validate(certificatePaths() As String)  
Boolean  
Function Validate(certificates As  
System.Collections.IEnumerable) As Boolean
```

- may throw `Exception()`

---

### Params

Name	Description
certificatePaths	An array of paths to X.509 certificate (.cer) files
certificates	A collection of <code>System.Security.Cryptography.X509Certificates</code>
return	True if the certificate is valid.

This function returns true if the signature is valid. To be precise, t

updates the properties [IsModified](#), [IsTimeValid](#) and [IsTrusted](#). The value is true only if the [IsModified](#) is false, and all of [IsTimeValid](#), and [IsTrusted](#) are true.

Signatures' certificates can only be validated by referencing certificates issued by certification authorities.

This method allows you to check and validate the status of a signature reference to a set of such certificates. Additionally, ABCpdf can also validate certificates found in the Windows Certificate Store for validation. See [ValidationPolicy](#) for details.

The certificates you provide will be cached at a document level. This function is efficient even when checking multiple signatures within a document. If you do not provide any parameters, this function will use previously cached certificates to validate the document. Therefore, if [ValidationPolicy](#) is set to `EntireChainTrust`, or certificates have been provided using a previous call to this function, calling the parameterized version of this function will cause an exception to be thrown if there are no certificates to validate against.

ABCpdf does not currently do revocation checks on certificates provided and on certificates embedded in a PDF document. If you need to perform this type of operation, you should use the [GetCertificates](#) function and validate certificates yourself.

If a certificate is unavailable or invalid, this method may throw an `InvalidCertificateException`. This means validating against an unsigned signature field will cause an exception to be thrown.

---

## Notes

### **How does Adobe Reader validate a PDF document with certificate files?**

You may find that Adobe Reader does not need a list of certificates to validate PDF documents. This is because Adobe Reader includes several built-in Public Key Infrastructure hierarchies to certify documents:

- **Certified Document Services (CDS)** is a trust hierarchy that chains back to the Adobe Root Certification Authority (Adobe Root CA).
- **Adobe Approved Trust List (AATL)** is an extra list of CA certificates that Adobe Reader may download from Adobe periodically (for Adobe Reader/Acrobat 9 or later).
- The Windows Certificate Store. This is only true if Windows signature integration is enabled in Acrobat, which is not default for Acrobat 9 and X.

In order to validate a PDF document the same way Adobe Reader does, you need to use the same certificates it uses. This can be achieved by exporting the *trusted identities* from Adobe Reader to PFX format certificate files. (Note: CDS and AATL certificates are *not* in your Windows Certificate Store by default.)

The Windows Certificate Store can be accessed by using `System.Security.Cryptography.X509Certificates.X509Store` (see below).

## [C#]

```
// Validate using certificate files
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../Rez/Signed.pdf"));
string[] theCerts =
Server.MapPath("../Rez/JohnSmith.cer").Split(new
char[] { ';' });
Signature theSig =
(Signature)theDoc.Form["Signature"];
if ((theSig.Validate(theCerts)) &&
(!theSig.IsModified)) {
    theDoc.AddText("Signature valid at " +
DateTime.Now.ToString());
}
```

```
}  
theDoc.Save(Server.MapPath("SignedAndValidated.
```

```
// Validate using the Windows Certificate Store  
Doc theDoc = new Doc();  
theDoc.Read(Server.MapPath("../Rez/Signed.pdf"));  
X509Store theStore = new X509Store(StoreName.Root  
StoreLocation.LocalMachine);  
theStore.Open(OpenFlags.ReadOnly);  
Signature theSig =  
(Signature)theDoc.Form["Signature"];  
if ((theSig.Validate(theStore.Certificates)) &&  
(!theSig.IsModified)) {  
    theDoc.AddText("Signature valid at " +  
DateTime.Now.ToString());  
}  
theStore.Close();  
theDoc.Save(Server.MapPath("SignedAndValidated.
```

---

Example

### [Visual Basic]

```
' Validate using certificate files  
Dim theDoc As Doc = New Doc()  
theDoc.Read(Server.MapPath("../Rez/Signed.pdf"))  
Dim theCerts() As String = {  
Server.MapPath("../Rez/JohnSmith.cer") }  
Dim theSig As Signature = theDoc.Form("Signature")  
If (theSig.Validate(theCerts)) And (Not  
theSig.IsModified) Then  
    theDoc.AddText("Signature valid at " +  
DateTime.Now.ToString())  
End If  
theDoc.Save(Server.MapPath("SignedAndValidated.
```

```
' Validate using the Windows Certificate Store  
Dim theDoc As Doc = New Doc()
```

```
theDoc.Read(Server.MapPath("../Rez/Signed.pdf"))
Dim theStore As X509Store = New
X509Store(StoreName.Root, StoreLocation.LocalMachine)
theStore.Open(OpenFlags.ReadOnly)
Dim theSig As Signature = theDoc.Form("Signature")
If (theSig.Validate(theStore.Certificates)) And (Not
theSig.IsModified) Then
    theDoc.AddText("Signature valid at " +
DateTime.Now.ToString())
End If
theStore.Close()
theDoc.Save(Server.MapPath("SignedAndValidated.pdf"))
```

# IsModified Property

---



Type	Default Value	Read Only	Description
[C#] bool			True if the PDF has been tampered with after signing.
[Visual Basic] Boolean	true	Yes	

This property allows you to determine if the PDF has been tampered with after signing.

It is important to understand that certain types of updates are allowed. The validity of this property relates to the validity of the particular revision of the document. So it allows you to determine if the original PDF has been tampered with in any way.

However it is possible to update PDF by incrementally updating the revision. This will provide a document which contains modifications but can be backtracked to show the exact state of the document at the point that a particular signature was applied.

---

## Example

Most commonly this is needed when multiple signatures are being applied as each signature needs to be applied to a new revision of the document and to be valid for that particular revision.

See also the [Validate](#) function.



# IsSecure Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	Yes	True if the signature and document look well formed and well applied.

True if the signature and document look well formed and well applied.

A signature with a PDF document occupies a specific portion of that document and should cover the rest of the document. However it is possible to create signatures which conform technically but are badly or suspiciously applied.

---

## Example

For example one could create a signature which did not cover the start of a document. While this is legal it is not sensible and this kind of structure should be regarded as insecure.

This property can be used to determine whether the signature has been well and sensibly applied.

# IsTrusted Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	false	Yes	True if the signature's certificate is trusted according to the validation policy.

See the [Validate](#) function and the [ValidationPolicy](#) property.

---

**Example**

# IsValid Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	false	Yes	True if the signature's certificate was time-valid when the document was signed.

See the [Validate](#) function.

---

Example

# Location Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>	null	No	The location of the signing.
[Visual Basic] <code>String</code>			

The location of the signing.

If you wish to set a value for this property, you should do so before calling the [Sign](#) function.

---

## Notes

This property can take null as a value. This indicates that no location was provided.

Note that this property can only be relied on if the certificate is valid. You can check whether the certificate is valid using the [Validate](#) function.

---

## Example

See the [Sign](#) function.

# Reason Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>	null	No	The reason for signing.
[Visual Basic] <code>String</code>			

The reason for signing.

If you wish to set a value for this property, you should do so before calling the [Sign](#) function.

---

## Notes

This property can take null as a value. This indicates that no reason was given.

Note that this property can only be relied on if the certificate is valid. You can check whether the certificate is valid using the [Validate](#) function.

---

## Example

See the [Sign](#) function.

# Signer Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>	See description.	No	The name of the person signing.
[Visual Basic] <code>String</code>			

The name of the person signing.

When you call the Sign function the name of the signer is automatically updated to match the name associated with the private key.

---

## Notes

Note that this property can only be relied on if the certificate is valid. You can check whether the certificate is valid using the [Validate](#) function.

You will not need to change the value of this property unless you are writing low level signature manipulation code.

---

## Example

None.

# SigningUtcTime Property

---



Type	Default Value	Read Only	Description
[C#] <code>DateTime</code>	See description.	No	The time of signing in UTC format.
[Visual Basic] <code>DateTime</code>			

- may throw `Exception()`

The time of signing in UTC format.

When you call the Sign function, the UTC time is automatically updated.

Note that this property can only be relied on if the certificate is valid. You can check whether the certificate is valid using the [Validate](#) function.

---

## Notes

You will not need to change the value of this property unless you are writing low level signature manipulation code.

If the format of the date in the PDF is incorrect, then querying the value of this property may result in an exception being thrown.

---

Example

None.

# TimestampServiceUrl Property

---



Type	Default Value	Read Only	Description
[C#] System.Uri	null	No	The URL to a time-stamping service.
[Visual Basic] System.Uri			

If this property is not null when you call [Sign](#), an X.509 [RFC 3161](#) time-stamped signature will be created by using the service provided in the URL. Note that this property serves as an option for [Sign](#), and is not persisted in the PDF document.

If your time-stamping service provider requires authentication, you can specify your credentials in the Uri. For example, "https://username:password@example.com/tsp".

---

## Notes

ABCpdf uses System.Net.WebRequest to send the time-stamping request. You can use [System.Net.ServicePointManager.ServerCertificateValidationCallback](#) to customize trust relationship establishment when you connect to an SSL/TLS channel.

None.

---

## Example



# ValidationPolicy Property

---



Type	Default Value	Read Only	Description
[C#]ValidationPolicyType			The validation policy for certificates.
[Visual Basic]ValidationPolicyType	EntireChainTrust	No	The validation policy for certificates.

The validation policy for certificates.

The ValidationPolicyType enumeration may take the following values:

- EntireChainTrust
- CertificateSignatureOnly

When you call [Validate](#), you can choose to provide additional certificates. This property indicates how such certificates are used and how the certificates in the document signature are validated.

When you set this property to EntireChainTrust, ABCpdf checks whether the certificates in the document signature can be validated against a trusted root Certificate Authority (trust anchor) by performing a X.509 certification path validation as described in [RFC 5280](#). ABCpdf will use the certificates found in "Trusted Root Certification Authorities" in the Windows Certificate Store or the local machine as trust anchors. Certificates you pass into the Validate method will be regarded as additional trust anchors.

---

## Notes

When you set this property to CertificateSignatureOnly, ABCpdf

checks whether at least one of the certificates in the document signature has been signed with the public key of one of the certificates passed to `Validate`. When this value is set, `Validate` does not check the Windows Certificate Store. If no certificates have been passed to `Validate`, an exception will be thrown.

`EntireChainTrust` is a sensible default because it is how Acrobat builds up a certificate trust chain and also how PKI generally works.

None.

---

**Example**

# StreamObject Constructor

---



StreamObject Constructor.

## [C#]

```
StreamObject(ObjectSoup soup)
StreamObject(ObjectSoup soup, byte[]
data)
StreamObject(ObjectSoup soup, string
path)
```

## [Visual Basic]

### Syntax

```
Sub New(soup As ObjectSoup)
Sub New(soup As ObjectSoup, data() As
Byte)
Sub New(soup As ObjectSoup, path As
String)
```

- may throw Exception()

### Params

Name	Description
soup	The ObjectSoup to contain the newly created StreamObject.
data	An array of bytes which should be placed in the StreamObject.
path	A path to a file containing data which should be placed in the StreamObject.

Create a StreamObject.

If no arguments are passed then the StreamObject contains no data. No exception will be thrown.

---

**Notes**

If an array of bytes is passed then the StreamObject data will be initialized with the contents of the array. No exception will be thrown.

If a string is passed then the StreamObject data will be initialized with the contents of the file. If the file cannot be read then an exception is thrown.

---

**Example**

None.



## ClearData Function

---

Clear the data and compression settings for the stream.

### [C#]

```
void ClearData()
```

---

### Syntax

### [Visual Basic]

```
Sub ClearData()
```

---

### Params

Name	Description
none	

---

### Notes

Clears all data and compression settings leaving the stream empty of data.

---

### Example

None.



# ClearCachedDecompressed Function

---

Clear the cached, decompressed data for the stream.

## [C#]

```
void ClearCachedDecompressed()
```

---

## Syntax

## [Visual Basic]

```
Sub ClearCachedDecompressed()
```

---

## Params

Name	Description
none	

The uncompressed data are retrieved from compressed streams during most operations, but the streams are still compressed. The uncompressed data are sometimes cached so that later operations (including the actual decompression of the streams) need not invoke the decompression algorithm.

This increases the memory usage, and it is not desirable if the document is to be kept in memory for an extended period of time.

---

## Notes

This method discards the cached, decompressed data so that a re-read of the document is not required to release the memory.

You can apply the effect to multiple stream objects using [Doc.ClearCachedDecompressedStreams](#).

---

**Example**

None.



# Compress Function

---

Compress the data in the stream.

## [C#]

```
void Compress()
```

---

### Syntax

## [Visual Basic]

```
Sub Compress()
```

---

### Params

Name	Description
none	

---

### Notes

Compresses the data in the stream if it is not already compressed.

---

### Example

None.

# CompressAscii85 Function

---



Compress the data in the stream using ASCII 85 encoding.

## [C#]

```
bool CompressAscii85()  
bool CompressAscii85(bool force)
```

---

## Syntax

## [Visual Basic]

```
Sub CompressAscii85() As Boolean  
Sub CompressAscii85(force As  
Boolean) As Boolean
```

---

## Params

Name	Description
force	Whether to force the stream to be compressed in this way.
return	Whether the compression was applied.

Compress the data in the stream using ASCII 85 encoding.

ASCII 85 replaces each five bytes of data with a four character ASCII notation. As such it is not strictly a compression method and it will result in the data size increasing by about twenty

percent. However the resultant data will be ASCII, which can be useful if you need to treat the PDF data as text.

PDF streams allow a set of compression filters to be applied to a stream of data. For example one might want to apply Flate compression and then ASCII85 encode the result. This is represented as two compression filters in sequence.

---

**Notes**

This function does not decompress the stream. So if compression is already present, then this method will compress the already-encoded data and append a compression specification to the sequence.

ABCpdf tries to avoid creating certain compression sequences. Some compression types on some objects are illegal. Some sequences are legal but not supported within Acrobat (though they are in most other viewers). However these are unusual situations and you are unlikely to ever see them.

You can override this behavior by forcing the compression to take place. However if you do this you may end up creating a document which is invalid or unviewable in Acrobat.

---

**Example**

None.

# CompressAsciiHex Function

---



Compress the data in the stream using the ASCII Hex encoding.

## [C#]

```
bool CompressAsciiHex()  
bool CompressAsciiHex(bool force)
```

---

## Syntax

## [Visual Basic]

```
Sub CompressAsciiHex() As Boolean  
Sub CompressAsciiHex(force As  
Boolean) As Boolean
```

---

## Params

Name	Description
force	Whether to force the stream to be compressed in this way.
return	Whether the compression was applied.

Compress the data in the stream using ASCII Hex encoding.

ASCII Hex replaces each byte of data with a two byte hexadecimal notation. As such it is not strictly a compression method and it will result in a doubling of the size of the data. However

the resultant data will be ASCII, which can be useful if you need to treat the PDF data as text.

PDF streams allow a set of compression filters to be applied to a stream of data. For example one might want to apply Flate compression and then ASCII85 encode the result. This is represented as two compression filters in sequence.

---

**Notes**

This function does not decompress the stream. So if compression is already present, then this method will compress the already-encoded data and append a compression specification to the sequence.

ABCpdf tries to avoid creating certain compression sequences. Some compression types on some objects are illegal. Some sequences are legal but not supported within Acrobat (though they are in most other viewers). However these are unusual situations and you are unlikely to ever see them.

You can override this behavior by forcing the compression to take place. However if you do this you may end up creating a document which is invalid or unviewable in Acrobat.

---

**Example**

None.

# CompressFlate Function

---



Compress the data in the stream using Flate compression.

## [C#]

```
bool CompressFlate()  
bool CompressFlate(bool force)
```

---

## Syntax

## [Visual Basic]

```
Sub CompressFlate() As Boolean  
Sub CompressFlate(force As  
Boolean) As Boolean
```

---

## Params

Name	Description
force	Whether to force the stream to be compressed in this way.
return	Whether the compression was applied.

Compress the data in the stream using Flate compression.

Flate compression is a standard, fast and efficient lossless compression method. It is used as the basis of file formats like ZIP and image formats like PNG. In most situations this

is the method you should prefer.

PDF streams allow a set of compression filters to be applied to a stream of data. For example one might want to apply Flate compression and then ASCII85 encode the result. This is represented as two compression filters in sequence.

---

Notes

This function does not decompress the stream. So if compression is already present, then this method will compress the already-encoded data and append a compression specification to the sequence.

ABCpdf tries to avoid creating certain compression sequences. Some compression types on some objects are illegal. Some sequences are legal but not supported within Acrobat (though they are in most other viewers). However these are unusual situations and you are unlikely to ever see them.

You can override this behavior by forcing the compression to take place. However if you do this you may end up creating a document which is invalid or unviewable in Acrobat.

---

Example

None.

# CompressRunLength Function

---



Compress the data in the stream using run length encoding.

## [C#]

```
bool CompressRunLength()  
bool CompressRunLength(bool  
force)
```

---

## Syntax

## [Visual Basic]

```
Sub CompressRunLength() As  
Boolean  
Sub CompressRunLength(force As  
Boolean) As Boolean
```

---

## Params

Name	Description
force	Whether to force the stream to be compressed in this way.
return	Whether the compression was applied.

Compress the data in the stream using run length encoding.

Run length encoding is a simple form of compression which replaces sequences of

identical bytes, with a count and the value of the bytes in the sequence. While simple to encode and decode, it is relatively inefficient and options such as [CompressFlate](#) should be preferred.

PDF streams allow a set of compression filters to be applied to a stream of data. For example one might want to apply Flate compression and then ASCII85 encode the result. This is represented as two compression filters in sequence.

---

**Notes**

This function does not decompress the stream. So if compression is already present, then this method will compress the already-encoded data and append a compression specification to the sequence.

ABCpdf tries to avoid creating certain compression sequences. Some compression types on some objects are illegal. Some sequences are legal but not supported within Acrobat (though they are in most other viewers). However these are unusual situations and you are unlikely to ever see them.

You can override this behavior by forcing the compression to take place. However if you do this you may end up creating a document which is invalid or unviewable in Acrobat.

---

**Example**

None.





# Decompress Function

---

Decompress the data in the stream.

## [C#]

```
bool Decompress()
```

---

### Syntax

## [Visual Basic]

```
Function Decompress() As Boolean
```

---

### Params

Name	Description
return	Whether the data in the stream is uncompressed.

This method removes all compression from the stream.

---

### Notes

Under some circumstances it may not be possible to fully decompress the stream. In these situations the function will return false.

None.

---

### Example





## GetData Function

---

Get the raw binary content of the stream.

### [C#]

```
byte[] GetData()
```

---

### Syntax

### [Visual Basic]

```
Function GetData() As Byte()
```

---

### Params

Name	Description
return	The raw binary content of the stream.

Get the raw binary content of the stream.

---

### Notes

If the stream is compressed then this data will be compressed.

---

### Example

None.





# GetText Function

---

Get the content of the stream as a string.

## [C#]

```
string GetText()
```

---

### Syntax

## [Visual Basic]

```
Function GetText() As String
```

---

### Params

Name	Description
return	The content of the stream as a string.

Get the content of the stream in string format.

---

### Notes

You will generally want to ensure your stream is decompressed before using this function.

---

### Example

None.





# SetData Function

---

Set the raw binary content of the stream.

## [C#]

```
void SetData(byte[] value)
void SetData(byte[] value, int
index, int count)
void SetData(StreamObject source)
```

## [Visual Basic]

### Syntax

```
Sub SetData(value() As Byte)
Sub SetData(value() As Byte,
index As Integer, count As
Integer)
Sub SetData(source As
StreamObject)
```

### Params

Name	Description
value	The raw binary content to be assigned to the stream.
index	The index in the array at which copying should start.
count	The number of bytes to copy.
source	The source stream from which to copy the data.

Set the raw binary content of the stream.

Compression settings are unaltered. So if - for example - you have a stream which is Flate compressed you must use `SetData` with Flate compressed data.

---

**Notes**

For this reason you may wish to call `ClearData` before using `SetData`.

Using the overload which accepts a `StreamObject` is equivalent to, but more efficient than, getting the data from the source `StreamObject` and then using `SetData` to assign it to this one.

---

**Example**

None.



## SetFile Function

---

Set the raw binary content of the stream using data from a file.

### [C#]

```
void SetFile(string path)
```

### [Visual Basic]

```
Sub SetFile(path As String)
```

- may throw Exception()

---

#### Syntax

---

#### Params

Name	Description
path	A path to a file containing data which should be placed in the StreamObject.

Set the raw binary content of the stream using data read from a file. If the file cannot be read then an exception is thrown.

Compression settings are unaltered. So if - for example - you have a stream which is Flate compressed you must use SetFile with Flate compressed data.

---

#### Notes

For this reason you may wish to call `ClearData` before using `SetFile`.

---

**Example**

None.



# SetText Function

---

Set the content of the stream as a string.

## [C#]

```
void SetText(string value)
```

---

### Syntax

## [Visual Basic]

```
Sub SetText(value As String)
```

---

### Params

Name	Description
value	The content of the stream as a string.

Get the content of the stream in string format.

---

### Notes

Compression settings are unaltered. For this reason you may wish to call [ClearData](#) before using [SetData](#).

---

### Example

None.



# Compressed Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the stream data is compressed or otherwise encoded.

Whether the stream data is compressed or otherwise encoded.

---

Notes

None.

---

Example

# Compression Property

---



Type	Default	Read Only	Description
[C#] CompressionType	See description.	Yes	The primary compression type.
[Visual Basic] CompressionType			

A single stream can be compressed and encoded using multiple methods. This property reflects the primary compression type used by the stream.

Compression types which result in high levels of compression (e.g. JPEG) are considered more important than those that do not (e.g. ASCII 85).

The CompressionType enumeration may take the following values:

- None
- Unknown
- Flate
- LZW
- Ccitt
- Jpeg
- Jpx
- AsciiHex
- Ascii85
- RunLength

- Jbig2
- Crypt

More details of these compression types can be found in Section 3.3 of the [Adobe PDF Specification](#).

---

**Example**

None.

# Compressions Property

---



Type	Default	Read Only	Description
[C#] CompressionType[]	See description.	Yes	All the compression types applied to the stream.
[Visual Basic] CompressionType()			

A stream can be compressed and encoded using multiple methods applied in sequence.

This property reflects all the compression types that have been applied to the stream.

---

## Notes

The [CompressionTypes](#) are ordered in terms of the complete decompression sequence required.

None.

---

## Example

# Length Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of bytes of encoded stream data.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The number of bytes of encoded stream data.

---

## Notes

None.

---

## Example

# ClearTextOperation Function

---



Removes any cached `TextOperation` and `TextFragments` associated with this object

## [C#]

```
void ClearTextOperation()
```

---

### Syntax

## [Visual Basic]

```
Sub ClearTextOperation()
```

---

### Params

Name	Description
none	n/a

Removes any cached `TextOperation` and `TextFragments` associated with this object.

---

### Notes

This can reduce memory use or it can be used to allow text information to be regenerated after the `TextLayer` contents have been updated.

None.

---

### Example



# Characters Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of characters appearing on the page.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The number of characters appearing on the page.

---

Notes

None.

---

Example

# EndPos Property

---



Type	Default	Read Only	Description
[C#] <code>XPoint</code>			
[Visual Basic] <code>XPoint</code>	n/a	Yes	The point defining the end position of the text.

The `XPoint` defining the end position of the text.

---

## Notes

This point is encoded in PDF coordinates rather than any abstracted coordinate space.

---

## Example

None.

# Lines Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The number of lines appearing on the page.

The number of lines appearing on the page.

---

Notes

None.

---

Example

# Previous Property

---



Type	Default	Read Only	Description
[C#] <code>TextLayer</code>			
[Visual Basic] <code>TextLayer</code>	n/a	Yes	The previous text object in the text chain.

The previous text object in the text chain.

---

## Notes

If the object is not part of a chain (or is at the head of the chain) this property will be null.

---

## Example

None.

# Truncated Property

---



Type	Default	Read Only	Description
[C#] bool			Whether the text had to be truncated.
[Visual Basic] Boolean	n/a	Yes	

Whether the text had to be truncated.

This property indicates if all the assigned text could be displayed. It is true if the text had to be truncated and false if not.

---

## Notes

Only text objects that have been truncated can be chained to. If the object has not been truncated then there's no more text to display.

None.

---

## Example

# FullText Property

---



Type	Default	Read Only	Description
[C#] string			The full text provided in the initial call to AddHtml or AddText.
[Visual Basic] String	n/a	Yes	

The full text provided in the initial call to AddHtml or AddText.

---

## Notes

This text is shared between all the items in a chain. It is in plain text and does not indicate any styles.

---

## Example

None.

## TextStart Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The offset to the first character drawn onto this layer.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The offset to the first character (in the [FullText](#)) to be drawn onto this layer.

---

### Notes

None.

---

### Example

# TextEnd Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The offset to the last character drawn onto this layer.

The offset to the last character (in the `FullText`) to be drawn onto this layer.

---

## Notes

None.

---

## Example

# TextEnd Property

---



Type	Default	Read Only	Description
[C#] int [Visual Basic] Integer	n/a	No	The offset to the character which will be drawn at the start of next item in the chain.

The offset to the character (in the [FullText](#)) which will be drawn as the next item in the chain.

---

## Notes

By changing this property you can control the character at which the next item in the chain starts. So by increasing the value you can skip characters between chained items. By decreasing the value you can repeat text from the previous item.

---

## Example

None.

# TextFragments Property

---



Type	Default Value	Read Only	Description
[C#] <code>IList&lt;TextFragment&gt;</code>	n/a	Yes	The TextFragments describing the precise layout of this text layer
[Visual Basic] <code>IList&lt;TextFragment&gt;</code>			

The [TextFragments](#) describing the precise layout of this text layer.

The properties of this object can be used to establish precise metrics for each of the items of text in the layer.

---

## Notes

The operation is created the first time this property is accessed and it is cached thereafter. To reduce memory use, you may wish to call [ClearTextOperation](#) after you have established the precise metrics that you require.

None.

---

## Example



# TextOperation Property

---



Type	Default Value	Read Only	Description
[C#] <a href="#">TextOperation</a>	n/a	Yes	A <a href="#">TextOperation</a> describing the precise layout of this text layer
[Visual Basic] <a href="#">TextOperation</a>			

A [TextOperation](#) describing the precise layout of this text layer.

The properties of this object can be used to establish precise metrics for each of the items of text in the layer.

---

## Notes

The operation is created the first time this property is accessed and it is cached thereafter. To reduce memory use, you may wish to call [ClearTextOperation](#) after you have established the precise metrics that you require.

None.

---

## Example



# ContentHeight Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The content height of the image in pixels.

The content height of the image. It is usually very close to [ScrollHeight](#).

---

## Notes

When adding HTML you only see one page at a time. This property allows you to find the height of the HTML before it is paged onto the PDF.

---

## Example

None.

# ContentWidth Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The content width of the image in pixels.

The content width of the image. It is usually very close to [ScrollWidth](#).

---

## Notes

When adding HTML you only see one page at a time. This property allows you to find the width of the HTML before it is paged onto the PDF.

---

## Example

None.

# PageHeight Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The height of the content on the current page in pixels.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The height of the content on the current page.

Note that the content may not vertically completely fill the area into which the view has been drawn.

---

## Notes

This may occur if the page is has been broken early to ensure that content is split at a sensible location. Or alternatively, it may occur if the content was not as tall as the area specified while it horizontally fills the area.

---

## Example

None.

# PageOffset Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The offset to the top of the current page in pixels.

The vertical offset from the top of the underlying graphic to the top of the current page.

---

## Notes

None.

---

## Example

# PageWidth Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The width of the content on the current page in pixels.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The width of the content on the current page.

---

## Notes

None.

---

## Example

# ScrollHeight Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The scroll height of the image in pixels.

The scroll height of the image.

---

## Notes

It is the same scroll height value as retrieved using script inside HTML.

---

## Example

None.

# ScrollWidth Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The scroll width of the image in pixels.

The scroll width of the image.

---

## Notes

It is the same scroll width value as retrieved using script inside HTML.

---

## Example

None.

# Truncated Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	Yes	Whether the image is truncated.

Whether the image is truncated.

---

Notes

None.

---

Example

# FromString Function

---



Create an appropriate type of Atom from a raw PDF string representation.

## [C#]

```
static Atom FromString(string value)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromString(value As String) As Atom
```

---

## Params

Name	Description
value	The string representing the value of the object.
return	The resulting Atom.

The text you pass this function must be in native PDF format. This means that unusual characters in text strings must be appropriately escaped.

---

## Notes

For full details of the way that PDF objects are represented you should see the Adobe PDF Specification.

---

**Example**

None.



## GetBool Function

---

Gets the Boolean value from the Atom if it is a BoolAtom.

### [C#]

```
static bool GetBool(Atom atom)
```

---

### Syntax

### [Visual Basic]

```
Shared Function GetBool(atom As Atom) As Boolean
```

---

### Params

Name	Description
atom	The Atom to get the Boolean from.
return	The returned Boolean.

Get the Boolean value from the supplied Atom if it is a BoolAtom.

---

### Notes

If the atom is not a BoolAtom or if it is null then false will be returned.

---

None.

## Example

# GetDouble Function

---



Gets the double value from the Atom if it is a NumAtom.

## [C#]

```
static double GetDouble(Atom  
atom)
```

---

## Syntax

## [Visual Basic]

```
Shared Function GetDouble(atom As  
Atom) As Double
```

---

## Params

Name	Description
atom	The Atom to get the double from.
return	The returned double.

Get the double value from the supplied Atom if it is a NumAtom.

---

## Notes

If the atom is not a NumAtom or if it is null then zero will be returned.

---

**Example**

None.



## GetID Function

---

Gets the Object ID value from the Atom if it is a RefAtom.

### [C#]

```
static int GetID(Atom atom)
```

---

### Syntax

### [Visual Basic]

```
Shared Function GetID(atom As Atom) As Integer
```

---

### Params

Name	Description
atom	The Atom to get the Object ID from.
return	The returned Object ID.

Get the Object ID from the supplied Atom if it is a RefAtom.

---

### Notes

If the atom is not a RefAtom or if it is null then zero will be returned.

---

None.

## Example



## GetInt Function

---

Gets the integer value from the Atom if it is a NumAtom.

### [C#]

```
static int GetInt(Atom atom)
```

---

### Syntax

### [Visual Basic]

```
Shared Function GetInt(atom As Atom) As Integer
```

---

### Params

Name	Description
atom	The Atom to get the integer from.
return	The returned integer.

Get the integer value from the supplied Atom if it is a NumAtom.

---

### Notes

If the atom is not a NumAtom or if it is null then zero will be returned.

---

None.

## Example

# GetItem Function

---



Gets the specified item from the Atom if it is of a type which contains other Atoms.

## [C#]

```
static Atom GetItem(Atom atom,  
string key)  
static Atom GetItem(Atom atom,  
int index)
```

---

## Syntax

## [Visual Basic]

```
Shared Function GetItem(atom As  
Atom, key As String) As Atom  
Shared Function GetItem(atom As  
Atom, index As Integer) As Atom
```

---

## Params

Name	Description
atom	The Atom to get the item from.
key	The name of the item to be retrieved.
index	The index of the item to be retrieved.
return	The returned Atom.

Both [DictAtoms](#) and [ArrayAtoms](#) can contain other Atoms.

This function allows you to get an item from an ArrayAtom or DictAtom.

Entries in ArrayAtoms can be referenced by number. If the supplied atom is not an ArrayAtom or if it is null or if the index is outside the bounds of the array then null will be returned.

---

**Notes**

Entries in DictAtoms can be referenced by name or by number. If the supplied atom is not a DictAtom or if it is null or if the name is not a key in the dictionary or if the index is outside the bounds of the dictionary then null will be returned.

None.

---

**Example**



## GetName Function

---

Gets the Name value from the Atom if it is a NameAtom.

### [C#]

```
static string GetName(Atom atom)
```

---

### Syntax

### [Visual Basic]

```
Shared Function GetName(atom As Atom) As String
```

---

### Params

Name	Description
atom	The Atom to get the name from.
return	The returned name.

Get the Name value from the supplied Atom if it is a NameAtom.

---

### Notes

If the atom is not a NameAtom or if it is null then an empty string ("") will be returned.

---

None.

## Example



# GetText Function

---

Gets the Text value from the Atom if it is a StringAtom.

## [C#]

```
static string GetText(Atom atom)
```

---

## Syntax

## [Visual Basic]

```
Shared Function GetText(atom As Atom) As String
```

---

## Params

Name	Description
atom	The Atom to get the text from.
return	The returned text.

Get the Text value from the supplied Atom if it is a StringAtom.

---

## Notes

If the atom is not a StringAtom or if it is null then an empty string ("") will be returned.

---

None.

## Example

# RemoveItem Function

---



Removes the named entry from the Atom if it is a DictAtom.

## [C#]

```
static void RemoveItem(Atom atom,  
string key)
```

---

## Syntax

## [Visual Basic]

```
Shared Sub RemoveItem(atom As  
Atom, key As String)
```

---

## Params

Name	Description
atom	The Atom from which the item should be removed.
key	The name of the item to be removed.

[DictAtoms](#) can contain other Atoms referenced by name.

This function allows you to remove a named item from a DictAtom.

---

## Notes

If the atom supplied is not a DictAtom then calling this function will have no effect.

---

**Example**

None.

# SetItem Function

---



Adds a specified item to the Atom if it is of a type which contains other Atoms.

## [C#]

```
static Atom SetItem(Atom atom,  
string key, Atom val)  
static Atom SetItem(Atom atom,  
int index, Atom val)
```

## [Visual Basic]

### Syntax

```
Shared Function SetItem(atom As  
Atom, key As String, val As Atom)  
As Atom  
Shared Function SetItem(atom As  
Atom, index As Integer, val As  
Atom) As Atom
```

### Params

Name	Description
atom	The Atom to which the item should be added.
key	The name of the item to be added.
index	The index at which the item should be added.
return	The returned Atom.

Both [DictAtoms](#) and [ArrayAtoms](#) can contain other Atoms.

This function allows you to add an item to an [ArrayAtom](#) or [DictAtom](#).

If the container Atom supplied is not a [DictAtom](#) or an [ArrayAtom](#) then insertion will not be successful.

Entries in [ArrayAtoms](#) can be referenced by number. If the container Atom supplied is an [ArrayAtom](#) and the index supplied is within the bounds of the array then insertion will be successful. If the index supplied is less than zero then the value Atom will be added to the end of the array. Again insertion will be successful.

---

Notes

Entries in [DictAtoms](#) can be referenced by name. If the container Atom supplied is a [DictAtom](#) and the key supplied is not empty then insertion will be successful.

If insertion was successful the function will return the Atom which was added. If it was not successful it will return null.

None.

---

Example

# GetData Function

---



The byte array representation of the Atom as it would appear in a PDF

## [C#]

```
byte[] GetData()
```

---

### Syntax

## [Visual Basic]

```
Function GetData() As Byte()
```

---

### Params

Name	Description
none	

---

### Notes

The byte array representation of the Atom as it would appear in a PDF.

---

### Example

None.



# Clone Function

---

Creates a deep copy of the current Atom.

## [C#]

```
Atom Clone()
```

---

## Syntax

## [Visual Basic]

```
Function Clone() As Atom
```

---

## Params

Name	Description
return	The newly created copy.

This function creates a new object that is a copy of this instance.

---

## Notes

The copy is a deep copy and all contained Atoms are copied as part of the clone process

---

## Example

None.





# Dispose Function

---

Dispose of the object.

## [C#]

```
void Dispose()  
protected void Dispose(bool  
disposing)
```

---

### Syntax

## [Visual Basic]

```
Sub Dispose()  
Protected Sub Dispose(disposing  
As Boolean)
```

---

### Params

Name	Description
none	

You can call this function to explicitly dispose of an object and reduce the garbage collection overhead.

This method follows the standard design pattern for objects implementing the IDisposable interface. The protected Dispose method can be overridden for sub-classes wishing to dispose of additional objects.

---

### Notes

Do not attempt to use an object after calling  
Dispose.

---

**Example**

None.



# Equals Function

---

Test whether the two Atoms are the same.

## [C#]

```
bool Equals(Atom other)
override bool Equals(object
test);
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As Atom) As
Boolean
Overrides Function Equals(test As
Object) As Boolean
```

---

## Params

Name	Description
other	The object to test against.
return	Whether the objects are equal.

This method can be used to determine whether the specified object is equal to the current object.

---

## Notes

Objects are considered equal if they refer to the same underlying object within the PDF document. So this method determines object

equality rather than value equality.

---

**Example**

None.



# GetHashCode Function

---

A hash code for the Atom.

## [C#]

```
override int GetHashCode()
```

---

### Syntax

## [Visual Basic]

```
Overrides Function GetHashCode()  
As Integer
```

---

### Params

Name	Description
return	The returned hash code.

---

### Notes

Derives a hash code suitable for use in hashing algorithms and data structures like hash tables.

---

### Example

None.



# ToString Function

---

The string representation of the Atom as it would appear in a PDF.

## [C#]

```
override string ToString()
```

---

## Syntax

## [Visual Basic]

```
Overrides Function ToString() As String
```

---

## Params

Name	Description
return	The string representation of the object.

This function derives the content of the object as it will be inserted into the final PDF document.

---

## Notes

Note that the the string value of an object may be large and it may contain unusual characters.

---

None.

## Example

# FromXRect Function

---



Create an ArrayAtom from a XRect representation

## [C#]

```
static ArrayAtom  
FromXRect(WebSupergoo.ABCpdf10.XRect  
value)
```

---

## Syntax

## [Visual Basic]

```
Shared Function FromXRect(value As  
WebSupergoo.ABCpdf10.XRect) As  
ArrayAtom
```

---

## Params

Name	Description
value	The XRect representing the value of the object.

Create an ArrayAtom from a XRect representation.

---

## Notes

The ArrayAtom will contain four [NumAtoms](#) corresponding to the horizontal and vertical elements of two diagonally opposite corners.

---

**Example**

None.

# FromXTransform Function

---



Create an ArrayAtom from a XTransform representation

## [C#]

```
static ArrayAtom  
FromXTransform(XTransform value)
```

---

## Syntax

## [Visual Basic]

```
Shared Function  
FromXTransform(value As  
XTransform) As ArrayAtom
```

---

## Params

Name	Description
value	The XTransform representing the value of the object.

Create an ArrayAtom from a XTransform representation.

---

## Notes

The ArrayAtom will contain six [NumAtoms](#) corresponding to the elements of the transform.

---

Example

None.

# FromContentStream Function

---



Create an array of Atoms from a byte array containing a sequence of PDF objects

## [C#]

```
static ArrayAtom FromContentStream(string value)
static ArrayAtom FromContentStream(byte[] value)
```

---

## Syntax

### [Visual Basic]

```
Shared Function FromContentStream(value As String) As ArrayAtom
Shared Function FromContentStream(value() As Byte) As ArrayAtom
```

Name	Description
value	The string holding the sequence of atoms.

---

## Params

Create an array of Atoms from a byte array containing a sequence of PDF objects.

---

## Notes

This method is useful for deconstructing PDF content streams for analysis and modification. To convert back into a content stream you can use the [Atom.GetData](#)

function.

This example shows how to use the `FromContentStream` function to parse and display a PDF content stream.

### [C#]

```
StringBuilder sb = new StringBuilder();
using (Doc doc = new Doc()) {
    doc.Read("spaceshuttle.pdf");
    Page page = doc.ObjectSoup[doc.Page] as
Page;
    StreamObject[] layers = page.GetLayers();
    MemoryStream st = new MemoryStream();
    foreach (StreamObject layer in layers) {
        if (!layer.Decompress())
            throw new Exception("Unable to
decompress stream.");
        byte[] data = layer.GetData();
        st.Write(data, 0, data.Length);
    }
    ArrayAtom array =
ArrayAtom.FromContentStream(st.ToArray());
    int indent = 0;
    HashSet<string> indentPlus = new
HashSet<string>(new string[] { "q", "BT" })
    HashSet<string> indentMinus = new
HashSet<string>(new string[] { "Q", "ET" })
    IList<Tuple<string, int>> items =
OpAtom.Find(array);
    int index = 0;
    foreach (var pair in items) {
        string op =
((OpAtom)array[pair.Item2]).Text;
        // add indent to code
```

```

    if (indentMinus.Contains(op))
        indent--;
    for (int i = 0; i < indent; i++)
        sb.Append(" ");
    // write out the operators
    for (int i = index; i <= pair.Item2; i+
{
    if (i != index)
        sb.Append(" ");
    Atom item = array[i];
    // we write arrays out individually s
that
    // we can override default cr lf
behavior
    ArrayAtom itemArray = item as ArrayAt
    if (itemArray != null) {
        int n = itemArray.Count;
        for (int j = 0; j < n; j++) {
            sb.Append(itemArray[j].ToString())
            if (j != n - 1)
                sb.Append(" ");
        }
    }
    else {
        sb.Append(item.ToString());
    }
}
    sb.AppendLine();
    if (indentPlus.Contains(op))
        indent++;
    index = pair.Item2 + 1;
}
// write out any atoms that are left over
for (int i = index; i < array.Count; i++)
    sb.Append(" ");
    sb.Append(array[i].ToString());
}

```

```

}
using (Doc doc = new Doc()) {
    doc.Font = doc.AddFont("Courier");
    doc.Rect.Inset(20, 20);
    doc.AddText(sb.ToString());
    doc.Save("PageContents.pdf");
}

```

## [Visual Basic]

```

Dim sb As New StringBuilder()
Using doc As New Doc()
    doc.Read("spaceshuttle.pdf")
    Dim page As Page =
TryCast(doc.ObjectSoup(doc.Page), Page)
    Dim layers As StreamObject() =
page.GetLayers()
    Dim st As New MemoryStream()
    For Each layer As StreamObject In layer
        If Not layer.Decompress() Then
            Throw New Exception("Unable to
decompress stream.")
        End If
        Dim data As Byte() = layer.GetData()
        st.Write(data, 0, data.Length)
    Next
    Dim array As ArrayAtom =
ArrayAtom.FromContentStream(st.ToArray())
    Dim indent As Integer = 0
    Dim indentPlus As New HashSet(Of String)
(New String() {"q", "BT"})
    Dim indentMinus As New HashSet(Of String)
(New String() {"Q", "ET"})
    Dim items As IList(Of Tuple(Of String,
Integer)) = OpAtom.Find(array)
    Dim index As Integer = 0
    For Each pair As var In items

```

Example

```

    Dim op As String =
DirectCast(array(pair.Item2), OpAtom).Text
    ' add indent to code
    If indentMinus.Contains(op) Then
        indent -= 1
    End If
    For i As Integer = 0 To indent - 1
        sb.Append(" ")
    Next
    ' write out the operators
    For i As Integer = index To pair.Item
        If i <> index Then
            sb.Append(" ")
        End If
        Dim item As Atom = array(i)
        ' we write arrays out individually
that
        ' we can override default cr lf
behavior
        Dim itemArray As ArrayAtom =
TryCast(item, ArrayAtom)
        If itemArray IsNot Nothing Then
            Dim n As Integer = itemArray.Count
            For j As Integer = 0 To n - 1
                sb.Append(itemArray(j).ToString)
                If j <> n - 1 Then
                    sb.Append(" ")
                End If
            Next
        Else
            sb.Append(item.ToString())
        End If
    Next
    sb.AppendLine()
    If indentPlus.Contains(op) Then
        indent += 1
    End If

```

```

        index = pair.Item2 + 1
    Next
    ' write out any atoms that are left over
    For i As Integer = index To array.Count
1
        sb.Append(" ")
        sb.Append(array(i).ToString())
    Next
End Using
Using doc As New Doc()
    doc.Font = doc.AddFont("Courier")
    doc.Rect.Inset(20, 20)
    doc.AddText(sb.ToString())
    doc.Save("PageContents.pdf")
End Using
End Sub

```

```

q
/CS1 cs
1 scn
/GS1 gs
1 i
19.1 735.7 574.5 -53.9 re
f*
0 0 0 1 k
19.25 774 574.15 -38.25 re
f*
q
573.3003 0 0 648.35 19.45 19.1001
cm
/Im1 Do
Q
BT
/F4 1 Tf
36 0 0 36 31.5 696.78 Tm
0.019 Tc
0 Tw
(THE 21) Tj
21 0 0 21 142.92 708.84 Tm
/GS2 gs
(ST) Tj
36 0 0 36 165.24 696.78 Tm
/GS1 gs
0.018 Tc
0.001 Tw
( CENTUR) 18 (Y SP) 88 (ACE SHUT)
-17 (TLE) TJ
/F5 1 Tf

```

PageContents.pdf





# ArrayAtom Constructor

---

ArrayAtom Constructor.

**[C#]**

```
ArrayAtom()
```

---

Syntax

**[Visual Basic]**

```
Sub New()
```

---

Params

Name	Description
none	

---

Notes

Create an empty ArrayAtom.

---

Example

None.



# CopyTo Function

---

Copies the Atoms into an array.

## [C#]

```
void CopyTo(Atom[] array, int  
index)
```

---

## Syntax

## [Visual Basic]

```
Sub CopyTo(array As Atom(), index  
As Integer)
```

---

## Params

Name	Description
array	The array that is the destination for the elements.
index	The zero-based index in array at which copying begins.

Copies the elements of the Collection to an array starting at a particular array index.

---

## Notes

The array must be one-dimensional and have zero-based indexing.

---

Example

None.



# Add Function

---

Add an item to the end of the array.

## [C#]

```
int Add(Atom atm)
int Add(int num)
int Add(double real)
int Add(string str)
```

## [Visual Basic]

Syntax

```
Function Add(atm As Atom) As
Integer
Function Add(num As Integer) As
Integer
Function Add(real As Double) As
Integer
Function Add(str As String) As
Integer
```

Params

Name	Description
atm	The Atom to be added.
num	The integer to be added.
real	The floating point value to be added.
str	The string to be added.
return	The position in which the new element was inserted.

This method adds an item to the array.

You can add an Atom directly into the array or you can use one of the overloaded operators to add numbers or strings.

When you add a string this is encapsulated within a [StringAtom](#) and then inserted. When you add an integer or floating point value this is converted to a [NumAtom](#) and then inserted.

---

**Notes**

These operations are more efficient than creating an Atom and adding it yourself.

Atoms can exist in only one place at a time. If the Atom supplied is already contained by another object then a [Clone](#) of the Atom is added.

Adding a null value will result in a [NullAtom](#) being added to the array.

---

**Example**

None.



# Clear Function

---

Removes all Atoms from the array.

## [C#]

```
void Clear()
```

---

## Syntax

## [Visual Basic]

```
Sub Clear()
```

---

## Params

Name	Description
none	

---

## Notes

Removes all Atoms from the array.

---

## Example

None.

# Contains Function

---



Determines whether the array contains a specific Atom.

## [C#]

```
bool Contains(Atom value)
```

---

## Syntax

## [Visual Basic]

```
Function Contains(value As Atom)  
As Boolean
```

---

## Params

Name	Description
value	The object to locate.
return	True if the object is found, otherwise false.

---

## Notes

Determines whether the Collection contains a specific Atom.

---

## Example

None.





# IndexOf Function

---

Determines the index of a specific Atom.

## [C#]

```
int IndexOf(Atom value)
```

---

### Syntax

## [Visual Basic]

```
Function IndexOf(value As Atom)  
As Integer
```

---

### Params

Name	Description
value	The object to locate in the Collection.
return	If found, the index of value, otherwise -1.

---

### Notes

Determines the index of a specific Atom in the Collection.

---

### Example

None.



# Insert Function

---



Inserts an Atom into the array at the specified position.

## [C#]

```
void Insert(int index, Atom  
value)
```

## [Visual Basic]

Syntax

```
Sub Insert(index As Integer,  
value As Atom)
```

- may throw `ArgumentOutOfRangeException()`

Params

Name	Description
index	The zero-based index at which value should be inserted.
value	The Atom to insert into the array.

Inserts an Atom into the array at the specified position.

If the index equals the number of items in the

array then the Atom is appended to the end.

---

**Notes**

Atoms can exist in only one place at a time. If the Atom supplied is already contained by another object then a **Clone** of the Atom is added.

Adding a null value will result in a **NullAtom** being added to the array.

If the index is not a valid index this method throws an **ArgumentOutOfRangeException**.

---

**Example**

None.



# Remove Function

---

Removes an Atom from the array.

## [C#]

```
bool Remove(Atom value)
```

---

## Syntax

## [Visual Basic]

```
Function Remove(value As Atom) As Boolean
```

---

## Params

Name	Description
value	The Atom to be removed.
return	True if the Atom is removed, otherwise false.

When an Atom is removed the elements that follow the removed element move up to occupy the vacated spot.

---

## Notes

---

None.

## Example



## RemoveAt Function

---

Removes an Atom at a specified position from the array.

### [C#]

```
void RemoveAt(int index)
```

### [Visual Basic]

```
Sub RemoveAt(index As Integer)
```

#### Syntax

- may throw `ArgumentOutOfRangeException()`

#### Params

Name	Description
index	The zero-based index of the item to remove.

If the index is not valid then an `ArgumentOutOfRangeException` will be thrown.

#### Notes

When an Atom is removed the elements that follow the removed element move up to occupy the vacated spot.

---

**Example**

None.

# AddRange Function

---



Adds the elements in the supplied array at the end of this array

## [C#]

```
void AddRange(ArrayAtom array)
```

## [Visual Basic]

```
Sub AddRange(array As ArrayAtom)
```

- may throw Exception()

---

### Syntax

---

### Params

Name	Description
array	The array whose elements should be added.

Adds the elements in the supplied array at the end of this array.

---

### Notes

If the supplied array is null then an exception will be raised.

---

Example

None.



# Equals Function

---

Test whether the two ArrayAtoms are the same

## [C#]

```
bool Equals(ArrayAtom other)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As  
ArrayAtom) As Boolean
```

---

## Params

Name	Description
other	The ArrayAtom to test against.

Test whether the two ArrayAtoms are the same.

Two ArrayAtoms are judged to be equal if they have the same number of Atoms contained within them and each Atom in one array is equal to the corresponding Atom in the other array.

---

## Notes

None.

---

## Example





# GetEnumerator Function

---

Gets an enumerator for the Collection.

## [C#]

```
IEnumerator<Atom> GetEnumerator()
```

---

## Syntax

## [Visual Basic]

```
Function GetEnumerator() As  
IEnumerator(Of Atom)
```

---

## Params

Name	Description
return	The enumerator for the collection.

---

## Notes

Gets an Atom enumerator for the Collection.

---

## Example

None.

# GetRange Function

---



Creates a shallow copy of a range of elements in the source array

## [C#]

```
ArrayAtom GetRange(int index, int count)
```

## [Visual Basic]

Syntax

```
Function GetRange(index As Integer, count As Integer) As ArrayAtom
```

- may throw Exception()

Params

Name	Description
index	The zero-based index specifying the first element.
count	The number of elements to be selected.

Creates a shallow copy of a range of elements in the source array.

---

**Notes**

If the index is equal to the **Count** then the elements are added to the end of the array. If the index or count is invalid then an exception will be raised.

---

**Example**

None.

# InsertRange Function

---



Inserts the elements in the supplied array into this array at the specified index

## [C#]

```
void InsertRange(int index,  
ArrayAtom array)
```

## [Visual Basic]

### Syntax

```
Sub InsertRange(index As Integer,  
array As ArrayAtom)
```

- may throw `Exception()`

### Params

Name	Description
index	The zero-based index at which the new elements should be inserted.
array	The array whose elements should be inserted.

Inserts the elements in the supplied array into this array at the specified index.

If the index is equal to the `Count` then the

---

**Notes**

elements are added to the end of the array.

If the supplied array is null or the index is invalid then an exception will be raised.

---

**Example**

None.

# RemoveRange Function

---



Removes a range of elements from the source array

## [C#]

```
void RemoveRange(int index, int count)
```

## [Visual Basic]

### Syntax

```
Sub RemoveRange(index As Integer, count As Integer)
```

- may throw `Exception()`

### Params

Name	Description
index	The zero-based index specifying the first element.
count	The number of elements to be removed.

Removes a range of elements from the source array.

If the index is equal to the **Count** then the

---

**Notes**

elements are added to the end of the array.

If the index or count is invalid then an exception will be raised.

---

**Example**

None.

# Count Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The number of Atoms in the array.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The number of Atoms in the array.

---

## Notes

As Atoms are added to the array the Count will increase.

---

## Example

None.

# Item Property

---

Type	Default	Read Only	Description
<b>[C#]</b> <code>Atom this[int index]</code> <code>int this[int index, int def]</code> <code>double this[int index, double def]</code> <code>string this[int index, string def]</code>			
 <b>[Visual Basic]</b> Default Property <code>Item(index As Integer) As Atom</code> Default Property <code>Item(index As Integer, def As Integer) As Integer</code> Default Property <code>Item(index As Integer, def As Double) As Double</code> Default Property <code>Item(index As Integer, def As String) As String</code>	n/a	No	Get or set the Atom at the specified index.

- may throw `ArgumentOutOfRangeException()`

Gets or sets the Atom at the specified index. In C# this property is the indexer for the class.

You can access an Atom directly or you can use one of the overloaded operators to specify numbers or strings. Using the overloads to access numbers or strings is more efficient than accessing an Atom and extracting the value from it.

You specify one of the overloads using the `def` parameter. If you are setting a value then this parameter is ignored. If you are getting a value then this parameter becomes the default value to be used if the underlying Atom was not the correct type. For example the default would be returned if you attempted to get an integer but the underlying Atom was actually a [StringAtom](#).

---

Notes

Atoms can exist in only one place at a time. If the Atom supplied is already contained by another object then a [Clone](#) of the Atom is added.

Adding a null value will result in a [NullAtom](#) being added to the array.

If the index is not a valid index this property throws an `ArgumentOutOfRangeException`.

---

None.

**Example**



# BoolAtom Constructor

---

Construct a BoolAtom.

## [C#]

```
BoolAtom()  
BoolAtom(bool value)
```

---

### Syntax

## [Visual Basic]

```
Sub New()  
Sub New(value As Boolean)
```

---

### Params

Name	Description
value	The initial value that the Atom should adopt.

Create a BoolAtom.

---

### Notes

If a value is not specified the default of false will be used.

---

### Example

None.





# Equals Function

---

Test whether the two BoolAtoms are the same

## [C#]

```
bool Equals(BoolAtom other)
```

---

### Syntax

## [Visual Basic]

```
Function Equals(other As  
BoolAtom) As Boolean
```

---

### Params

Name	Description
other	The BoolAtom to test against.

Test whether the two BoolAtoms are the same.

---

### Notes

Two BoolAtoms are judged to be equal if their **Truth** is equal.

---

### Example

None.



# Truth Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	n/a	No	Whether the Boolean is true or false.

Whether the Boolean is true or false.

---

Notes

None.

---

Example



# DictAtom Constructor

---

Construct a DictAtom.

**[C#]**

```
DictAtom()
```

---

Syntax

**[Visual Basic]**

```
Sub New()
```

---

Params

Name	Description
none	

---

Notes

Create an empty DictAtom.

---

Example

None.



## CopyTo Function

---

Copies the Atoms into an array.

### [C#]

```
void CopyTo(Atom[] array, int index)
void CopyTo(KeyValuePair<string, Atom>[] array, int index)
```

---

### Syntax

### [Visual Basic]

```
Sub CopyTo(array As Atom(), index As Integer)
Sub CopyTo(array As KeyValuePair(Of String, Atom)(), index As Integer)
```

---

### Params

Name	Description
array	The array that is the destination for the elements.
index	The zero-based index in array at which copying begins.

Copies the elements of the dictionary to an array starting at a particular array index.

The array must be one-dimensional and have zero-based indexing.

---

**Notes**

The implementation of `ICollection.CopyTo` copies `KeyValuePair<string, Atom>` values to the array if the element type of the array is compatible. Otherwise, it copies `DictionaryEntry` values to the array.

---

**Example**

None.



# Add Function

---

Add an item to the dictionary.

## [C#]

```
void Add(string key, Atom atm)
void Add(string key, int num)
void Add(string key, double real)
void Add(string key, string str)
```

## [Visual Basic]

```
Sub Add(key As String, atm As
Atom)
Sub Add(key As String, num As
Integer)
Sub Add(key As String, real As
Double)
Sub Add(key As String, str As
String)
```

Syntax

- may throw `ArgumentException()`

- may throw `ArgumentNullException()`

Name	Description
key	The string to use as the key of the element to add.

---

**Params**

atm	The Atom to be added.
num	The integer to be added.
real	The floating point value to be added.
str	The string to be added.

This method adds an item to the dictionary.

You can add an Atom directly into the dictionary or you can use one of the overloaded operators to add numbers or strings.

When you add a string this is encapsulated within a [StringAtom](#) and then inserted. When you add an integer or floating point value this is converted to a [NumAtom](#) and then inserted. These operations are more efficient than creating an Atom and adding it yourself.

---

**Notes**

Atoms can exist in only one place at a time. If the Atom supplied is already contained by another object then a [Clone](#) of the Atom is added.

Adding a null value will result in a [NullAtom](#) being added to the dictionary.

If the key is null then this method will throw an `ArgumentNullException`. If the key is already present in the dictionary then this method will throw an `ArgumentException`.

None.

---

**Example**



# Clear Function

---

Removes all elements from the dictionary.

## [C#]

```
void Clear()
```

---

### Syntax

## [Visual Basic]

```
Sub Clear()
```

---

### Params

Name	Description
none	

---

### Notes

Removes all elements from the dictionary.

---

### Example

None.

# Contains Function

---



Determines whether the dictionary contains an element with a specific name.

## [C#]

```
bool Contains(string name)
```

---

## Syntax

## [Visual Basic]

```
Function Contains(name As String)  
As Boolean
```

---

## Params

Name	Description
name	The element to locate in the dictionary.
return	True if the name is found, otherwise false.

---

## Notes

Determines whether the dictionary contains an element with a specific name.

---

## Example

None.





# Remove Function

---

Remove an element from the dictionary.

## [C#]

```
void Remove(string name)
```

---

### Syntax

## [Visual Basic]

```
Sub Remove(name As String)
```

---

### Params

Name	Description
name	The name of the element to be removed.

---

### Notes

Remove the element with the specified name from the dictionary.

---

### Example

None.



# GetKeys Function

---

Get an array of all the names in the dictionary.

## [C#]

```
string[] GetKeys()
```

---

### Syntax

## [Visual Basic]

```
Function GetKeys() As String()
```

---

### Params

Name	Description
return	An array of all the names in the dictionary.

---

### Notes

Get an array of all the names in the dictionary.

---

### Example

None.



# GetValues Function

---

Get an array of all the Atoms in the dictionary.

## [C#]

```
Atom[] GetValues()
```

---

### Syntax

## [Visual Basic]

```
Function GetValues() As Atom()
```

---

### Params

Name	Description
return	An array of all the Atoms in the dictionary.

---

### Notes

Get an array of all the Atoms in the dictionary.

---

### Example

None.



# GetEnumerator Function

---

Get an enumerator for the dictionary.

## [C#]

```
DictAtom.Enumerator  
GetEnumerator()
```

---

## Syntax

## [Visual Basic]

```
Function GetEnumerator() As  
DictAtom.Enumerator
```

---

## Params

Name	Description
return	The enumerator for the dictionary.

Get an `KeyValuePair<string, Atom>` enumerator for the dictionary.

---

## Notes

`DictAtom.Enumerator` is a value type that implements `IEnumerator<KeyValuePair<string, Atom>>` and `IDictionaryEnumerator`.

None.

---

**Example**



# Equals Function

---

Test whether the two DictAtoms are the same

## [C#]

```
bool Equals(DictAtom other)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As  
DictAtom) As Boolean
```

---

## Params

Name	Description
other	The DictAtom to test against.

Test whether the two DictAtoms are the same.

Two DictAtoms are judged to be equal if they have the same named entries and the Atom corresponding with each named entry is equal to the corresponding Atom in the other dictionary.

---

## Notes

---

None.

## Example

# Count Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			Get the number of elements in the dictionary.
[Visual Basic] <code>Integer</code>	n/a	Yes	

Get the number of elements in the dictionary.

---

## Notes

As elements are added to the dictionary the Count will increase.

---

## Example

None.

# Item Property

---



Type	Default	Read Only	Description
<b>[C#]</b> <code>Atom this[string name]</code> <code>int this[string name, int def]</code> <code>double this[string name, double def]</code> <code>string this[string name, string def]</code>			
<b>[Visual Basic]</b> <code>Default Property Item(name As String) As Atom</code> <code>Default Property Item(name As String, def As Integer) As Integer</code> <code>Default Property Item(name As String, def As Double) As Double</code> <code>Default Property Item(name As String, def As String) As String</code>	n/a	Yes	Get or set the entry with the specified name.

- may throw `ArgumentNullException()`

Get or set the Atom with the specified name. In C# this property is the indexer for the class.

You can access an Atom directly or you can use one of the overloaded operators to specify numbers or strings. Using the overloads to access numbers or strings is more efficient than accessing an Atom and extracting the value from it.

You specify one of the overloads using the `def` parameter. If you are setting a value then this parameter is ignored. If you are getting a value then this parameter becomes the default value to be used if the underlying Atom was not the correct type. For example the default would be returned if you attempted to get an integer but the underlying Atom was actually a [StringAtom](#).

---

#### Notes

Atoms can exist in only one place at a time. If the Atom supplied is already contained by another object then a [Clone](#) of the Atom is added.

Adding a null value will result in a [NullAtom](#) being added to the array.

If the name is null this property throws an `ArgumentNullException`.

None.

---

#### Example



# Keys Property

---



Type	Default	Read Only	Description
[C#] DictAtom.KeyCollection	n/a	Yes	Get the collection of the keys in the dictionary.
[Visual Basic] DictAtom.KeyCollection			

Get the collection of the keys in the dictionary.

---

## Notes

DictAtom.KeyCollection implements ICollection<string> and ICollection.

---

## Example

None.

# Values Property

---



Type	Default	Read Only	Description
[C#] <code>DictAtom.ValueCollection</code>	n/a	Yes	Get the collection of the Atoms in the dictionary.
[Visual Basic] <code>DictAtom.ValueCollection</code>			

Get the collection of the Atoms in the dictionary.

---

## Notes

`DictAtom.ValueCollection` implements `ICollection<Atom>` and `ICollection`.

---

## Example

None.



# Equals Function

---

Test whether the two NameAtoms are the same

## [C#]

```
bool Equals(NameAtom other)
```

---

### Syntax

## [Visual Basic]

```
Function Equals(other As  
NameAtom) As Boolean
```

---

### Params

Name	Description
other	The NameAtom to test against.

Test whether the two NameAtoms are the same.

---

### Notes

Two NameAtoms are judged to be equal if their `Text` is equal.

---

### Example

None.





# NameAtom Constructor

---

Construct a NameAtom.

## [C#]

```
NameAtom()  
NameAtom(string value)
```

---

### Syntax

## [Visual Basic]

```
Sub New()  
Sub New(value As String)
```

---

### Params

Name	Description
value	The initial value that the Atom should adopt.

Create a NameAtom.

---

### Notes

If a value is not specified the default - an empty string - will be used.

None.

---

### Example



# Text Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	No	The text of the name.

The text of the name.

---

Notes

None.

---

Example



# Equals Function

---

Test whether the two NullAtoms are the same

## [C#]

```
bool Equals(NullAtom other)
```

---

### Syntax

## [Visual Basic]

```
Function Equals(other As  
NullAtom) As Boolean
```

---

### Params

Name	Description
other	The NullAtom to test against.

Test whether the two NullAtoms are the same.

---

### Notes

NullAtoms are are always equal to each other.

---

### Example

None.





# NullAtom Constructor

---

Construct a NullAtom.

**[C#]**

```
NullAtom()
```

---

Syntax

**[Visual Basic]**

```
Sub New()
```

---

Params

Name	Description
none	

---

Notes

Create a NullAtom.

---

Example

None.

# Encode Function

---



Encode a number into a PDF string. This format may be needed for direct insertion into a content stream.

## [C#]

```
static string Encode(double  
value)
```

---

## Syntax

## [Visual Basic]

```
Shared Function Encode(value As  
Double) As String
```

---

## Params

Name	Description
value	The number to convert.
return	The string representation.

Encode a number into a PDF string. This format may be needed for direct insertion into a content stream.

PDF content streams accept certain formats of numbers. In many situations these are the same as the strings returned by .NET functions such as ToString. However in the case of floating

point conversions, not all output formats are valid in a PDF content stream.

---

**Notes**

For example, very small or very large numbers can lead to the insertion of exponential format strings which are not an accepted PDF format and can lead to errors. Because numbers typically occur in a non exponential range these errors can be infrequent and difficult to track down.

This method will produce PDF valid strings for use in content streams and similar applications.

---

**Example**

None.



# Equals Function

---

Test whether the two NumAtoms are the same

## [C#]

```
bool Equals(NumAtom other)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As NumAtom)  
As Boolean
```

---

## Params

Name	Description
other	The NumAtoms to test against.

Test whether the two NumAtoms are the same.

---

## Notes

Two NumAtoms are judged to be equal if their numeric value is equal.

---

## Example

None.





# NumAtom Constructor

---

Construct a NumAtom.

## [C#]

```
NumAtom()  
NumAtom(int num)  
NumAtom(double real)
```

---

### Syntax

## [Visual Basic]

```
Sub New()  
Sub New(num As Integer)  
Sub New(real As Double)
```

---

### Params

Name	Description
num	The initial integer value that the Atom should adopt.
real	The initial floating point value that the Atom should adopt.

Create a NumAtom.

---

### Notes

If a value is not specified the default of zero will be used.

---

**Example**

None.

# Num Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The integer value of the number.
[Visual Basic] <code>Integer</code>	n/a	No	

The integer value of the number.

---

Notes

None.

---

Example

# Num64 Property

---



Type	Default	Read Only	Description
[C#] long			The 64-bit integer value of the number.
[Visual Basic] Long	n/a	No	

The 64-bit integer value of the number.

---

## Notes

None.

---

## Example

# Real Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	No	The floating point value of the number.

The floating point value of the number.

---

Notes

None.

---

Example



# OpAtom Function

---

Create an operator atom for a string

## [C#]

```
OpAtom()  
OpAtom(string value)
```

---

### Syntax

## [Visual Basic]

```
New()  
New(value As String)
```

---

### Params

Name	Description
value	The operator text.

---

### Notes

Create an operator atom for a string.

---

### Example

None.



# Equals Function

---

Test whether the two OpAtoms are the same

## [C#]

```
bool Equals(OpAtom other)
```

---

## Syntax

## [Visual Basic]

```
Function Equals(other As OpAtom)  
As Boolean
```

---

## Params

Name	Description
other	The OpAtom to test against.

Test whether the two OpAtoms are the same.

---

## Notes

Two OpAtoms are judged to be equal if their **Text** is equal.

---

## Example

None.





## Find Function

---

Finds specified types of OpAtom entries in an array

### [C#]

```
IList<Tuple<string, int>>  
Find(ArrayAtom atom)  
IList<Tuple<string, int>>  
Find(ArrayAtom atom, string[] names)
```

---

### Syntax

### [Visual Basic]

```
Function Find(atom As ArrayAtom) As  
IList(Of Tuple(Of string, int))  
Function Find(atom As ArrayAtom,  
names() As String) As IList(Of  
Tuple(Of string, int))
```

---

### Params

Name	Description
atom	The ArrayAtom to be searched.
names	The types of OpAtoms that should be returned.
return	The positions and text of the items which were found.

Finds specified types of OpAtom entries in an array.

---

**Notes**

If you specify no names then the positions and text of all the OpAtoms in the ArrayAtom will be returned. If you are only interested in a particular set of operators then you can specify an array of those types to restrict the number of items returned.

This example shows how to use the [Array.FromContentStream](#) function to parse a content stream and then use the Find method to search for certain types of color operators and replace them with other color operators. In this example this has the effect of converting some black parts of the PDF to red.

**[C#]**

```
using (Doc doc = new Doc()) {
    doc.Read("spaceshuttle.pdf");
    doc.RemapPages(new int[] { 1, 1 });
    doc.PageNumber = 2;
    Page page = doc.ObjectSoup[doc.Page] as
Page;
    StreamObject[] layers =
page.GetLayers();
    MemoryStream st = new MemoryStream();
    foreach (StreamObject layer in layers) {
        if (!layer.Decompress())
            throw new Exception("Unable to
decompress stream.");
        byte[] data = layer.GetData();
        st.Write(data, 0, data.Length);
        layer.CompressFlate();
    }
    ArrayAtom array =
ArrayAtom.FromContentStream(st.ToArray());
    if (true) {
```

```

        var items = OpAtom.Find(array, new
string[] { "k" });
        foreach (var pair in items) { // make
red
            Atom[] args =
OpAtom.GetParameters(array, pair.Item2);
            if (args != null) {
                ((NumAtom)args[0]).Real = 0;
                ((NumAtom)args[1]).Real = 1;
                ((NumAtom)args[2]).Real = 1;
                ((NumAtom)args[3]).Real = 0;
            }
        }
    }
    if (true) {
        var items = OpAtom.Find(array, new
string[] { "rg" });
        foreach (var pair in items) { // make
green
            Atom[] args =
OpAtom.GetParameters(array, pair.Item2);
            if (args != null) {
                ((NumAtom)args[0]).Real = 0;
                ((NumAtom)args[1]).Real = 1;
                ((NumAtom)args[2]).Real = 0;
            }
        }
    }
    byte[] arrayData = array.GetData();
    StreamObject so = new
StreamObject(doc.ObjectSoup);
    so.SetData(arrayData, 1,
arrayData.Length - 2);
    doc.SetInfo(page.ID, "/Contents:Del",
"");
    page.AddLayer(so);
    doc.Save("ReplaceColors.pdf");

```

```
}
```

## [Visual Basic]

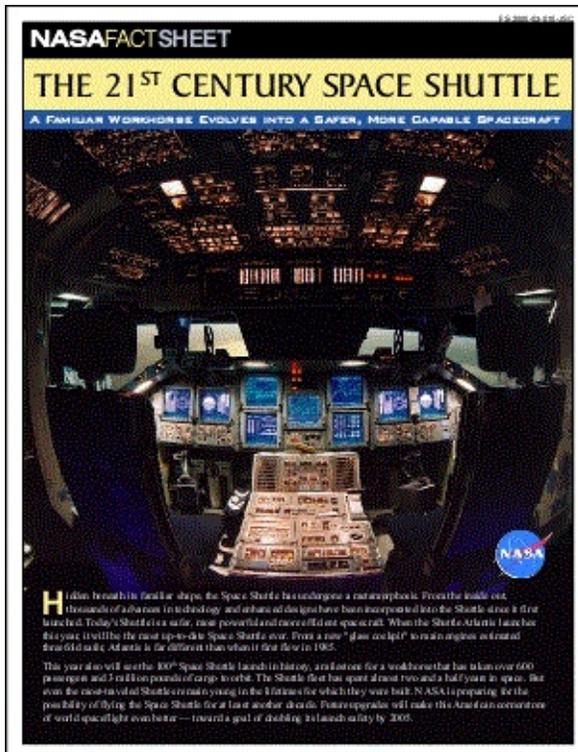
```
Sub ...
    Using doc As New Doc()
        doc.RemapPages(New Integer[] { 1, 1 })
        doc.PageNumber = 2
        Dim page As Page =
TryCast(doc.ObjectSoup(doc.Page), Page)
        Dim layers As StreamObject() =
page.GetLayers()
        Dim st As New MemoryStream()
        For Each layer As StreamObject In
layers
            If Not layer.Decompress() Then
                Throw New Exception("Unable to
decompress stream.")
            End If
            Dim data As Byte() = layer.GetData()
            st.Write(data, 0, data.Length)
            layer.CompressFlate()
        Next
        Dim array As ArrayAtom =
ArrayAtom.FromContentStream(st.ToArray())
        If True Then
            Dim items = OpAtom.Find(array, New
String() {"k"})
            For Each pair As var In items
                ' make red
                Dim args As Atom() =
OpAtom.GetParameters(array, pair.Item2)
                If args IsNot Nothing Then
                    DirectCast(args(0),
NumAtom).Real = 0
                    DirectCast(args(1),
NumAtom).Real = 1
```

**Example**

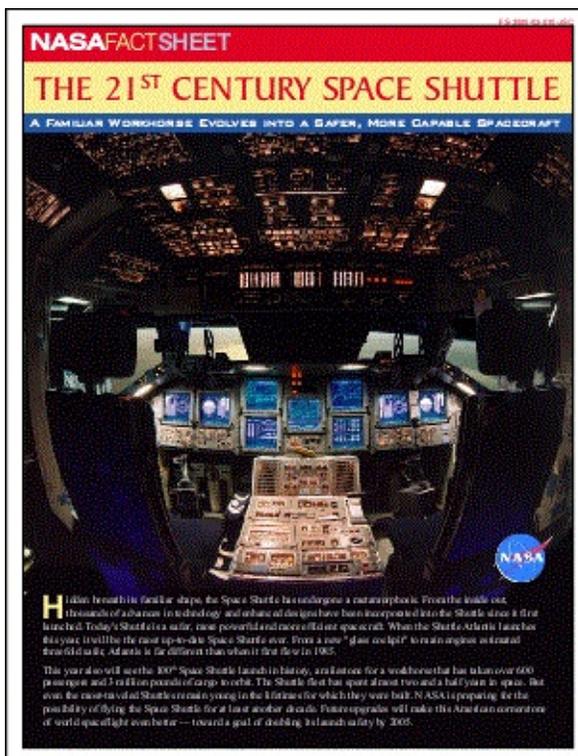
```

        DirectCast(args(2),
NumAtom).Real = 1
        DirectCast(args(3),
NumAtom).Real = 0
    End If
Next
End If
If True Then
    Dim items = OpAtom.Find(array, New
String() {"rg"})
    For Each pair As var In items
        ' make green
        Dim args As Atom() =
OpAtom.GetParameters(array, pair.Item2)
        If args IsNot Nothing Then
            DirectCast(args(0),
NumAtom).Real = 0
            DirectCast(args(1),
NumAtom).Real = 1
            DirectCast(args(2),
NumAtom).Real = 0
        End If
    Next
End If
    Dim arrayData As Byte() =
array.GetData()
    Dim so As New
StreamObject(doc.ObjectSoup)
    so.SetData(arrayData, 1,
arrayData.Length - 2)
    doc.SetInfo(page.ID, "/Contents:Del",
"")
    page.AddLayer(so)
    doc.Save("ReplaceColors.pdf")
End Using
End Sub

```



ReplaceColors.pdf [Page 1]



ReplaceColors.pdf [Page 2]



# GetParameter Function

---



Gets the parameter associated with the OpAtom at the specified index and validates that the atom is of the correct type

## [C#]

```
Atom GetParameter(ArrayAtom  
array, int index)
```

## [Visual Basic]

Syntax

```
Function GetParameter(array As  
ArrayAtom, index As Integer) As  
Atom
```

- may throw Exception()

Params

Name	Description
array	The array in which the OpAtom is found.
index	The index to the OpAtom.
return	The parameter.

Gets the parameter associated with the OpAtom at the specified index and validates that the

atom is of the correct type.

Only the top level atoms are checked. So for example, the TJ operator takes an ArrayAtom consisting of NumAtoms and StringAtoms. This function will ensure that there is one ArrayAtom available as a parameter. However it will not ensure that all items inside the array are NumAtoms or StringAtoms.

If the atom is not of the correct type, is not available in the array (e.g. because the array is too small) or the operator is unrecognized then the return value will be null. If the operator does not take any parameters the return will be null.

---

**Notes**

The Atom is the actual object in the supplied array so changing the value of the Atom will change the value in the array. If the index does not point to an OpAtom or is not inside the array bounds then an exception will be thrown.

This function supports all the standard PDF operator types as defined in the PDF Specification apart from BI, ID and EI which are used for inline images. The reason these are not supported is because this sequence is read as a DictAtom with embedded data so you should never encounter it.

None.

---

**Example**

# GetParameters Function

---



Gets the parameters associated with the OpAtom at the specified index and validates that the atoms are of the correct type

## [C#]

```
Atom[] GetParameters(ArrayAtom  
array, int index)
```

---

## Syntax

## [Visual Basic]

```
Function GetParameters(array As  
ArrayAtom, index As Integer) As  
Atom()
```

---

## Params

Name	Description
array	The array in which the OpAtom is found.
index	The index to the OpAtom.
return	The parameters.

Gets the parameters associated with the OpAtom at the specified index and validates that the atoms are of the correct type.

Because the arguments are validated and the

types are checked so you can rely on the fact that the returned array contains a valid number of Atoms of the correct types.

Only the top level atoms are checked. So for example, the TJ operator takes an ArrayAtom consisting of NumAtoms and StringAtoms. This function will ensure that there is one ArrayAtom available as a parameter. However it will not ensure that all items inside the array are NumAtoms or StringAtoms. Also resource lookups are not done so, for example, it is not checked that a font name actually exists in the font resource dictionary.

---

**Notes**

If the operator takes no parameters a zero length array will be returned. If the operator parameters are incorrect or if the operator is not recognized then the return value will be null. In this unusual situation you may wish to set the [OpAtom.Text](#) to white space to remove the invalid operator from the content stream. While this does not fix the underlying problem, it will at least prevent applications like Acrobat from reporting an error.

None.

---

**Example**

# Text Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	n/a	No	The text of the operator

The text of the operator.

---

## Notes

Typical operators include "rg" for setting a RGB color and "cm" for concatenating a matrix. Further details may be found in the PDF Specification.

---

## Example

None.



# RefAtom Constructor

---

Construct a RefAtom.

## [C#]

```
RefAtom()  
RefAtom(IndirectObject value)
```

---

### Syntax

## [Visual Basic]

```
Sub New()  
Sub New(value As IndirectObject)
```

---

### Params

Name	Description
value	The initial IndirectObject that the Atom should point to.

Create a RefAtom.

---

### Notes

If an [IndirectObject](#) is not specified the default of the null object at entry zero will be used.

None.

---

### Example





# Assign Function

---

Set the `IndirectObject` that the reference should point to.

## [C#]

```
void Assign(IndirectObject value)
```

---

### Syntax

## [Visual Basic]

```
Sub Assign(value As IndirectObject)
```

---

### Params

Name	Description
value	The <code>IndirectObject</code> that the <code>RefAtom</code> should point to.

---

### Notes

Set the `IndirectObject` that the reference should point to.

---

### Example

None.





# Equals Function

---

Test whether the two RefAtoms are the same

## [C#]

```
bool Equals(RefAtom other)
```

---

### Syntax

## [Visual Basic]

```
Function Equals(other As RefAtom)  
As Boolean
```

---

### Params

Name	Description
other	The RefAtom to test against.

Test whether the two RefAtoms are the same.

---

### Notes

Two RefAtoms are judged to be equal if their **ID** and **Gen** properties are equal.

---

### Example

None.



# Resolve Function

---



Get the `IndirectObject` that the reference is pointing to.

## [C#]

```
IndirectObject Resolve(ObjectSoup  
objects)
```

---

### Syntax

## [Visual Basic]

```
Function Resolve(objects As  
ObjectSoup) as IndirectObject
```

---

### Params

Name	Description
objects	The collection of objects to be searched.
return	The matching <code>IndirectObject</code> .

This function resolves a `RefAtom` to the `IndirectObject` that it references.

If the `RefAtom` is referencing another `RefAtom` then the `Resolve` process will continue until an `Atom` which is not a `RefAtom` is found. If no item is found then `null` is returned.

---

### Notes

---

**Example**

None.

# ID Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	No	The ID of the referenced <code>IndirectObject</code> .

The ID of the referenced `IndirectObject`.

---

## Notes

None.

---

## Example

# Gen Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	No	The generation of the referenced <code>IndirectObject</code> .

The generation of the referenced [IndirectObject](#).

---

Notes

None.

---

Example

# Encode Function

---



Encode a string into a format for use in a content stream using a normal or simple font.

## [C#]

```
static StringBuilder  
Encode(string text,  
IDictionary<char, char> encoding,  
StringBuilder builder)
```

---

## Syntax

## [Visual Basic]

```
Shared Function Encode(text As  
String, encoding As  
IDictionary<char, char>, builder  
As StringBuilder) As  
StringBuilder
```

---

## Params

Name	Description
text	The text to be encoded.
encoding	The encoding to use - typically obtained from the <a href="#">FontObject.GetEncoding</a> method.
builder	The StringBuilder to which the encoded representation should be appended.
return	The updated StringBuilder.

Encode a string into a format for use in a content stream using a normal or simple font.

Typically you will obtain an appropriate encoding from the `FontObject.GetEncoding` method. This method supports eight bit characters only as this is what is supported by simple fonts. Most commonly you will want to use the Latin horizontal encoding.

---

**Notes**

You should not use a null or identity encoding as there are subtle differences between the Latin1 encoding used in PDF and the first 256 characters of Unicode. As such if you use a null or identity encoding then characters such as the Euro sign will appear incorrectly.

---

**Example**

None.

# EncodeDoubleByte Function

---



Encode a string into a format for use in a content stream using a composite font with a double byte CMap.

## [C#]

```
static StringBuilder  
EncodeDoubleByte(string text,  
IDictionary<char, char> encoding,  
StringBuilder builder)
```

---

## Syntax

## [Visual Basic]

```
Shared Function  
EncodeDoubleByte(text As String,  
encoding As IDictionary<char,  
char>, builder As StringBuilder)  
As StringBuilder
```

---

## Params

Name	Description
text	The text to be encoded.
encoding	The encoding to use - typically obtained from the <a href="#">FontObject.GetEncoding</a> method.
builder	The StringBuilder to which the encoded representation should be appended.
return	The updated StringBuilder.

Encode a string into a format for use in a content stream using a composite font with a double byte CMap.

Typically you will obtain an appropriate encoding from the `FontObject.GetEncoding` method. This method supports eight bit characters only as this is what is supported by simple fonts. Most commonly you will want to use a Korean, Japanese, ChineseS or ChineseT encoding in either horizontal or vertical format.

---

**Notes**

This method should only be used with composite fonts that specify this type of encoding as only these fonts will understand the fact that the characters are multi-byte. If you attempt to use this method with a simple font then you will get extra characters inserted since each double byte entry will be interpreted as two single characters.

See the [Fonts and Languages](#) section for details on language types.

---

**Example**

None.



# StringAtom Constructor

---

Construct a StringAtom.

## [C#]

```
StringAtom()  
StringAtom(string value)
```

---

### Syntax

## [Visual Basic]

```
Sub New()  
Sub New(value As String)
```

---

### Params

Name	Description
value	The initial value that the Atom should adopt.

Create a StringAtom.

---

### Notes

If a value is not specified the default - an empty string - will be used.

None.

---

### Example



# Decode Function

---



Decode a PDF encoded string into a plain string format

## [C#]

```
StringBuilder Decode(string text,  
StringBuilder builder)
```

---

## Syntax

## [Visual Basic]

```
Function Decode(text As String,  
builder As StringBuilder) As  
StringBuilder
```

---

## Params

Name	Description
text	The text to be decoded.
builder	The StringBuilder to which the encoded representation should be appended. If null then one will be created.
return	The StringBuilder supplied or created.

Decode a PDF encoded string into a plain string format.

This type of operation can be useful if you are

extracting text from a PDF content stream containing raw PDF string operators.

---

**Notes**

Typically you would use the `StringAtom Decode` or `DecodeDoubleByte` methods to allow text operator parameters to be decoded into the base text encoding. These can then be passed through the `FontObject EncodingToChar` and `EncodingToString` properties to allow mapping from the text encoding through to Unicode values.

---

**Example**

None.

# DecodeDoubleByte Function

---



Decode a double byte PDF encoded string into a plain string format

## [C#]

```
StringBuilder  
DecodeDoubleByte(string text,  
StringBuilder builder)
```

---

## Syntax

## [Visual Basic]

```
Function DecodeDoubleByte(text As  
String, builder As StringBuilder)  
As StringBuilder
```

---

## Params

Name	Description
text	The text to be decoded.
builder	The StringBuilder to which the encoded representation should be appended. If null then one will be created.
return	The StringBuilder supplied or created.

Decode a double byte PDF encoded string into a plain string format.

This may be necessary if you are looking at a content stream using a Composite font with a double byte CMap.

This type of operation can be useful if you are extracting text from a PDF content stream containing raw PDF string operators.

---

**Notes**

Typically you would use the StringAtom [Decode](#) or [DecodeDoubleByte](#) methods to allow text operator parameters to be decoded into the base text encoding. These can then be passed through the FontObject [EncodingToChar](#) and [EncodingToString](#) properties to allow mapping from the text encoding through to Unicode values.

None.

---

**Example**



# Equals Function

---

Test whether the two StringAtoms are the same

## [C#]

```
bool Equals(StringAtom other)
```

---

### Syntax

## [Visual Basic]

```
Function Equals(other As  
StringAtom) As Boolean
```

---

### Params

Name	Description
other	The StringAtom to test against.

Test whether the two StringAtoms are the same.

---

### Notes

Two StringAtoms are judged to be equal if their `Text` is equal.

None.

---

### Example



# Text Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	No	The text of the string.

---

## Notes

The text of the string.

---

## Example

None.

# ProcessingObject Event

---



Occurs just before an IndirectObject is processed.

## [C#]

```
event ProcessingObjectEventHandler  
ProcessingObject;  
  
delegate void  
ProcessingObjectEventHandler(object  
sender, ProcessingObjectEventArgs  
e);
```

---

## Syntax

## [Visual Basic]

```
Event ProcessingObject As  
ProcessingObjectEventHandler  
  
Delegate Sub  
ProcessingObjectEventHandler(sender  
As Object, e As  
ProcessingObjectEventArgs);
```

---

## Params

Name	Description
sender	The Operation firing the event.
e	The <a href="#">ProcessingObjectEventArgs</a> used to control the operation.

Typically you handle the `ProcessingObject` event in order to determine properties about an [IndirectObject](#) before processing.

Because the handler passes an `EventArgs` class which inherits from `CancelEventArgs` you can also choose to cancel the operation on the `IndirectObject`.

---

#### Notes

To associate the event with your event handle add an instance of the `ProcessingObjectEventHandler` delegate to the event. The event handler will be called whenever the event occurs.

See the [RecolorOperation.Recolor](#) method.

---

#### Example

# ProcessedObject Event

---



Occurs just after an IndirectObject has been processed.

## [C#]

```
event ProcessedObjectEventHandler  
ProcessedObject;  
  
delegate void  
ProcessedObjectEventHandler(object  
sender, ProcessedObjectEventArgs  
e);
```

---

## Syntax

## [Visual Basic]

```
Event ProcessedObject As  
ProcessedObjectEventHandler  
  
Delegate Sub  
ProcessedObjectEventHandler(sender  
As Object, e As  
ProcessedObjectEventArgs);
```

---

## Params

Name	Description
sender	The Operation firing the event.
e	The <a href="#">ProcessedObjectEventArgs</a> used to report back the status of the operation.

Typically you handle the `ProcessedObject` event in order to post-process an [IndirectObject](#).

---

**Notes**

To associate the event with your event handle add an instance of the `ProcessedObjectEventHandler` delegate to the event. The event handler will be called whenever the event occurs.

---

**Example**

See the [RecolorOperation.Recolor](#) method.



# ProcessedObjectEventArgs Constructor

---

ProcessedObjectEventArgs Constructor.

## [C#]

```
ProcessedObjectEventArgs(IndirectObject  
obj, bool successful)
```

---

## Syntax

## [Visual Basic]

```
Sub New(obj As IndirectObject,  
successful As Boolean)
```

---

## Params

Name	Description
obj	The IndirectObject processed.
successful	A value indicating whether the processing was successful.

---

## Notes

Create a ProcessedObjectEventArgs detailing an IndirectObject and whether it was processed successfully or not.

---

**Example**

None.

# Object Property

---



Type	Default	Read Only	Description
[C#]IndirectObject			
[Visual Basic]IndirectObject	n/a	Yes	Gets the IndirectObject which has just been processed.

The IndirectObject which has just been processed.

---

Notes

See the `RecolorOperation.Recolor` method.

---

Example

# Successful Property

---



Type	Default	Read Only	Description
[C#]bool			Gets a value indicating whether the processing was successful.
[Visual Basic]Boolean	n/a	Yes	Gets a value indicating whether the processing was successful.

Gets a value indicating whether the processing was successful.

---

## Notes

See the [RecolorOperation.Recolor](#) method.

---

## Example

# Tag Property

---



Type	Default	Read Only	Description
[C#]object			
[Visual Basic] Object	n/a	No	Gets or sets an object which can be used to save data about the event.

Gets or sets an object which can be used to save data about the event.

---

## Notes

This can be useful if you need to pass information from a [ProcessingObject](#) event to a [ProcessedObject](#) event.

---

## Example

See the [RecolorOperation.Recolor](#) method.



# ProcessingObjectEventArgs Constructor

---

ProcessingObjectEventArgs Constructor.

## [C#]

```
ProcessingObjectEventArgs(IndirectObject  
obj)  
ProcessingObjectEventArgs(IndirectObject  
obj, ProcessingInfo info)
```

---

## Syntax

## [Visual Basic]

```
Sub New(obj As IndirectObject)  
Sub New(obj As IndirectObject, info As  
ProcessingInfo)
```

---

## Params

Name	Description
obj	The IndirectObject which is about to be processed.
info	The ProcessingInfo containing related information.

Create a ProcessingObjectEventArgs detailing an IndirectObject which is about to be processed.

---

## Notes

Example None.

# Object Property

---



Type	Default	Read Only	Description
[C#]IndirectObject			
[Visual Basic]IndirectObject	n/a	Yes	Gets the IndirectObject to be processed.

The IndirectObject to be processed.

---

Notes

See the [RecolorOperation.Recolor](#) method.

---

Example

# Tag Property

---



Type	Default	Read Only	Description
[C#]object			
[Visual Basic]Object	n/a	No	Gets or sets an object which can be used to save data about the event.

Gets or sets an object which can be used to save data about the event.

---

## Notes

This can be useful if you need to pass information from a [ProcessingObject](#) event to a [ProcessedObject](#) event.

---

## Example

See the [RecolorOperation.Recolor](#) method.

# Info Property

---



Type	Default	Read Only	Description
[C#]ProcessingInfo			
[Visual Basic]ProcessingInfo	n/a	Yes	Gets the ProcessingInfo containing related information.

The ProcessingInfo containing related information.

---

Notes

See the [XpsImportOperation.Import](#) and the [SwfImportOperation.Import](#) methods.

---

Example



# ProcessingInfo Constructor

---

ProcessingInfo Constructor.

## [C#]

```
ProcessingInfo()
```

---

## Syntax

## [Visual Basic]

```
Sub New()
```

---

## Params

Name	Description
none	

---

## Notes

Create a ProcessingInfo.

---

## Example

None.

# SourceType Property

---



Type	Default	Read Only	Description
[C#]ProcessingSourceType [Visual Basic] ProcessingSourceType	See description.	Yes	Gets the type of the source object to be processed and the stage of operation.

The ProcessingSourceType enumeration may take the following values:

- Unknown
- Document
- Page
- PageContent
- Image
- ImageMask
- MultiFrameImage
- ImageFrame
- Stream
- Path
- Text
- FormXObject
- Shading

---

## Notes

The meaning of these source types depends on the

operations.

---

**Example**

None.

# Handled Property

---



Type	Default	Read Only	Description
[C#]bool [Visual Basic] Boolean	false	No	Gets or sets a value that indicates whether the event handler has handled the event so that the operation skips the default processing.

This property determines whether the operation skips the default processing.

---

## Notes

See the [XpsImportOperation.Import](#) method.

---

## Example

# X Property

---



Type	Default	Read Only	Description
[C#]double? [Visual Basic] Nullable(Of Double)	n/a	Yes	Gets the x-coordinate of the location of the source object.

This property holds the horizontal offset of the location.

---

## Notes

Values are in the coordinate system of the source object.

---

## Example

None.

# Y Property

---



Type	Default	Read Only	Description
[C#]double? [Visual Basic] Nullable(Of Double)	n/a	Yes	Gets the y-coordinate of the location of the source object.

This property holds the vertical offset of the location.

---

## Notes

Values are in the coordinate system of the source object.

---

## Example

None.

# Width Property

---



Type	Default	Read Only	Description
[C#]double?  [Visual Basic] Nullable(Of Double)	n/a	Yes	Gets the width of the source object.

This property holds the width of the source object.

---

## Notes

Values are in the coordinate system of the source object.

---

## Example

None.

# Height Property

---



Type	Default	Read Only	Description
[C#]double?  [Visual Basic] Nullable(Of Double)	n/a	Yes	Gets the height of the source object.

This property holds the height of the source object.

---

## Notes

Values are in the coordinate system of the source object.

---

## Example

None.

# DocNumber Property

---



Type	Default	Read Only	Description
<code>[C#]int?</code> <code>[Visual Basic] Nullable(Of Integer)</code>	n/a	No	Gets or sets the document number of the source document being/to be processed.

The value of this property when the [Operation.ProcessingObject](#) event is generated depends on the operation.

---

## Notes

The event handler should change this value when a processing sequence other than the default is desired. Set this value to null to end the processing of the current set of documents.

---

## Example

See the [XpsImportOperation.Import](#) method.

# DocCount Property

---



Type	Default	Read Only	Description
[C#]int? [Visual Basic] Nullable(Of Integer)	n/a	Yes	Gets the number of documents in the source object.

This property holds the number of pages in the document.

---

## Notes

None.

---

## Example

# PageNumber Property

---



Type	Default	Read Only	Description
[C#]int?  [Visual Basic] Nullable(Of Integer)	n/a	No	Gets or sets the page number of the source page being/to be processed.

The value of this property when the [Operation.ProcessingObject](#) event is generated depends on the operation.

---

## Notes

The event handler should change this value when a processing sequence other than the default is desired. Set this value to null to end the processing of the current set of pages.

---

## Example

See the [XpsImportOperation.Import](#) method.

# PageCount Property

---



Type	Default	Read Only	Description
[C#]int? [Visual Basic] Nullable(Of Integer)	n/a	Yes	Gets the number of pages in the source object.

This property holds the number of pages in the source object.

---

## Notes

None.

---

## Example

# FrameNumber Property

---



Type	Default	Read Only	Description
[C#]long?  [Visual Basic] Nullable(Of Long)	n/a	No	Gets or sets the frame number of the source frame being/to be processed.

The value of this property when the [Operation.ProcessingObject](#) is generated depends on the operation.

---

## Notes

The event handler should change this value when a processing object is generated other than the default is desired. Set this value to null to end the processing of the current set of frames.

See the [SwfImportOperation.Import](#) method.

Here we import the frame at 3.5 seconds.

## [C#]

```
Doc doc = new Doc();
using(SwfImportOperation operation = new SwfImportOperation())
{
```

Example

```
operation.Doc = doc;
operation.ContentAlign = ContentAlign.Top;
operation.ContentScaleMode = ContentScaleMode
operation.ProcessingObject += delegate(object
ProcessingObjectEventArgs e) {
    if(e.Info.SourceType==ProcessingSourceType.M
&& e.Info.FrameNumber.HasValue)
        e.Info.FrameNumber = 1+(long)
(e.Info.FrameRate.Value*3.5);
};
operation.Import(Server.MapPath("ABCpdf.swf"))
}
doc.Save(Server.MapPath("swf.pdf"));
doc.Clear();
```

### [Visual Basic]

```
Dim theDoc As Doc = New Doc()
Using theOperation As New SwfImportOperation
    theOperation.Doc = theDoc
    theOperation.ContentAlign = ContentAlign.Top
    theOperation.ContentScaleMode = ContentScaleM
    AddHandler theOperation.ProcessingObject, Add
Processing
    theOperation.Import(Server.MapPath("ABCpdf.sw
End Using
theDoc.Save(Server.MapPath("swf.pdf"))
theDoc.Clear()

Private Shared Sub Processing(sender As Object,
ProcessingObjectEventArgs)
    If e.Info.SourceType = ProcessingSourceType.M
    AndAlso e.Info.FrameNumber.HasValue Then
        e.Info.FrameNumber = 1 + CLng(e.Info.FrameR
3.5) End If
End Sub
```



# FrameCount Property

---



Type	Default	Read Only	Description
[C#]long? [Visual Basic] Nullable(Of Long)	n/a	Yes	Gets the number of frames in the source object.

This property holds the number of frames in the source object.

---

## Notes

None.

---

## Example

# FrameRate Property

---



Type	Default	Read Only	Description
[C#]double? [Visual Basic] Nullable(Of Double)	n/a	Yes	Gets the number of frames per second for the source object.

This property holds the number of frames per second for the source object.

---

## Notes

See the [FrameNumber](#) property and the [SwfImportOperation.Import](#) method.

---

## Example

# BackgroundColor Property

---



Type	Default	Read Only	Description
[C#]XColor			
[Visual Basic]XColor	n/a	Yes	Gets the background color of the source object.

This property holds the background color of the source object.

---

## Notes

See the [SwfImportOperation.Import](#) method.

---

## Example

# StreamPosition Property

---



Type	Default	Read Only	Description
[C#]long? [Visual Basic] Nullable(Of Long)	n/a	Yes	Gets or Sets the stream position of the source object, if applicable.

The value of this property when the [Operation.ProcessingObject](#) event is generated depends on the operation.

---

## Notes

This property holds the position in the PDF stream. Set this value to null to end the processing of the current stream.

See the [RenderOperation.Save](#) method.

---

## Example

None.

# StreamLength Property

---



Type	Default	Read Only	Description
[C#]long? [Visual Basic] Nullable(Of Long)	n/a	No	Gets the stream length of the source object, if applicable.

The value of this property when the [Operation.ProcessingObject](#) event is generated depends on the operation.

---

## Notes

See the [RenderOperation.Save](#) method.

None.

---

## Example

# Text Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	n/a	No	Gets the unicode Text of the Source Event, if applicable.

The value of this property when the [Operation.ProcessingObject](#) event is generated depends on the operation.

---

## Notes

See the [RenderOperation.Save](#) method.

None.

---

## Example



# RecolorOperation Constructor

---

RecolorOperation Constructor.

**[C#]**

```
RecolorOperation()
```

---

Syntax

**[Visual Basic]**

```
Sub New()
```

---

Params

Name	Description
none	

Create a RecolorOperation.

---

Notes

You must specify a [ColorSpace](#) before calling the [Recolor](#) method.

---

Example

See the [Recolor](#) method.





# Recolor Function

---

Recolor pages in a document.

## [C#]

```
void Recolor(Doc doc)
void Recolor(Pages pages)
void Recolor(Page page)
```

## [Visual Basic]

### Syntax

```
Sub Recolor(doc As Doc)
Sub Recolor(pages As Pages)
Sub Recolor(page As Page)
```

- may throw Exception()

### Params

Name	Description
doc	The document to be recolored.
pages	The pages to be recolored as referenced by a Page IndirectObject.
page	The page to be recolored as referenced by a Page IndirectObject.

Converts the specified pages from one color space to another.

The destination color space is specified by assigning a value to the [DestinationColorSpace](#) property. All images used on the page are converted to the new color space. All color operators used in the page content stream are converted to the new color space.

Annotations and fields are not part of the page but instead float over the page. You can choose whether to convert the appearance stream of any annotations or leave them in the native color space. This is controlled using the [ConvertAnnotations](#) property.

Colors can only be sensibly mapped from one color space to another if we know something about the characteristics of the color space. If your color spaces do not contain this information (e.g. if they are device color spaces) then ABCpdf will use a default color profile.

An exception will be thrown if the operation is not possible. This may happen if the IndirectObjects provided are not in an ObjectSoup or if the destination ColorSpace is in some way invalid.

---

Notes

As part of the Recolor process, all images used on the page are also recolored. (See also [PixMap.Recolor](#).) After the recolor process has been completed these [PixMap](#) objects will no longer be compressed. You may wish to analyse and recompress these images by pre and post processing them during the [ProcessingObject](#) and [ProcessedObject](#) events.

Although the Recolor operation is designed to convert a document to a single color space, such as CMYK, with actual colors unchanged, it can also be used to convert a document to a single, different, color. When you add a spot color (please see [AddColorSpaceSpot](#)) and make it the destination color space, the conversion will be to shades of the spot color specified.

Here we recolor one page out of a document. We pick up the [ProcessingObject](#) events so that we can store the source color of all the PixMap objects which are processed. We don't want CMYK pixmaps so we set the Cancel property to true if we

We then pick up the [ProcessedObject](#) events so that we can store the destination color of the PixMap objects after they have been recolored. We vary the recompression method used dependent on the source color and the size of the image.

Here we use standard delegates for backwards compatibility with 1.1 code. However .NET 2.0 anonymous delegates will provide a more compact solution.

### [C#]

```
Doc theDoc = new Doc();
theDoc.Read(Server.MapPath("../mypics/sample1.tif"));
MyOp.Recolor(theDoc,
(Page)theDoc.ObjectSoup[theDoc.Page]);
theDoc.Save(Server.MapPath("RecolorOperation.tif"));
theDoc.Clear();

private class MyOp{
    public static void Recolor(Doc doc, Page page,
        RecolorOperation op = new RecolorOperation(
            doc, page, op.DestinationColorSpace = new
            ColorSpace(doc.ObjectSoup,
            ColorSpaceType.DeviceGray);
            op.ConvertAnnotations = false;
            op.ProcessingObject += Recoloring;
            op.ProcessedObject += Recolored;
            op.Recolor(page);
        }

    public static void Recoloring(object sender, EventArgs e)
```

```

ProcessingObjectEventArgs e) {
    PixMap pm = e.Object as PixMap;
    if (pm != null) {
        ColorSpaceType cs = pm.ColorSpaceType
        if (cs == ColorSpaceType.DeviceCMYK)
            e.Cancel = true;
        e.Tag = cs;
    }
}

public static void Recolored(object sender,
ProcessedObjectEventArgs e) {
    if (e.Successful) {
        PixMap pm = e.Object as PixMap;
        if (pm != null) {
            ColorSpaceType cs = (ColorSpaceType)
            if (pm.Width > 1000)
                pm.CompressJpx(30);
            else if (cs == ColorSpaceType.DeviceCMYK)
                pm.CompressJpeg(30);
            else
                pm.Compress(); // Flate
        }
    }
}
}
}
}

```

---

#### Example

### [Visual Basic]

```

Dim theDoc As Doc = New Doc()
theDoc.Read(Server.MapPath("../mypics/sample1.doc"))
MyOp.Recolor(theDoc,
CType(theDoc.ObjectSoup(theDoc.Page), Page)
theDoc.Save(Server.MapPath("RecolorOperation.doc"))
theDoc.Clear()

```

```

Private Class MyOp

```

```

Public Shared Sub Recolor(doc As Doc, page
Page)
    Dim op As New RecolorOperation()
    op.DestinationColorSpace = New
ColorSpace(doc.ObjectSoup,
ColorSpaceType.DeviceGray)
    op.ConvertAnnotations = False
    AddHandler op.ProcessingObject, Address
Recoloring
    AddHandler op.ProcessedObject, AddressC
Recolored
    op.Recolor(page)
End Sub 'Recolor

```

```

Public Shared Sub Recoloring(sender As Obj
As ProcessingObjectEventArgs)
    Dim pm As PixMap = TryCast(e.Object, Pi
If pm IsNot Nothing Then
    Dim cs As ColorSpaceType = pm.ColorSp
    If cs = ColorSpaceType.DeviceCMYK The
        e.Cancel = True
    End If
    e.Tag = cs
End If
End Sub 'Recoloring

```

```

Public Shared Sub Recolored(sender As Obj
ProcessedObjectEventArgs)
    If e.Successful Then
        Dim pm As PixMap = TryCast(e.Object,
        If pm IsNot Nothing Then
            Dim cs As ColorSpaceType = CType(e.
ColorSpaceType)
            If pm.Width > 1000 Then
                pm.CompressJpx(30)
            Else
                If cs = ColorSpaceType.DeviceRGB

```

```
        pm.CompressJpeg(30)
    Else
        pm.Compress() ' Flate
    End If
End If
End If
End If
End Sub 'Recolored
End Class 'MyOp
```

# ConvertAnnotations Property

---



Type	Default	Read Only	Description
[C#]bool			Gets or sets a value indicating whether annotations are to be recolored.
[Visual Basic]Boolean	true	No	

Whether annotations should be recolored.

Annotations and fields are not part of the page but instead float over the page. You can choose whether to convert the appearance stream of any annotations or leave them in their native color space.

---

## Notes

See the [Recolor](#) method.

---

## Example

# DestinationColorSpace Property

---



Type	Default	Read Only	Description
[C#]ColorSpace			
[Visual Basic]ColorSpace	null	No	Gets or sets the destination ColorSpace.

The destination [ColorSpace](#) for the operation.

---

## Notes

See the [Recolor](#) method.

---

## Example



# GetApplicationFolder Function

---

Gets the installation folder of an import application.

## [C#]

```
static string  
GetApplicationFolder(ImportApplicationType  
app)
```

---

## Syntax

## [Visual Basic]

```
Shared Function GetApplicationFolder(app  
As ImportApplicationType) As String
```

---

## Params

Name	Description
app	The import <a href="#">application</a> .
return	The installation folder of the application.

This function returns the installation folder of an application that can be used by [ImportAny](#).

The installation folder of `ImportApplicationType.ShellAssociation` is null. For the Microsoft Office applications, it will be the Microsoft Office installation folder. If this function returns null for an application other than `ShellAssociation`, it means that the application is not

---

## Notes

installed or that ABCpdf has failed to find the application. If this is the case, `ImportAny` will fail. Before calling `ImportAny`, you can use this function to make sure that ABCpdf finds the application.

---

**Example**

None.



# XpsImportOperation Constructor

---

XpsImportOperation Constructor.

## [C#]

```
XpsImportOperation()
```

---

## Syntax

## [Visual Basic]

```
Sub New()
```

---

## Params

Name	Description
none	

---

## Notes

Create an XpsImportOperation.

---

## Example

See the [Import](#) method.

# ApplicationIsRunning Function

---



Gets a value indicating whether an import application is running for this XpsImportOperation.

## [C#]

```
bool  
ApplicationIsRunning(ImportApplicationType  
app)
```

---

## Syntax

## [Visual Basic]

```
Function ApplicationIsRunning(app As  
ImportApplicationType) As Boolean
```

Name	Description
app	The import <a href="#">application</a> .
return	Whether the application is running.

---

## Params

This function indicates whether an application is running for this XpsImportOperation when the process reuse for the application is enabled with [EnableApplicationReuse](#).

---

## Notes

None.  
Example

# EnableApplicationReuse Function

---



Enables an import application process to be reused so that it is terminated only manually or when this operation is disposed of.

## [C#]

```
bool  
EnableApplicationReuse(ImportApplicationType  
app, bool reuse)
```

---

## Syntax

## [Visual Basic]

```
Function EnableApplicationReuse(app As  
ImportApplicationType, reuse As Boolean) As  
Boolean
```

---

## Params

Name	Description
app	The import <a href="#">application</a> .
reuse	Whether the application process is to be reused.
return	Whether the reuse status of the application has been changed.

This function changes the [reuse status](#) of an application. Currently, the application processes of only Microsoft Word, Microsoft Excel, and Microsoft

PowerPoint can be reused.

Without application reuse, each call to `ImportAny` starts and terminates the application process. To avoid the overhead of the application start-up and exit, enable the reuse of the process for the particular application(s). The same process of that application is used in multiple calls to `ImportAny` of the same `XpsImportOperation` that uses the `application`.

---

Notes

The `reuse status` is not changed if the process reuse of the application is not supported. If the process is `running` (for this `XpsImportOperation` object), the running process is terminated when its reuse is disabled.

None.

---

Example



# Import Function

---

Imports a portion of an XPS document.

## [C#]

```
void Import(Doc doc, string path)
```

## [Visual Basic]

Syntax

```
Sub Import(doc As Doc, path As String)
```

- may throw `Exception()`

Params

Name	Description
doc	The target PDF document.
path	The file path to the XPS document.

Imports an XPS or OXPS Document. By default, the entire

An exception will be thrown if the operation is not possible.  
corrupt.

You may notice that colors in the PDF files are slightly different differently from other file formats. Refer to [SwfImportOperat](#) blending.

This method generates events as follows:

1. After the StartPart FixedDocumentSequence is opened ProcessingSourceType.Document is generated.

Name	Description
ProcessingObjectEventArgs.Cancel	See ProcessingObjectEventArgs below.
ProcessingObjectEventArgs.Info.Handled	Ignored
ProcessingObjectEventArgs.Info.SourceType	ProcessingObjectEventArgs.Info.SourceType
ProcessingObjectEventArgs.Info.DocCount	The number of FixedDocuments in this processing object.
ProcessingObjectEventArgs.Info.DocNumber	The one-based index of the FixedDocument in this processing object. The DocCount value returned by the previous event is between 1 and DocCount. Otherwise, the DocNumber value changes to 1, inclusive of all FixedDocuments. If the DocCount corresponds to the number of FixedDocuments generated, the DocNumber is not generated. If DocCount is not generated, the DocNumber is immediately incremented.

2. After a previous ProcessingObject event of ProcessingObjectEventArgs.DocNumber and the FixedDocument is opened, a ProcessingObjectEventArgs.Page ProcessingSourceType.Page is generated.

Name	Description



<a href="#">ProcessingObjectEventArgs.Cancel</a>	is not i <a href="#">Proces</a>
<a href="#">ProcessingObjectEventArgs.Info.Handled</a>	The va event i handle the tar crop b accorc and th entire the pa page u
<a href="#">ProcessingObjectEventArgs.Info.SourceType</a>	Proces
<a href="#">ProcessingObjectEventArgs.Info.DocCount</a>	The nu FixedE
<a href="#">ProcessingObjectEventArgs.Info.DocNumber</a>	The do being j
<a href="#">ProcessingObjectEventArgs.Info.PageCount</a>	The nu FixedE
<a href="#">ProcessingObjectEventArgs.Info.PageNumber</a>	The pa proces
<a href="#">ProcessingObjectEventArgs.Info.Width</a>	The wi the Fix PageC
<a href="#">ProcessingObjectEventArgs.Info.Height</a>	The he the Fix PageC

4. While processing page content, a [ProcessingObject](#) ev generated when an image is imported.

<b>Name</b>	<b>Descr</b>
<a href="#">ProcessingObjectEventArgs.Cancel</a>	Ignore
<a href="#">ProcessingObjectEventArgs.Info.Handled</a>	Ignore
<a href="#">ProcessingObjectEventArgs.Info.SourceType</a>	Proces

<code>ProcessingObjectEventArgs.Info.DocCount</code>	The number of pages in the document. FixedLength = true.
<code>ProcessingObjectEventArgs.Info.DocNumber</code>	The document number. The document being processed.
<code>ProcessingObjectEventArgs.Info.PageCount</code>	The number of pages in the document. FixedLength = true.
<code>ProcessingObjectEventArgs.Info.PageNumber</code>	The page number. The page being processed.
<code>ProcessingObjectEventArgs.Info.Width</code>	The page width.
<code>ProcessingObjectEventArgs.Info.Height</code>	The page height.

Unless stated otherwise, a `ProcessedObject` event corresponds to a `ProcessingObjectEventArgs` object as follows:

<code>ProcessingObjectEventArgs.Info.SourceType</code>	Process success
<code>ProcessingSourceType.Document</code>	null
<code>ProcessingSourceType.Page</code>	<code>GraphicsImage</code>
<code>ProcessingSourceType.PageContent</code>	null
<code>ProcessingSourceType.Image</code>	<code>PixelFormat</code>

Imported images are not compressed. You may wish to analyze objects by pre and post processing them during the `Process`.

Here we import the pages with odd page numbers of an XPS document. The aspect ratio is preserved.

### [C#]

```
Doc doc = new Doc();
MyImportOperation importOp = new MyImportOperation();
importOp.Import(Server.MapPath("AdvancedGraphics.pdf"));
doc.Save(Server.MapPath("xps.pdf"));
```

```

doc.Clear();

public class MyImportOperation {
    private Doc _doc = null;
    private double _margin = 10;
    private int _pagesAdded = 0;

    public MyImportOperation(Doc doc) {
        _doc = doc;

        _doc.Transform.Rotate(90, _doc.MediaBox
        _doc.MediaBox.Bottom);
        _doc.Transform.Translate(_doc.MediaBox.

        int theID = _doc.GetInfoInt(_doc.Root,
        _doc.SetInfo(theID, "/Rotate", "90");
    }

    public void Import(string inPath) {
        using(XpsImportOperation op = new XpsIn
        op.ProcessingObject += Processing;
        op.ProcessedObject += Processed;
        op.Import(_doc, inPath);
    }
}

    public void Processing(object sender, Proc
e) {
        if (e.Info.SourceType == ProcessingSour
e.Info.PageNumber != null) {
            if ((e.Info.PageNumber % 2) == 0)
                e.Info.PageNumber++;

            if (e.Info.PageNumber >= 8)
                e.Info.PageNumber = null;

            e.Tag = e.Info.PageNumber;

```

```

    } else if (e.Info.SourceType ==
ProcessingSourceType.PageContent) {
        if ((_pagesAdded % 2) == 0)
            _doc.Page = _doc.AddPage();

        double width = _doc.MediaBox.Height;
        double height = _doc.MediaBox.Width;
        double scale = Math.Min((width - 4 *
e.Info.Width.Value),
            (height - 2 * _margin) / e.Info.Hei

        double rectWidth = scale * e.Info.Wid
        double rectHeight = scale * e.Info.He

        double distanceX = (width - 2 * rectW
        double distanceY = (height - rectHeig

        _doc.Rect.SetRect(distanceX + (_pages
+ rectWidth),
            distanceY, rectWidth, rectHeight);

        e.Info.Handled = true;
        _pagesAdded++;
    }
}

public void Processed(object sender, Proc
{
    if (e.Successful) {
        PixMap pixmap = e.Object as PixMap;
        if (pixmap != null)
            pixmap.Compress();

        GraphicLayer graphic = e.Object as Gr
        if (graphic != null) {
            _doc.FrameRect();
        }
    }
}

```

```

        int pageNumber = (int)e.Tag;
        _doc.FontSize = 16;
        _doc.TextStyle.HPos = 0.5;
        _doc.Rect.Top = _doc.Rect.Bottom -
        _doc.Rect.Bottom = _doc.MediaBox.Bottom;
        _doc.AddText(string.Format("Page {0}"));
    }
}
}
}
}

```

### [Visual Basic]

```

Dim doc As New Doc()
Dim importOp As MyImportOperation = New MyImportOperation()
importOp.Import(Server.MapPath("AdvancedGraphics.xps"))
doc.Save(Server.MapPath("xps.pdf"))
doc.Clear();

```

```

Public Class MyImportOperation
    Private _doc As Doc = Nothing
    Private _margin As Double = 10
    Private _pagesAdded As Integer = 0

    Public Sub New(doc As Doc)
        _doc = doc

        _doc.Transform.Rotate(90, _doc.MediaBox.Right,
        _doc.MediaBox.Bottom)
        _doc.Transform.Translate(_doc.MediaBox.Right,
        _doc.MediaBox.Bottom)

        Dim theID As Integer = _doc.GetInfoInt("Page", "90")
        _doc.SetInfo(theID, "/Rotate", "90")
    End Sub

    Public Sub Import(inPath As String)
        Using op As New XpsImportOperation()

```

**Example**

```

        AddHandler op.ProcessingObject, Address
        AddHandler op.ProcessedObject, Address
        op.Import(_doc, inPath)
    End Using
End Sub

Public Sub Processing(sender As Object, e As
ProcessingObjectEventArgs)
    If e.Info.SourceType = ProcessingSource
e.Info.PageNumber IsNot Nothing Then
        If (e.Info.PageNumber Mod 2) = 0 Then
            e.Info.PageNumber += 1
        End If

        If e.Info.PageNumber >= 8 Then
            e.Info.PageNumber = Nothing
        End If

        e.Tag = e.Info.PageNumber
    ElseIf e.Info.SourceType = ProcessingSc
Then
        If (_pagesAdded Mod 2) = 0 Then
            _doc.Page = _doc.AddPage()
        End If

        Dim width As Double = _doc.MediaBox.F
        Dim height As Double = _doc.MediaBox.F
        Dim scale As Double = Math.Min((width
e.Info.Width.Value),
            (height - 2 * _margin) / e.Info.Hei

        Dim rectWidth As Double = scale * e.I
        Dim rectHeight As Double = scale * e.

        Dim distanceX As Double = (width - 2
        Dim distanceY As Double = (height - r

```

```

        _doc.Rect.SetRect(distanceX + (_pages
(distanceX + rectWidth), _
        distanceY, rectWidth, rectHeight)

        e.Info.Handled = True
        _pagesAdded += 1
    End If
End Sub

Public Sub Processed(sender As Object, e
ProcessedObjectEventArgs)
    If e.Successful Then
        Dim pixmap As PixMap = TryCast(e.Object)
        If pixmap IsNot Nothing Then
            pixmap.Compress()
        End If

        Dim graphic As GraphicLayer = TryCast
        If graphic IsNot Nothing Then
            _doc.FrameRect()

            Dim pageNumber As Integer = CInt(e.
            _doc.FontSize = 16
            _doc.TextStyle.HPos = 0.5
            _doc.Rect.Top = _doc.Rect.Bottom -
            _doc.Rect.Bottom = _doc.MediaBox.Bottom
            _doc.AddText(String.Format("Page {0}
        End If
    End If
End Sub
End Class

```

Stroke path example



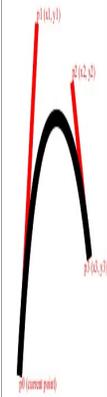
Page 1

Even-Odd filling rule example



Page 3

Line cap example



Page 1

Line cap example



Page 1

# ImportAny Function

---



Imports a portion of a document via an intermediate XPS print-out.

## [C#]

```
void ImportAny(Doc doc, string path)
void ImportAny(Doc doc, string path, int timeout
```

## [Visual Basic]

### Syntax

```
Sub ImportAny(doc As Doc, path As String)
Sub ImportAny(doc As Doc, path As String, timeout As Integer)
```

- may throw Exception()

### Params

Name	Description
doc	The target PDF document.
path	The file path to the source document.
timeout	The timeout in milliseconds for the XPS printer driver. The default is 15500 milliseconds.

Imports several types of documents, by converting them to XPS first using the Microsoft XPS Document Writer (MXDW). An attempt is made to silently print the document to a temporary XPS file. An application which will print the document is launched. This application is intercepted and, assuming that only the default Print dialogs are used, the XPS print output is silently captured. This output is then imported into PDF just like any other XPS file.

You can refer to [Application](#) for a list of supported applications. This property also determines which application will be used to print the document to XPS.

By default the shell default program will be used. To set the shell default program for a file, in Windows Explorer, right-click on the file, select "Open With" and then "Choose Default Program". When the shell default program is used, there must be a "Print" option when right-clicking on the document in Windows Explorer. If this is not the case, ImportAny will fail except for the following applications: MS Word, MS Excel, MS PowerPoint, MS InfoPath and MS Project.

If you prefer not to rely on the shell default program, set [Application](#) to any of the applications defined in `ImportAnyApplicationType`. These applications are officially supported by ImportAny. You may try your luck with other applications, but in this case you must associate your application via the shell default program. We officially support Microsoft Office 2007, older versions may work for some applications but there is no guarantee.

The Microsoft XPS Document Writer (MXDW) must also be available. This is the default on Windows Vista. For Windows XP, you can download MXDW at this address:  
[http://msdn.microsoft.com/en-us/library/dd183313\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd183313(VS.85).aspx).

When ImportAny fails, the associated application will likely hang, normally waiting for a "Print" dialog of some sort to be closed, but this dialog is hidden by ImportAny so you will not see it. If this is the case, ImportAny will wait for the specified timeout before killing the associated application and reporting an error. The default timeout is 15.5 seconds. You can reduce or increase this value depending on your requirements, by using the overload that accepts a timeout as a parameter.

**When using ImportAny from either a Windows Service or an IIS process, it is recommended that you wrap ABCpdf into a COM+ application. Refer to the PdfEnterpriseServices example in order to do this.**

You may debug ImportAny problems using the PrintHookShow and PrintHookLog registry keys. For further details see the [Registry Keys](#) section in Concepts.

The usage of ImportAny is identical to [Import](#) except that it takes any file format whose associated application supports the "Print" option.

---

**Example**





# KillApplication Function

---

Terminates a running import application.

## [C#]

```
bool  
KillApplication(ImportApplicationType  
app)
```

---

## Syntax

## [Visual Basic]

```
Function KillApplication(app As  
ImportApplicationType) As Boolean
```

---

## Params

Name	Description
app	The import <a href="#">application</a> .
return	Whether the application has been running and killed.

---

## Notes

This function terminates an application [running](#) for this XpsImportOperation when the process reuse for the application is enabled with [EnableApplicationReuse](#).

---

Example

None.

# ReusesApplication Function

---



Gets a value indicating whether an import application process is to be kept running and reused.

## [C#]

```
bool  
ReusesApplication(ApplicationType  
app)
```

---

### Syntax

## [Visual Basic]

```
Function ReusesApplication(app As  
ImportApplicationType) As Boolean
```

---

### Params

Name	Description
app	The import <a href="#">application</a> .
return	Whether the application process is to be kept running and reused.

This function returns the reuse status of an application.

---

### Notes

The reuse status can be changed with [EnableApplicationReuse](#).

---

**Example**

None.

# Application Property

---



Type	Default	Read Only	Description
[C#]ImportApplicationType			The application used for import.
[Visual Basic]ImportApplicationType	ShellAssociation	No	The application used for import.

The application used by the import-any operation, see [ImportAny](#)

This property can have one of the following values:

- ShellAssociation – This is the default value. The Shell will decide which application to use based on file association. The application that starts up when opening a file from the Windows explorer will also be used to do the conversion.
- MicrosoftWord – Microsoft Word will be used. If it is not installed an exception will be thrown.
- MicrosoftExcel – Microsoft Excel will be used. If it is not installed an exception will be thrown.
- MicrosoftPowerPoint – Microsoft PowerPoint will be used. If it is not installed an exception will be thrown.
- MicrosoftInfoPath – Microsoft InfoPath will be used. If it is not installed an exception will be thrown. Because only one process exists at any one time per user, the interactive user cannot use InfoPath if ABCpdf is importing InfoPath documents from an application running interactively. If the ABCpdf application runs from within a Windows service this limitation does not apply because a separate instance of InfoPath is created. However you cannot have multiple

---

**Notes**

ABCpdf processes using InfoPath at the same time. A mutex has been implemented to prevent this. So only one InfoPath document at a time can be imported.

- MicrosoftOneNote - Microsoft OneNote will be used. If it is not installed an exception will be thrown. Because only one process instance exists at any one time per user, the same limitations apply, as described for InfoPath.
- MicrosoftPublisher - Microsoft Publisher will be used. If it is not installed an exception will be thrown.
- MicrosoftProject - Microsoft Project will be used. If it is not installed an exception will be thrown. Because only one process instance exists at any one time per user, the same limitations apply, as described for InfoPath.
- MicrosoftVisio - Microsoft Visio will be used. If it is not installed an exception will be thrown.
- MicrosoftOutlook - Microsoft Outlook will be used. If it is not installed an exception will be thrown. Importing Outlook files will not work if Outlook is already running. This is because when Outlook is already running, it is the existing process that will perform the printing, not the new process created for printing.

---

**Example** None.

# ApplicationFolder Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	null	No	The folder of the application used by ImportAny.

The folder of the application used by the import-any operation, see [ImportAny](#).

---

## Notes

If this property is null, the application path will be determined automatically by looking at the registry. Set this property only if the automatic mechanism is failing.

---

## Example

None.

# DocNumber Property

---



Type	Default	Read Only	Description
[C#]int			
[Visual Basic] Integer	0	No	The one-based document number used by ImportAny.

This property can be used to select which document will be imported when calling [ImportAny](#). In most cases only a single document is available in any file and this property is ignored. However in some cases, such as Microsoft Excel, a single workbook contains multiple worksheets. This property will select which Excel worksheet will be imported.

When it is set to zero or a negative value, the entire Excel workbook will be imported. This is the default case. When it is set to any positive value the corresponding worksheet will be imported. Indexing is one-based so set it to one to print the first worksheet and so on. It is up to you to make sure that you set it to a valid value, less than or equal to the total number of worksheets in the workbook.

For applications other than Microsoft Excel this property is currently ignored.

---

## Notes

---

**Example**

None.

# EnableMSOfficeMacros Property

---



Type	Default	Read Only	Description
[C#]bool [Visual Basic] Boolean	false	No	Whether to enable macros when opening MS Office documents.

This property specifies whether to enable macros when opening MS Office documents. It is supported only for MS Word, Excel and PowerPoint. It is ignored in all other cases.

---

## Notes

By default, macros are disabled.

---

## Example

None.

# Password Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	null	No	The password needed to read the source.

Specify with this property the password for accessing the source. It is supported only for MS Word and MS Excel documents.

---

## Notes

---

## Example

None.



# SwfImportOperation Constructor

---

SwfImportOperation Constructor.

## [C#]

```
SwfImportOperation()
```

---

## Syntax

## [Visual Basic]

```
Sub New()
```

---

## Params

Name	Description
none	

Create a SwfImportOperation.

---

## Notes

You must specify a [Doc](#) before the [ProcessingObject](#) event of [ProcessingSourceType.ImageFrame](#) returns/is handled.

---

## Example

See the [Import](#) method.





# Import Function

---

Imports selected frames of a Flash movie.

## [C#]

```
void Import(string path)
void Import(Stream stream, string url)
```

## [Visual Basic]

Syntax

```
Sub Import(path As String)
Sub Import(stream As Stream, url As String)
```

- may throw Exception()

Params

Name	Description
path	The file path to the Flash movie.
stream	The stream containing the Flash movie.
url	The value of the MovieClip._url ActionScript property.

Imports selected frames of a Flash movie. By default, only color mode 2 below is supported. ActionScript 3 is not supported.

---

**Alpha Blending.** PDF supports using different color spaces.

a transparent object on a page, the blending color space DeviceCMYK. You will likely see some color shifts even converted to the alpha blending color space and then to more noticeable for objects using component-wise alpha

To use RGB as it is used in SWF for alpha blending, see the page to DeviceRGB:

```
doc.SetInfo(doc.Page, "/Group*/Type:Name",  
doc.SetInfo(doc.Page, "/Group*/S:Name",  
doc.SetInfo(doc.Page, "/Group*/CS:Name",
```

You must specify a [Doc](#) before the [ProcessingObject](#) event returns/is handled.

An exception will be thrown if the operation is not possible. corrupt.

This method generates events as follows:

1. Before the Flash engine advances, a [ProcessingObject](#) [ProcessingSourceType.MultiFrameImage](#) is generated. [BackgroundRegion](#) and [ClipRegion](#) are reset to the frame when the event is generated. The regions are set as if [Width](#), and [Height](#) properties of the [ProcessingInfo](#). You must initialize the SWF machine before the first time a page

Name	Description
<code>ProcessingObjectEventArgs.Cancel</code>	See <a href="#">ProcessingObjectEventArgs</a> below
<code>ProcessingObjectEventArgs.Info.Handled</code>	Ignore
<code>ProcessingObjectEventArgs.Info.SourceType</code>	Processing
<code>ProcessingObjectEventArgs.Info.FrameCount</code>	The number of frames
	The page number

<code>ProcessingObjectEventArgs.Info.FrameNumber</code>	value value is on hand or gr even corre indica end t null, 1 is not Proce true, is not imme to nu
<code>ProcessingObjectEventArgs.Info.FrameRate</code>	The r movi
<code>ProcessingObjectEventArgs.Info.X</code>	The x of the
<code>ProcessingObjectEventArgs.Info.Y</code>	The y of the
<code>ProcessingObjectEventArgs.Info.Width</code>	The v movi
<code>ProcessingObjectEventArgs.Info.Height</code>	The h movi

2. After a proper `FrameNumber` is returned in a previous `ProcessingSourceType.MultiFrameImage` and the `FlashProcessingObject` event of `ProcessingSourceType.Image` is true, `BackgroundRegion` and `ClipRegion` are reset to the movie when the event is generated because the frame mode. You must specify a `Doc` before this event returns. `Doc.Rect` of the current page using `Doc.Transform`.

---

Notes

Name	Doc
	If t

ProcessingObjectEventArgs.Cancel	re Pr
ProcessingObjectEventArgs.Info.Handled	Ig
ProcessingObjectEventArgs.Info.SourceType	Pr
ProcessingObjectEventArgs.Info.FrameCount	Th
ProcessingObjectEventArgs.Info.FrameNumber	Th of
ProcessingObjectEventArgs.Info.FrameRate	Th Fl
ProcessingObjectEventArgs.Info.X	Th bc
ProcessingObjectEventArgs.Info.Y	Th bc
ProcessingObjectEventArgs.Info.Width	Th m
ProcessingObjectEventArgs.Info.Height	Th Fl
ProcessingObjectEventArgs.Info.BackgroundColor	Th

3. While rendering a frame, a `ProcessingObject` event of `EventArgs` is raised when an image is imported.

Name	Desc
ProcessingObjectEventArgs.Cancel	Ignor
ProcessingObjectEventArgs.Info.Handled	Ignor
ProcessingObjectEventArgs.Info.SourceType	Proce
ProcessingObjectEventArgs.Info.FrameCount	The r
ProcessingObjectEventArgs.Info.FrameNumber	The r the m
ProcessingObjectEventArgs.Info.FrameRate	The r movi
ProcessingObjectEventArgs.Info.Width	The p
ProcessingObjectEventArgs.Info.Height	The p

4. While rendering a frame, a [ProcessingObject](#) event of type [ProcessingObjectEvent](#) is generated when an image used as a mask is imported.

Name	Description
<a href="#">ProcessingObjectEventArgs.Cancel</a>	Ignored
<a href="#">ProcessingObjectEventArgs.Info.Handled</a>	Ignored
<a href="#">ProcessingObjectEventArgs.Info.SourceType</a>	ProcessingSource
<a href="#">ProcessingObjectEventArgs.Info.FrameCount</a>	The number of frames in the movie clip.
<a href="#">ProcessingObjectEventArgs.Info.FrameNumber</a>	The number of the frame being processed.
<a href="#">ProcessingObjectEventArgs.Info.FrameRate</a>	The frame rate of the movie clip.
<a href="#">ProcessingObjectEventArgs.Info.Width</a>	The width of the image.
<a href="#">ProcessingObjectEventArgs.Info.Height</a>	The height of the image.

Unless stated otherwise, a [ProcessedObject](#) event corresponding to the [ProcessingObject](#) event is generated.

<a href="#">ProcessingObjectEventArgs.Info.SourceType</a>	ProcessedObjectEvent
<a href="#">ProcessingSourceType.MultiFrameImage</a>	<a href="#">GraphicText</a>
<a href="#">ProcessingSourceType.ImageFrame</a>	null
<a href="#">ProcessingSourceType.Image</a>	<a href="#">PixMap</a>
<a href="#">ProcessingSourceType.ImageMask</a>	<a href="#">PixMap</a>

Imported images are not compressed. You may wish to analyze and process them by pre and post processing them during the [ProcessingObject](#) event.

See the [SwfParameters.FlashVars](#) property or the [ProcessingObject](#) simple examples.

Here we import 24 frames from a Flash movie alternately in are 0.2 seconds apart. [Doc.Rect](#) is set so that the outputs a ratio. For the fifth to the eighth imported frames in each doc suppressed and a transparent oval in the same color is draw

### [C#]

```
Doc doc1 = new Doc();
Doc doc2 = new Doc();
using(SwfImportOperation operation = new SwfImportOperation())
{
    const int fontSize = 20;
    int k = 0;
    bool failed = false;
    operation.ProcessingObject += delegate(object ProcessingObjectEventArgs e) {
        switch(e.Info.SourceType) {
            case ProcessingSourceType.MultiFrameImage:
                if(failed || k>=24)
                    e.Info.FrameNumber = null;
                else
                    e.Info.FrameNumber =
1+Convert.ToInt64(0.2*k*e.Info.FrameRate.Value);
                e.Tag = ProcessingSourceType.MultiFrameImage;
                break;
            case ProcessingSourceType.ImageFrame: {
                SwfImportOperation op = (SwfImportOperation)operation;
                op.Doc = op.Doc==doc1? doc2: doc1;
                const int distance = 20;
                const int margin = 30;
                double width = op.Doc.MediaBox.Width;
                double height = op.Doc.MediaBox.Height;
                double scale = Math.Min((width-2*distance)/(2*e.Info.Width.Value),
                    (height-2*distance-3*fontSize)/(3*e.Info.Height.Value));

                int p = k/2;
                double rectWidth = scale*e.Info.Width.Value;
                double rectHeight = scale*e.Info.Height.Value;
            }
        }
    };
}
```

```

        op.Doc.Rect.SetRect(margin+(width+dis
            margin+height-rectHeight-(height+2*
                rectWidth, rectHeight));
        if(p%6==0)
            op.Doc.Page = op.Doc.AddPage();
        if(p>=4 && p<8 && e.Info.BackgroundCo
            op.BackgroundRegion = null;
            op.Doc.Color.String = e.Info.Backgr
            op.Doc.Color.Alpha = 127;
            op.Doc.AddOval(true);
        }
    } break;
}
};
operation.ProcessedObject += delegate(obj
ProcessedObjectEventArgs e) {
    if(!e.Successful) {
        failed = true;
        return;
    }
    if(e.Tag is ProcessingSourceType
        &&
(ProcessingSourceType)e.Tag==ProcessingSour
    {
        SwfImportOperation op = (SwfImportOpe
        op.Doc.Color.Gray = 0;
        op.Doc.Color.Alpha = 255;
        op.Doc.FontSize = fontSize;
        op.Doc.TextStyle.HPos = 0.5;
        op.Doc.Rect.Top = op.Doc.Rect.Bottom;
        op.Doc.Rect.Bottom = op.Doc.MediaBox.
        op.Doc.AddText(string.Format("{0} sec
        ++k;
    }
    PixMap pixmap = e.Object as PixMap;
    if(pixmap!=null)
        pixmap.Compress();
}

```

```

};
operation.Import(Server.MapPath("ABCpdf.s
}
doc1.Save(Server.MapPath("swf1.pdf"));
doc1.Clear();
doc2.Save(Server.MapPath("swf2.pdf"));
doc2.Clear();

```

## [Visual Basic]

```

Dim theMyOp As MyOp
Using theOperation As New SwfImportOperatic
    theMyOp = New MyOp(theOperation)
    theOperation.Import(Server.MapPath("ABCpc
End Using
theMyOp.Save(Server.MapPath("swf1.pdf"), Se
theMyOp.Clear()

```

```

Private NotInheritable Class MyOp
    Private Const theFontSize As Integer = 20
    Private theDoc1 As Doc
    Private theDoc2 As Doc
    Private k As Integer
    Private theFailed As Boolean

```

```

Public Sub New(theOperation As SwfImportC
    theDoc1 = New Doc()
    theDoc2 = New Doc()
    k = 0
    theFailed = False
    AddHandler theOperation.ProcessingObjec
    AddHandler theOperation.ProcessedObject
End Sub

```

```

Public Sub Save(path1 As String, path2 As
    theDoc1.Save(Server.MapPath(path1))
    theDoc2.Save(Server.MapPath(path2))

```

---

Example

```
End Sub

Public Sub Clear()
    theDoc1.Clear()
    theDoc2.Clear()
End Sub

Private Sub Processing(sender As Object,
ProcessingObjectEventArgs)
    Select Case e.Info.SourceType
    Case ProcessingSourceType.MultiFrameImage
        If theFailed Or k >= 24 Then
            e.Info.FrameNumber = Nothing
        Else
            e.Info.FrameNumber = 1 + Convert.ToInt32(
e.Info.FrameRate.Value)
        End If
        e.Tag = ProcessingSourceType.MultiFrameImage
    Case ProcessingSourceType.ImageFrame
        Dim op As SwfImportOperation = CType(
SwfImportOperation)
        If op.Doc Is theDoc1 Then
            op.Doc = theDoc2
        Else
            op.Doc = theDoc1
        End If
        Const distance As Integer = 20
        Const margin As Integer = 30
        Dim width As Double = op.Doc.MediaBox.Width
        Dim height As Double = op.Doc.MediaBox.Height
        Dim scale As Double = Math.Min((width -
e.Info.Width.Value), (height - 2 *
distance - 3 * theFor
e.Info.Height.Value))

        Dim p As Integer = k \ 2
        Dim rectWidth As Double = scale * e.I
```

```

        Dim rectHeight As Double = scale * e.
        op.Doc.Rect.SetRect(margin + (width +
2, _
        margin + height - rectHeight - (hei
2 Mod 3) / 3, _
        rectWidth, rectHeight)
    If p Mod 6 = 0 Then
        op.Doc.Page = op.Doc.AddPage()
    End If
    If p >= 4 And p < 8 And e.Info.Backgr
Then
        op.BackgroundRegion = Nothing
        op.Doc.Color.String = e.Info.Backgr
        op.Doc.Color.Alpha = 127
        op.Doc.AddOval(True)
    End If
End Select
End Sub

```

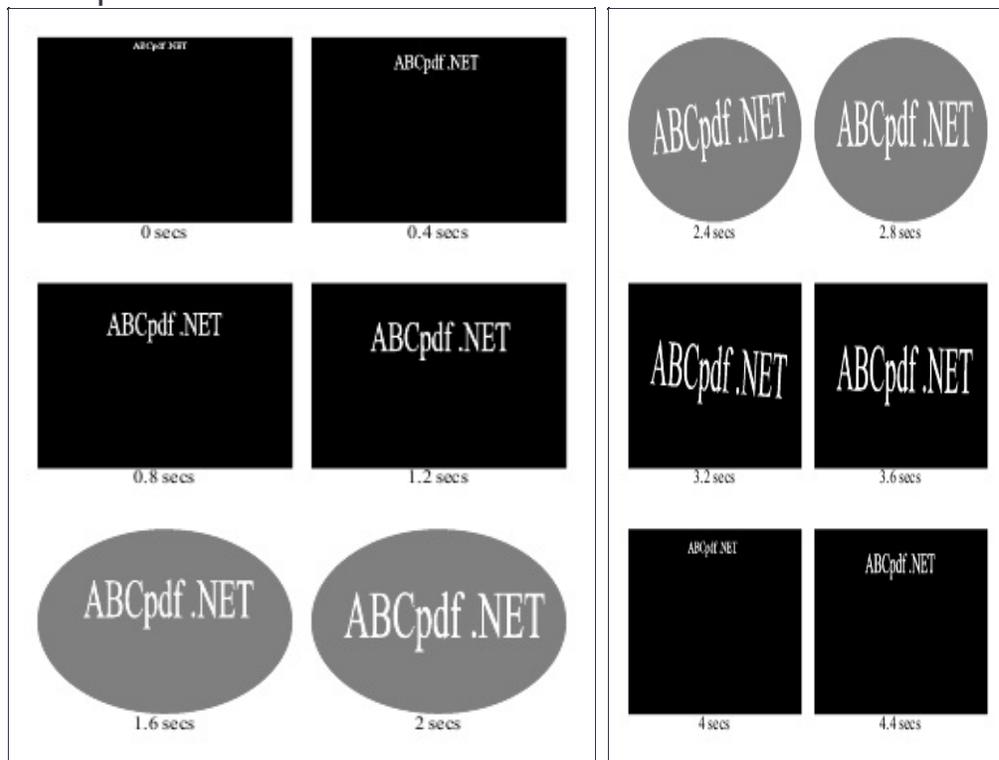
```

Private Sub Processed(sender As Object, e
ProcessedObjectEventArgs)
    If Not e.Successful Then
        theFailed = True
        Return
    End If
    If TypeOf e.Tag Is ProcessingSourceType
        If CType(e.Tag, ProcessingSourceType
ProcessingSourceType.MultiFrameImage Then
            Dim op As SwfImportOperation = CTy
SwfImportOperation)
            op.Doc.Color.Gray = 0
            op.Doc.Color.Alpha = 255
            op.Doc.FontSize = theFontSize
            op.Doc.TextStyle.HPos = 0.5
            op.Doc.Rect.Top = op.Doc.Rect.Bott
            op.Doc.Rect.Bottom = op.Doc.MediaE
            op.Doc.AddText(String.Format("{0}

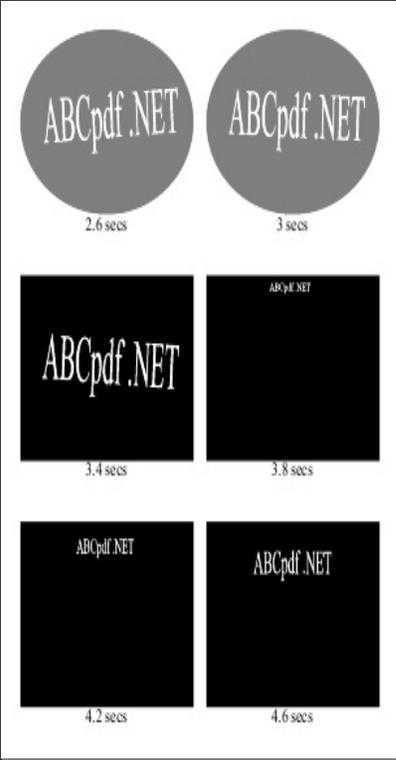
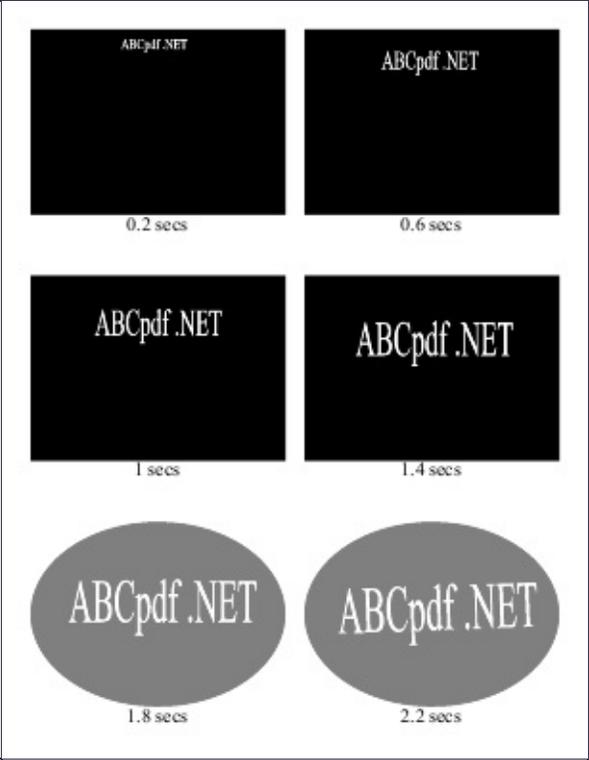
```

```
        k += 1
    End If
End If
Dim thePixMap As PixMap = TryCast(e.Obj)
If thePixMap IsNot Nothing Then
    thePixMap.Compress()
End If
End Sub
End Class
```

swf1.pdf



swf2.pdf



# Doc Property

---



Type	Default	Read Only	Description
[C#]Doc			
[Visual Basic]Doc	null	No	The target PDF document.

The document used by the import operation. This is where the flash movie frames are imported into.

---

## Notes

This property must not be null before the [ProcessingObject](#) event of [ProcessingSourceType.ImageFrame](#) returns/is handled.

---

## Example

See the [Import](#) method.

# BackgroundRegion Property

---



Type	Default	Read Only	Description
[C#]XRect			
[Visual Basic]XRect	null	No	Gets or sets the background rectangle.

The background region (in twips) is where the frame background is drawn. Set this property to change the extent or location of the background. The background region is set to the frame bounds/stage size if [ResetRegions](#) is true. If it is null (and not set to non-null because of [ResetRegions](#)), no background is drawn.

---

## Notes

See the [Import](#) method.

---

## Example

# ClipRegion Property

---



Type	Default	Read Only	Description
[C#]XRect			
[Visual Basic]XRect	null	No	Gets or sets the clip rectangle.

The clip region (in twips) is where the frame foreground is drawn. Set this property to change the extent or location of the frame foreground. If the foreground extends outside of this region, it will be clipped. The clip region is set to the frame bounds/stage size if [ResetRegions](#) is true. If it is null (and not set to non-null because of [ResetRegions](#)), the content is not clipped.

---

## Notes

None.

---

## Example

# ResetRegions Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Gets or sets a value that indicates whether BackgroundRegion and ClipRegion are reset when the Operation.ProcessingObject event of ProcessingSourceType.MultiFrameImage or of ProcessingSourceType.ImageFrame is generated.

Indicates whether the background and clip regions should be reset to the frame bounds each time a new frame is about to be imported. If regions are set as if `XRect.SetRect` is called with the `X`, `Y`, `Width`, and `Height` properties of the `ProcessingInfo`. Set this property to false if you want the regions not to be reset when the `Operation.ProcessingObject` event of `ProcessingSourceType.MultiFrameImage` or of `ProcessingSourceType.ImageFrame` is generated.

---

## Notes

None.

---

## Example

# Timeout Property

---



Type	Default	Read Only	Description
[C#]TimeSpan?  [Visual Basic] Nullable(Of TimeSpan)	null	No	Gets or sets the time-out for script execution.

This property specifies for each advancement of the Flash engine the period within which scripts are executed while the Flash engine advances to the specified frame. It enables the termination of infinite loops in the scripts.

---

## Notes

None.

---

## Example

# ContentAlign Property

---



Type	Default	Read Only	Description
[C#]ContentAlign?  [Visual Basic] Nullable(Of ContentAlign)	null	No	Gets or sets the content alignment.

This property specifies the alignment of the content in the target rectangle.

It can take a combination of the following values:

- Center
- Left
- Right
- Top
- Bottom

---

## Notes

If it is null, the value of the ActionScript property `Stage.align` is used.

See the [ProcessingInfo.FrameNumber](#) property.

---

## Example



# ContentScaleMode Property

---



Type	Default	Read Only	Description
[C#]ContentScaleMode?  [Visual Basic] Nullable(Of ContentScaleMode)	ExactFit	No	Gets or sets the content scale mode.

This property specifies the scale mode of the content in the target rectangle.

It can take any of the following values:

- ShowAll – make the content the biggest and completely within the area while keeping the aspect ratio.
- NoBorder – make the content the smallest and covering the entire area while keeping the aspect ratio.
- ExactFit – scale the content possibly with distortion to fit the entire area.
- NoScale – no scaling.

---

## Notes

If it is null, the value of the ActionScript property `Stage.scaleMode` is used.

See the [ProcessingInfo.FrameNumber](#) property.

---

## Example



# Parameters Property

---



Type	Default	Read Only	Description
[C#]SwfParameters [Visual Basic]SwfParameters	null	No	The parameters for initializing the SWF machine.

This property specifies the parameters for initializing the SWF machine.

---

## Notes

If it is set, it should be set before the first time a proper [FrameNumber](#) is returned in a [ProcessingObject](#) event of [ProcessingSourceType.MultiFrameImage](#). It is not used after the machine is initialized.

---

## Example

See the [SwfParameters.FlashVars](#) property.

# FlashVars Property

---



Type	Default	Re Or
<b>[C#]</b> IEnumerable<KeyValuePair<string, object>>	null	Ne
<b>[Visual Basic]</b> IEnumerable(Of KeyValuePair(Of String, Object))		

This property specifies the Flash variables. The object values can be the following:

- SwfActionValue.Undefined – the undefined value.
- SwfActionValue.Null – the null value.
- A Boolean – a primitive Boolean.
- A Char – a primitive string consisting of one character.
- A SByte, Byte, Int16, UInt16, or Int32 – a primitive number (signed integer in the range of 32-bit integer; double-precision floating-point number otherwise).
- A Single – a primitive number (single-precision floating-point number).
- A Double – a primitive number (double-precision floating-point number).
- A Decimal – a primitive number (32-bit integer if it is an integer; double-precision floating-point number otherwise).
- A String – a primitive string.
- IEnumerable<KeyValuePair<string, object>> – an object with KeyValuePair<string, object>.Key as the property names and KeyValuePair<string, object>.Value as the property values.

KeyValuePair<string, object>.Value after this conversion as values.

- IEnumerable<object> – an array with the objects after this conversion and the element values.

Reference semantics of objects is supported. When the same IEnumerable<KeyValuePair<string, object>> or the same IEnumerable<object> is in different parts of this property value, the same object is used. You can create objects and properties/element values that form cross-references.

Here, we import the chart in the frame at two seconds into a Flash movie clip.

### [C#]

```
Doc doc = new Doc();
using(SwfImportOperation operation = new SwfImportOperation())
{
    operation.Doc = doc;
    operation.ContentAlign = ContentAlign.Top;
    operation.ContentScaleMode = ContentScaleMode.NoScale;
    operation.ProcessingObject += delegate(object obj,
        ProcessingObjectEventArgs e) {
        if(e.Info.SourceType==ProcessingSourceType.MovieClip
            && e.Info.FrameNumber.HasValue)
            e.Info.FrameNumber = 1+(long)(e.Info.FrameNumber
                * 2);
    };

    const int chartWidth = 400;
    const int chartHeight = 300;
    SwfParameters parameters = new SwfParameters();
    parameters.StageWidth = chartWidth*20;
    parameters.StageHeight = chartHeight*20;

    Dictionary<string, object> flashVars = new
```

```

Dictionary<string, object>();
    flashVars.Add("chartWidth", chartWidth);
    flashVars.Add("chartHeight", chartHeight);
    flashVars.Add("dataXML",
        "<chart caption='Weekly Sales Summary' xAxi
        +" yAxisName='Amount' numberPrefix='$'>"
        +"<set label='Week 1' value='14400' />"
        +"<set label='Week 2' value='19600' />"
        +"<set label='Week 3' value='24000' />"
        +"<set label='Week 4' value='15700' />"
        +"</chart>");
    parameters.FlashVars = flashVars;
    operation.Parameters = parameters;
    operation.Import(Server.MapPath("Column3D.swf
}
doc.Save(Server.MapPath("chart.pdf"));
doc.Clear();

```

## [Visual Basic]

### Example

```

Dim theDoc As Doc = New Doc()
Using theOperation As New SwfImportOperation
    theOperation.Doc = theDoc
    theOperation.ContentAlign = ContentAlign.Top
    theOperation.ContentScaleMode = ContentScaleM
    AddHandler theOperation.ProcessingObject, Add
Processing

    Const theChartWidth As Integer = 400
    Const theChartHeight As Integer = 300
    Dim theParameters As SwfParameters = New SwfP
    theParameters.StageWidth = theChartWidth * 20
    theParameters.StageHeight = theChartHeight *

    Dim theFlashVars As Dictionary(Of String, Obj
Dictionary(Of String, Object)()
    theFlashVars.Add("chartWidth", theChartWidth)

```

```

theFlashVars.Add("chartHeight", theChartHeight)
theFlashVars.Add("dataXML",
    "<chart caption='Weekly Sales Summary' xAxisName='Week' yAxisName='Amount' numberPrefix='$'>"
    & "<set label='Week 1' value='14400' />"
    & "<set label='Week 2' value='19600' />"
    & "<set label='Week 3' value='24000' />"
    & "<set label='Week 4' value='15700' />"
    & "</chart>")
theParameters.FlashVars = theFlashVars
theOperation.Parameters = theParameters
theOperation.Import(Server.MapPath("Column3D.mxml"))
End Using
theDoc.Save(Server.MapPath("chart.pdf"))
theDoc.Clear()

Private Shared Sub Processing(sender As Object, e As ProcessingObjectEventArgs)
    If e.Info.SourceType = ProcessingSourceType.Month Then
        If Also e.Info.FrameNumber.HasValue Then
            e.Info.FrameNumber = 1 + CLng(e.Info.FrameNumber)
        End If
    End Sub

```

# StageAlign Property

---



Type	Default	Read Only	Description
[C#]ContentAlign?  [Visual Basic] Nullable(Of ContentAlign)	null	No	The stage alignment.

This property specifies the value of the ActionScript property `Stage.align`. If it is not null, `Stage.align` is set to (the string representation of) this value.

It can take a combination of the following values:

---

## Notes

- Center
- Left
- Right
- Top
- Bottom

None.

---

## Example

# StageHeight Property

---



Type	Default	Read Only	Description
[C#]double? [Visual Basic] Nullable(Of Double)	null	No	The stage height when the scale mode is NoScale.

This property specifies 20 times the value of the ActionScript property Stage.height. If it is not null, Stage.height holds 1/20 of this value when Stage.scaleMode is "noScale".

---

## Notes

See the [FlashVars](#) property.

---

## Example

# StageScaleMode Property

---



Type	Default	Read Only	Description
[C#]ContentScaleMode?  [Visual Basic] Nullable(Of ContentScaleMode)	null	No	The stage scale mode

This property specifies the value of the ActionScript property `Stage.scaleMode`. If it is not null, `Stage.scaleMode` is set to (the string representation of) this value.

It can take any of the following values:

- ShowAll – make the content the biggest and completely within the area while keeping the aspect ratio.
- NoBorder – make the content the smallest and covering the entire area while keeping the aspect ratio.
- ExactFit – scale the content possibly with distortion to fit the entire area.
- NoScale – no scaling.

---

## Notes

None.

---

## Example



# StageWidth Property

---



Type	Default	Read Only	Description
[C#]double? [Visual Basic] Nullable(Of Double)	null	No	The stage width when the scale mode is NoScale.

This property specifies 20 times the value of the ActionScript property Stage.width. If it is not null, Stage.width holds 1/20 of this value when Stage.scaleMode is "noScale".

---

## Notes

See the [FlashVars](#) property.

---

## Example



# RenderOperation Constructor

---

RenderOperation Constructor.

## [C#]

```
RenderOperation(Doc doc)
```

---

### Syntax

## [Visual Basic]

```
Sub New(doc As Doc)
```

---

### Params

Name	Description
doc	The PDF Document

Create a RenderOperation to allow rendering of multiple pages from different threads.

You can use Doc.Rendering to set rendering options before creating a RenderOperation. When a RenderOperation is created, relevant doc properties are copied, so modifying properties of Doc.Rendering after creating the operation does not affect the RenderOperation. Likewise, setting the current page to another page after creating the operation has no effect.

---

### Notes

---

**Example**

See the [Save](#) method.



## Save Function

---

Renders a page into a file.

### [C#]

```
void Save(string path)
```

### [Visual Basic]

```
Sub Save(path As String)
```

- may throw `Exception()`

Syntax

Params

Name	Description
path	The destination file path.

Renders a page to a file. The current page at the time the operation has been created is rendered.

This method is similar in functionality to [XRendering.Save\(\)](#), but it can be safely called on different instances of `RenderOperation` from different threads even for the same page of the same document provided the page is not modified while being rendered because the document rendering options and the page ID

are local to the operation. In the example below we show how to render document pages using parallel threads.

Furthermore, this method generates the following events, allowing fine tuning of the rendering operations:

1. Before rendering begins, a [ProcessingObject](#) event of `ProcessingSourceType.PageContent` is generated. Set the event arguments' `Cancel` property to true to skip rendering altogether.
2. When a PDF path stroking or filling operator is found in the page content, a [ProcessingObject](#) event of `ProcessingSourceType.Path` is generated. Set the event arguments' `Cancel` property to true to skip this object. The stream length and position can be retrieved via the `StreamPosition` and `StreamLength` properties. Set the stream position to null to skip the remaining of the stream.
3. When a text PDF operator is found, a [ProcessingObject](#) event of `ProcessingSourceType.Text` is generated. The Unicode text can be retrieved in the `Text` property of the event `Info` property. the event cancel property to true to skip this object. The stream length and position can be retrieved via the [StreamPosition](#) and [StreamLength](#) properties. Set the stream position to null to skip the remaining of the stream.
4. When a shading PDF operator is found, a [ProcessingObject](#) event of `ProcessingSourceType.Shading` is

generated. The same comments of point 2 above apply, for skipping the object or retrieving/setting the stream position and length.

5. When an image is found (inline or XObject), a **ProcessingObject** event of `ProcessingSourceType.Image` is generated. If the image is XObject, the indirect object can be retrieved via the event arguments' `Object` property. The event cancel property to true to skip this object. The stream length and position can be retrieved via the **StreamPosition** and **StreamLength** properties. Set the stream position to null to skip the remaining of the stream.
6. When a Form XObject is found, a **ProcessingObject** event of `ProcessingSourceType.FormXObject` is generated. The indirect object can be retrieved via the event arguments' `Object` property. Set the event arguments' `Cancel` property to true to skip this object. The stream length and position can be retrieved via the **StreamPosition** and **StreamLength** properties. Set the stream position to null to skip the remaining of the stream. In addition, because Forms contain streams, a **ProcessingObject** event of `ProcessingSourceType.Stream` will follow. You can set the stream position to null to skip the stream at any time. That is events for the objects contained in the Form stream will be generated, as described in points 2 to 5. Setting the stream position to null for Form objects will skip the Form stream, not the entire page content.

Every `ProcessingObject` event is followed by a corresponding `ProcessedObject` event with the same source type. The `PageContentProcessedObject` event will be the last event received, in that all the page objects events are sandwiched between a `PageContent` processing and processed events. Similarly for form streams, all the form objects events are sandwiched between a `Stream` processing and processed events, which in turn are sandwiched between the `FormXObject` processing and processed events.

Any event property or event Info property not mentioned here are ignored.

Here we render all the pages of the doc using 10 threads at a time. We alternate rendering format between jpg and tiff. We also alternate resolution between 150 and 300 dpi. Note how the `RenderingOperation` is created in the constructor of `TheRenderingWorker`. This is because at this point a copy of the rendering options is made. Had we created the `RenderingOperation` in `DoWork`, we would have picked up only the last `doc.Rendering.DotsPerInch`, because the threads are started in the following loop. Also note how we dispose the operation in `DoWork`, to release resources stored on the native side (the copy of the rendering options basically).

### [C#]

```
class RenderingWorker {
    private string mPath;
    private RenderOperation mOp;

    public RenderingWorker(Doc inDoc,
        string inPath) {
```

```

        mPath = inPath;
        mOp = new
RenderOperation(inDoc);
    }

    public void DoWork() {
        mOp.Save(mPath);
        mOp.Dispose();
    }
}

Doc doc = new Doc();
doc.Read(Server.MapPath("ABCpdf.pdf"));
string[] theExts = { ".jpg", ".tif" };
int[] theDpis = { 150, 300 };
Thread[] threadList = new Thread[10];
int pageNum = 1, pageCount =
doc.PageCount;
while (pageNum <= pageCount) {
    int count = 0;
    while (count < threadList.Length &&
pageNum <= pageCount) {
        doc.Rendering.DotsPerInch =
theDpis[(pageNum - 1) % 2];
        doc.PageNumber = pageNum;
        string path =
Server.MapPath("ABCpdf" +
pageNum.ToString() + theExts[(pageNum -
1) % 2]);

        threadList[count] = new
Thread(new RenderingWorker(doc,
path).DoWork);
        ++count;
        ++pageNum;
    }
    for (int i = 0; i < count; ++i)

```

---

**Example**

```
        threadList[i].Start();
    for (int i = 0; i < count; ++i)
        threadList[i].Join();
    }
doc.Clear();
```

**[Visual Basic]**

```
Class RenderingWorker
    Private mPath As String
    Private mOp As RenderOperation

    Public Sub New(ByVal inDoc As Doc,
        ByVal inPath As String)
        mPath = inPath
        mOp = New
        RenderOperation(inDoc)
    End Sub

    Public Sub DoWork()
        mOp.Save(mPath)
        mOp.Dispose()
    End Sub
End Class

Dim doc As Doc = new Doc()
doc.Read(Server.MapPath("ABCpdf.pdf"))
Dim theExts As String() = {".jpg",
    ".tif"}
Dim theDpis As Integer() = {150, 300}
Dim threadList(10 - 1) As Thread
Dim pageNum As Integer = 1, pageCount
As Integer = doc.PageCount
While pageNum <= pageCount
    Dim count As Integer = 0
    While count < threadList.Length
        AndAlso pageNum <= pageCount
```

```
        doc.Rendering.DotsPerInch =
theDpis((pageNum - 1) Mod 2)
        doc.PageNumber = pageNum
        Dim path As String =
Server.MapPath("ABCpdf" &
pageNum.ToString()) + theExts((pageNum
- 1) Mod 2))

        threadList(count) = New
Thread(AddressOf New
RenderingWorker(doc, path).DoWork)
        count += 1
        pageNum += 1
    End While

    For i As Integer = 0 To count - 1
        threadList(i).Start()
    Next
    For i As Integer = 0 To count - 1
        threadList(i).Join()
    Next
End While
doc.Clear()
```



# GetBitmap Function

---

Renders a page into a bitmap.

## [C#]

```
System.Drawing.Bitmap GetBitmap()
```

## [Visual Basic]

```
Function GetData(name As String)  
As Byte()
```

- may throw `Exception()`

---

### Syntax

---

### Params

Name	Description
return	The Bitmap containing the image.

Renders a page into a Bitmap. The current page at the time the operation has been created is rendered.

This method is similar in functionality to [XRendering.GetBitmap\(\)](#), but it can be safely called on different instances of `RenderOperation` from different threads even for the same page of the same document provided the page is not modified while being rendered

---

### Notes

because the document rendering options and current page are saved when the operation is created.

---

**Example**

See the [Save](#) method.



## GetData Function

---

Renders a page into an array of bytes.

### [C#]

```
byte[] GetData(string name)
```

### [Visual Basic]

```
Function GetData(name As String)  
As Byte()
```

Syntax

- may throw `Exception()`

Params

Name	Description
name	A dummy file name used to determine the type of image required.
return	The image as an array of bytes.

Renders a page into an array of bytes. The current page at the time the operation has been created is rendered.

This method is similar in functionality to [XRendering.GetData\(\)](#), but it can be safely called on different instances of

---

**Notes** RenderOperation from different threads even for the same page of the same document provided the page is not modified while being rendered because the document rendering options and current page are saved when the operation is created.

---

**Example** See the [Save](#) method.

# FromDoc Function

---



Gets the conformance identification from a document.

## [C#]

```
static  
PdfConformanceIdentification  
FromDoc(Doc doc, PdfConformance  
conformance)
```

---

### Syntax

## [Visual Basic]

```
Shared Function FromDoc(doc As  
Doc, conformance As  
PdfConformance) As  
PdfConformanceIdentification
```

---

### Params

Name	Description
doc	The document.
conformance	The conformance specification for which the identification is to be retrieved.
return	The conformance identification.

This methods identifies the conformance a

document is marked to satisfy. It is possible that a document conforms to more than one specification (e.g. PDF/A-1 and PDF/X-1). The conformance parameter identifies the desired specification. The Pdf value for conformance is not allowed. Both PdfA1b and PdfA1a specify the PDF/A-1 specification.

---

**Notes**

None.

---

**Example**

# Part Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	""	No	The conformance version identifier.

This property indicates the part number of the conformance specification. For PDF/A-1, the value is "1".

---

## Notes

None.

---

## Example

# Amd Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	""	No	The conformance amendment identifier.

This property indicates the amendment to a part of the conformance specification. For PDF/A, the value is the amendment number and year, separated by a colon. There is currently no amendment to PDF/A-1.

---

## Notes

None.

---

## Example

# Conformance Property

---



Type	Default	Read Only	Description
[C#]string			
[Visual Basic]String	""	No	The conformance level.

This property indicates the conformance level. For PDF/A-1a, the value is "A"; for PDF/A-1b, the value is "B".

---

## Notes

None.

---

## Example

# EffectiveConformance Property

---



Type	Default	Read Only	Description
[C#]PdfConformance			
[Visual Basic] PdfConformance	Pdf	Yes	The effective PdfConformance value.

This property indicates the effective conformance specified by the [Part](#), [Amd](#), and [Conformance](#) properties.

It can take any of the following values:

---

## Notes

- Pdf – PDF.
- PdfA1b – PDF/A-1b.
- PdfA1a – PDF/A-1a.

None.

---

## Example



## GetData Function

---

Get the conforming document as raw data.

### [C#]

```
byte[] GetData(Doc doc)
```

### [Visual Basic]

```
Function GetData(doc As Doc) As  
Byte()
```

---

#### Syntax

- may throw `Exception()`

---

#### Params

Name	Description
doc	The document.
return	The PDF document as an array of bytes.

---

#### Notes

The method writes the document in a conforming PDF format according to the [Conformance](#) property. Any conformance error is reported in the [Errors](#) property after the method finishes.

---

Example

None.

# GetStream Function

---



Get the conforming document as raw data stream.

## [C#]

```
Stream GetStream(Doc doc)
```

## [Visual Basic]

```
Function GetStream(doc As Doc) As Stream
```

- may throw `Exception()`

---

### Syntax

---

### Params

Name	Description
doc	The document.
return	The PDF document as a stream.

The method writes the document in a conforming PDF format according to the [Conformance](#) property. Any conformance error is reported in the [Errors](#) property after the method finishes.

Because of the CLR limit of 2 GB per object, the

---

**Notes** `GetData` method cannot return the data for a document larger than 2 GB. Use this method for documents larger than 2 GB. Dispose of the returned stream as soon as it is no longer needed for small memory footprint.

---

**Example**

None.



## Save Function

---

Write the conforming PDF document.

### [C#]

```
void Save(Doc doc, string path)
void Save(Doc doc, Stream stream)
```

### [Visual Basic]

#### Syntax

```
Sub Save(doc As Doc, path As String)
Sub Save(doc As Doc, stream As Stream)
```

- may throw `Exception()`

#### Params

Name	Description
doc	The document.
path	The destination file path.
stream	The destination stream.

#### Notes

This method writes the document in a conforming PDF format according to the [Conformance](#) property. Any conformance error is reported in the [Errors](#) property after the method finishes.

Here we save a document in PDF/A-1b format.

### [C#]

```
Doc theDoc = ...;
string thePath = Server.MapPath("pdfa.pdf")
using(PdfConformityOperation theOperation =
PdfConformityOperation()) {
    theOperation.Conformance =
PdfConformance.PdfA1b;
    theOperation.Save(theDoc, thePath);

    if(theOperation.Errors.Count>0) {
        Console.WriteLine("Errors:");
        for(int i = 0; i<theOperation.Errors.Co
++i)
            Console.WriteLine(theOperation.Errors
    }
}
```

**Example**

### [Visual Basic]

```
Dim theDoc As Doc = ...
Dim thePath As String =
Server.MapPath("pdfa.pdf")
Using theOperation As New PdfConformityOper
    theOperation.Conformance =
PdfConformance.PdfA1b
    theOperation.Save(theDoc, thePath)

    If theOperation.Errors.Count > 0 Then
        Console.WriteLine("Errors:")
        For i As Integer = 0 To
theOperation.Errors.Count - 1
            Console.WriteLine(theOperation.Errors
        Next
    End If
```

End Using

# Conformance Property

---



Type	Default	Read Only	Description
[C#]PdfConformance			
[Visual Basic] PdfConformance	Pdf	No	The PDF conformance.

This property specifies the conformance of the output PDF.

It can take any of the following values:

---

## Notes

- Pdf – PDF.
- PdfA1b – PDF/A-1b.
- PdfA1a – PDF/A-1a.

See the [Save](#) method.

---

## Example

# Messages Property

---



Type	Default	Read Only	Description
[C#]IList<string> [Visual Basic]IList(Of String)	null	Yes	The messages for writing in the conforming PDF format.

This property holds the messages for writing in the conforming PDF format. They describe changes made to the document.

---

## Notes

None.

---

## Example

# Conformity Property

---



Type	Default	Read Only	Description
[C#]PdfConformity			
[Visual Basic]PdfConformity	None	No	The PDF conformity.

This property specifies the conformity operations to perform. They provides more precise specifications than [Conformance](#).

It can take a combination of the following values:

---

#### Notes

- None – no operation.
- NoJavaScriptAction – removes JavaScript actions. (included in PDF/A conformance.)

None.

---

#### Example

# Errors Property

---



Type	Default	Read Only	Description
[C#]IList<string> [Visual Basic]IList(Of String)	null	Yes	The errors for writing in the conforming PDF format.

This property holds the errors for writing in the conforming PDF format.

---

## Notes

See the [Save](#) method.

---

## Example



# Read Function

---

Read and validate an existing document.

## [C#]

```
Doc Read(string path, XReadOptions options)  
Doc Read(byte[] data, XReadOptions options)  
Doc Read(Stream stream, XReadOptions optior
```

## [Visual Basic]

```
Function Read(path As String, options As  
XReadOptions) As Doc  
Function Read(data() As Byte, options As  
XReadOptions) As Doc  
Function Read(stream As Stream, options As  
XReadOptions) As Doc
```

Syntax

- may throw Exception()

Params

Name	Description
path	The file path to PDF document.
data	The source PDF data.
stream	The source PDF stream.
options	The settings for the read. (May be null.)

This method reads and validates the document according to the [Conformance](#) property. Any conformance error or warning is reported in the [Errors](#) property or the [Warnings](#) property and the method finishes.

---

**Notes** If the [Doc](#) property is null, the document is read into a new object, which is returned; otherwise, the document is read in the value of the [Doc](#) property, which is returned.

Here we validate a document against PDF/A-1b format.

**[C#]**

```
string thePath = Server.MapPath("pdfa.pdf")
using(PdfValidationOperation theOperation =
PdfConformityOperation()) {
    theOperation.Conformance =
PdfConformance.PdfA1b;
    Doc theDoc = theOperation.Read(thePath, r
theDoc.Dispose());

    if(theOperation.Errors.Count>0) {
        Console.WriteLine("Errors:");
        for(int i = 0; i<theOperation.Errors.Co
++i)
            Console.WriteLine(theOperation.Errors
    }
    if(theOperation.Warnings.Count>0) {
        if(theOperation.Errors.Count>0)
            Console.WriteLine();
        Console.WriteLine("Warnings:");
        for(int i = 0; i<theOperation.Warnings.
++i)
            Console.WriteLine(theOperation.Warnir
    }
}
```

```
}
```

---

**Example**

### [Visual Basic]

```
Dim thePath As String = Server.MapPath("pdf")
Using theOperation As New PdfConformityOperation
    theOperation.Conformance = PdfConformance
    Dim theDoc As Doc = theOperation.Read(thePath)
    Nothing)
    theDoc.Dispose()

    If theOperation.Errors.Count > 0 Then
        Console.WriteLine("Errors:")
        For i As Integer = 0 To
theOperation.Errors.Count - 1
            Console.WriteLine(theOperation.Errors(i))
        Next
    End If
    If theOperation.Warnings.Count > 0 Then
        If theOperation.Errors.Count > 0 Then
            Console.WriteLine()
        End If
        Console.WriteLine("Warnings:")
        For i As Integer = 0 To
theOperation.Warnings.Count - 1
            Console.WriteLine(theOperation.Warnings(i))
        Next
    End If
End Using
```

# Conformance Property

---



Type	Default	Read Only	Description
[C#]PdfConformance			
[Visual Basic]PdfConformance	Pdf	No	The PDF conformance.

This property specifies the conformance for validation.

It can take any of the following values:

---

## Notes

- Pdf – PDF.
- PdfA1b – PDF/A-1b.
- PdfA1a – PDF/A-1a.

See the [Read](#) method.

---

## Example

# Doc Property

---



Type	Default	Read Only	Description
[C#]Doc			
[Visual Basic]Doc	Pdf	No	The Doc object into which the document is read.

This property specifies the target Doc object into which the document is read. If it is null, the [Read](#) method returns a new Doc object.

---

## Notes

None.

---

## Example

# Errors Property

---



Type	Default	Read Only	Description
[C#]IList<string>			
[Visual Basic] IList(Of String)	null	Yes	The validation errors.

This property holds the validation errors.

If any errors are emitted, then the document is not compliant.

Each PDF/A error indicates the problem and also the part of the specification which is not being adhered to.

Some errors indicate problem with the PDF rather than the PDF/A specification. In this case, no specification section is emitted.

For example, the following errors might be emitted.

---

## Notes

Outline item 88's required Count is missing.

PDF/A-1 6.3.5 Font subsets: Type-0 font 198 is a font subset without CIDSet.

PDF/A-1 6.3.6 Font metrics: TrueType font 201's Widths is inconsistent

with the embedded font program.  
PDF/A-1 6.7.3 Document information  
dictionary: Metadata 15--Document  
"Producer" and XMP property  
pdf:Producer are different.

See the [Read](#) method.

---

**Example**

# Warnings Property

---



Type	Default	Read Only	Description
[C#]IList<string> <b>[Visual Basic]</b> IList(Of String)	null	Yes	The validation warnings.

This property holds the validation warnings.

Validation warnings indicate unexpected characteristics that neither PDF/A nor PDF regards as problems but ABCpdf regards as suspect.

For example, if the PDF header indicates PDF Version 1.4 but then a 1.5 feature is referenced in the PDF, a warning will be emitted.

Warnings are outside the PDF/A spec and so a warning does not indicate any problem with PDF/A compliance.

For example, the following warnings might be emitted.

---

## Notes

```
Type-2 CIDFont descriptor 198 uses
PDF 1.5 FontFamily.
Layout attribute object 187 uses PDF
1.5 BackgroundColor.
Layout attribute object 187 uses PDF
```

```
1.5 BorderColor.  
Layout attribute object 187 uses PDF  
1.5 BorderStyle.  
Layout attribute object 187's  
BorderStyle is PDF 1.5 border style.
```

---

**Example**

See the [Read](#) method.



# PageContents Constructor

---

PageContents Constructor.

## [C#]

```
PageContents(Doc doc)
```

## [Visual Basic]

```
Sub New(doc As Doc)
```

### Syntax

- may throw `NullReferenceException()`

### Params

Name	Description
doc	The PDF Document

Create a PageContents to hold a set of pages that need to be analysed.

### Notes

If the doc is null then an exception will be raised.

None.

---

**Example**

# AddLayers Function

---



Add a particular set of Layer objects from a particular Page

## [C#]

```
void AddLayers(Page page, Layer[] layers)
```

---

## Syntax

## [Visual Basic]

```
Sub AddLayers(page As Page, layers() As Layer)
```

---

## Params

Name	Description
page	The page on which the provided layers are located.
layers	The layers to be added.

Add a particular set of Layer objects from a particular Page.

---

## Notes

This is an alternative to the [AddPages](#) overloads. It allows specific parts of a page to be added to the operation rather than requiring the operation to be applied to the entirety of a page.

---

**Example**

None.



# AddPages Function

---

Add pages to be processed.

## [C#]

```
void AddPages()  
void AddPages(int pageNumber)  
void AddPages(int  
startPageNumber, int  
endPageNumber)
```

## [Visual Basic]

Syntax

```
Sub AddPages()  
Sub AddPages(pageNumber As  
Integer)  
Sub AddPages(startPageNumber As  
Integer, endPageNumber As  
Integer)
```

- may throw Exception()

Params

Name	Description
pageNumber	The number of the page to be processed. This is a one based index.
startPageNumber	The number of the first page to be processed. This is a one based index.

endPageNumber	The number of the last page to be processed. This is a one based index.
---------------	---

This method adds pages that need to be processed. If the provided page numbers are incorrect then an exception will be raised.

The parameterless version of this function adds all the pages in the document. If you are only interested in a subsection of the document you can specify individual pages or a range of pages using the overloads. You can call this function multiple times with multiple sets of pages to get the precise selection that you require.

---

Notes

Adding pages is a fairly expensive operation. For this reason if you are performing multiple types of analysis on the same set of pages then you will probably want to share one PageContents object between them.

Note that the PageContents object is a cache of the current document content. Thus it may be invalidated by a call that remaps or replaces existing content in the document. For this reason you should not call [Doc.Save](#), [Doc.GetData](#) or [Doc.Flatten](#) while using an operation that contains a PageContents object.

---

See [TextOperation.Group](#).

## Example

# IncludeColor Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	false	No	Whether to include color information in the output.

- may throw `Exception()`

Whether to include color information in the output.

This property must not be changed after the [AddPages](#) method has been called. To do so will result in an exception being raised.

---

## Notes

None.

---

## Example

# IncludeAnnotations Property

---



Type	Default	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	true	No	Whether to include annotation and field text in the output.

- may throw `Exception()`

Whether to include annotation and field text in the output.

---

## Notes

This property must not be changed after the `AddPages` method has been called. To do so will result in an exception being raised.

---

## Example

None.

# RegenerateUnicode Property

---



Type	Default	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	true	No	Whether to attempt to regenerate missing Unicode tables from embedded fonts.

- may throw `Exception()`

Whether to attempt to regenerate missing Unicode tables from embedded fonts.

For details of the type of fix that is attempted on non-conforming fonts please see the [FontObject.RegenerateToUnicode](#) method.

---

## Notes

This property must not be changed after the [AddPages](#) method has been called. To do so will result in an exception being raised.

None.

---

## Example





# TextOperation Constructor

---

TextOperation Constructor.

## [C#]

```
TextOperation(Doc doc)
```

## [Visual Basic]

Syntax

```
Sub New(doc As Doc)
```

- may throw `Exception()`

Params

Name	Description
doc	The PDF Document

Notes

Create a `TextOperation` for text analysis. If the `doc` is null then an exception will be raised.

Example

See the [Group](#) function.





# GetText Function

---

Get all the text in the page contents.

## [C#]

```
string GetText()  
string GetText(XRect rect, int  
pageNumber)  
string GetText(XRect[] rects,  
int[] pageNumbers)
```

## [Visual Basic]

Syntax

```
Function GetText() As String  
Function GetText(rect As XRect,  
pageNumber As Integer) As String  
Function GetText(rects As  
XRect(), pageNumbers As  
Integer()) As String
```

- may throw Exception()

Params

Name	Description
rect	The rectangle from which to return text.
rects	A set of rectangles forming a union from which to return text.
pageNumber	The page number - a one based index.

Return	The text.
--------	-----------

This method returns the text of the content that has been assigned via the [PageContents](#). If the PageContents is null then an exception will be raised.

The overloads of this function allow you to select an area of interest in terms of an area defined by single rectangle or by a union of multiple rectangles. They also allow you to select a subset of pages from which text should be returned. This page array is a set of page numbers - one based indexes - for which order is not important. Passing null for these parameters defaults to the entire page and the entire set of available pages respectively.

---

#### Notes

The process of getting the text involves scanning through the text fragments in the document and joining them up dependent on their positions and sizes. The result is a standard string with line ends where the line ends in the document have been found.

If there are parts of this string that you are interested in you can use the offset and length of the selections in conjunction with the [Select](#) method to identify those portions within the document.

---

See the [Group](#) function.

## Example



# Select Function

---

Select a range of text in the document.

## [C#]

```
IList<TextFragment> Select(int  
startIndex, int length)
```

## [Visual Basic]

### Syntax

```
Function Select(startIndex As  
Integer, length As Integer) As  
IList<TextFragment>
```

- may throw Exception()

### Params

Name	Description
startIndex	The start index for the selection as related to the string returned by the GetText method.
length	The length for the selection as related to the string returned by the GetText method.
Return	Returns a list of find matches encompassing the selected text.

Select a range of text in the document. If `GetText` has not been called previously then an exception will be raised.

This function is purposely designed to be similar to the `String.Substring` function. The concept is that you can use the text provided via the `GetText` method to identify pieces of text you are interested in. Then using the offset and length of the pieces you can select the actual text in the document.

---

**Notes**

The selection specifies the precise set of text fragments in the document. Each fragment has an area, some text and relates back to a text drawing operation within the content stream of the page. Applying the `Group` function to these fragments can group connected fragments into connected parts.

See the `Group` function.

---

**Example**



## Group Function

---

Group a range of text fragments into a set of lines.

### [C#]

```
ICollection<TextGroup>  
Group(List<TextFragment> fragments)
```

---

### Syntax

### [Visual Basic]

```
Function Group(fragments As  
List<TextFragment>) As  
ICollection<TextGroup>
```

---

### Params

Name	Description
fragments	The fragments to be grouped.
Return	Returns a list of groups defining the words.

Group a range of text fragments into a set of lines.

Calling Select will return a list of TextFragments corresponding to the section of text you are interested in. However each TextFragment may only make up a small portion of a word. Calling this function allows you to group connected fragments into continuous sections with a common rectangular area.

---

**Notes** Strictly speaking this grouping does not always correspond to lines. For example two fragments on the same line, separated by a large distance, will not be considered contiguous. However for most purposes the two broadly correspond.

Here we highlight a set of words in a source document by drawing a rectangle around each one.

### [C#]

```
string theSrc =
Server.MapPath("Acrobat.pdf");
string theDst =
Server.MapPath("HighlightedText.pdf");
string searchString = "Acrobat";
using (Doc theDoc = new Doc()) {
    theDoc.Read(theSrc);
    TextOperation op = new
TextOperation(theDoc);
    op.PageContents.AddPages();
    string text = op.GetText();
    int pos = 0;
    while (true) {
        pos = text.IndexOf(searchString, pos,
StringComparison.CurrentCultureIgnoreCase);
        if (pos < 0)
            break;
        IList<TextFragment> theSelection =
op.Select(pos, searchString.Length);
        IList<TextGroup> theGroups =
op.Group(theSelection);
        foreach (TextGroup theGroup in
theGroups) {
```

```

        theDoc.Rect.String =
theGroup.Rect.String;
        theDoc.FrameRect();
    }
    pos += searchString.Length;
}
theDoc.Save(theDst);
}

```

**Example**

### [Visual Basic]

```

Dim theSrc As String =
Server.MapPath("Acrobat.pdf")
Dim theDst As String =
Server.MapPath("HighlightedText.pdf")
Dim searchString As String = "Acrobat"
Using theDoc As New Doc()
    theDoc.Read(theSrc)
    Dim op As New TextOperation(theDoc)
    op.PageContents.AddPages()
    Dim text As String = op.GetText()
    Dim pos As Integer = 0
    While True
        pos = text.IndexOf(searchString, pos,
StringComparison.CurrentCultureIgnoreCase)
        If pos < 0 Then
            Exit While
        End If
        Dim theSelection As IList(Of
TextFragment) = op.[Select](pos,
searchString.Length)
        Dim theGroups As IList(Of TextGroup) =
op.Group(theSelection)
        For Each theGroup As TextGroup In
theGroups
            theDoc.Rect.[String] = theGroup.Rect.
[String]

```

```
        theDoc.FrameRect()  
    Next  
    pos += searchString.Length  
End While  
theDoc.Save(theDst)  
End Using
```

# Hyphenation Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to de-hyphenate words that appear to be split across two lines.

Whether to de-hyphenate words that appear to be split across two lines.

---

## Notes

If de-hyphenation is enabled then the hyphen will be removed and the second half of the hyphenated word will be migrated to the end of the previous line.

---

## Example

None.

# NativeColors Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to provide native colors such as CMYK, separations and spot colors, or whether to convert all colors to RGB

---

## Notes

Whether to provide native colors such as CMYK, separations and spot colors, or whether to convert all colors to RGB.

---

## Example

None.

# PageContents Property

---



Type	Default	Read Only	Description
[C#] PageContents	n/a	No	The pages to be operated upon.
[Visual Basic] PageContents			

This property specifies the pages to be operated upon.

Adding pages to a PageContents object can be a costly procedure taking a noticeable amount of time.

---

## Notes

So if you are performing a set of analysis operations on the same pages it can be more efficient to assign the PageContents from one to another rather than repeatedly re-populate from the original document.

None.

---

## Example

# ShowArtifactText Property

---



Type	Default	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to show text content that is marked as an artifact.

Whether to show text content that is marked as an artifact.

By default all text is returned, including text that may not be relevant to the user because it is tagged as an artifact. In cases where you are extracting text you may wish to ignore artifacts since, by definition, they are irrelevant. In cases where you are changing text you typically want to include artifacts as, while they offer no semantic information, they still have an appearance that may need updating.

---

## Notes

Using this setting you control whether you want this text returned by [GetText](#).

None.

---

## Example



# ShowClippedText Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to show text which is invisible because it is affected by a clip path.

Whether to show text which is invisible because it is affected by a clip path.

---

## Notes

By default all text is returned. This includes text that may not be visible to the user because it has been affected by a clipping path. Using this setting you control whether you want this text returned by [GetText](#).

---

## Example

None.

# ShowObscuredText Property

---



Type	Default	Read Only	Description
[C#] bool			Whether to show overlapping repeated text content.
[Visual Basic] Boolean	false	No	Whether to show overlapping repeated text content.

Whether to show overlapping repeated text content.

Overlapping repeated text content is most typically used for a synthetic bold effect. While the multiple copies of the text provide an appropriate bold-like look, they are not semantically valid since they appear as one unit. In cases where you are looking at the semantic meaning of the text, you typically only want the initial text item. However if you are replacing text then you may wish to include all the text items so that when you change one you also change the others.

---

## Notes

Using this setting you control whether you want this text returned by [GetText](#).

None.

---

## Example



# Substitutions Property

---



Type	Default	Read Only	Description
[C#] IDictionary<char, string>	n/a	No	A set of character to string substitutions used for translating typographic characters like ligatures into more normal text.
[Visual Basic] IDictionary<char, string>			

A set of character to string substitutions used for translating typographic characters like ligatures into more normal text.

PDF text often includes typographic characters that are uncommon in normal text. For example these may include ligatures such as 'ff' or 'fl'. They may include different types of dash such as n-dash and m-dash. They may include curly/smart single and double quote characters. For the purposes of text extraction and searching it is generally desirable to replace these characters with their more common equivalents.

---

## Notes

The default settings for this operation include the substitutions of the types of characters detailed

above. The default settings are liable to change so if it is important that your settings remain constant between different versions of ABCpdf you should assign your own values.

---

**Example**

None.

# TabAffinity Property

---



Type	Default Value	Read Only	Description
[C#] double [Visual Basic] Double	1.0	No	The minimum distance at which two text fragments will be assumed to be part of separate tab groups

The minimum distance at which two text fragments will be assumed to be part of separate tab groups.

---

## Notes

This value is specified in terms of a fraction of the current font size.

---

## Example

None.

# TabChar Property

---



Type	Default Value	Read Only	Description
[C#] char [Visual Basic] Char	Space (0x20)	No	The character used to separate out tab groups when GetText is called

The character used to separate out tab groups when GetText is called.

---

## Notes

None.

---

## Example

# TextObjects Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to get information on the BT and ET text object markers used to contain and group text operators

---

## Notes

Whether to get information on the BT and ET text object markers used to contain and group text operators.

---

## Example

None.

# WordAffinity Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	0.15	No	The minimum distance at which two fragments will be assumed to be part of one undivided word.

The minimum distance at which two text fragments will be assumed to be part of one undivided word. This value is specified in terms of a fraction of the font size.

PDF text does not always include spaces. It is not uncommon for spaces simply to be indicated by the positions at which characters are drawn. The fact that two text fragments are drawn some distance from each other visually indicates that there is a space between them. When extracting text from a PDF one needs to decide how far apart text fragments can be, before one assumes that they should be separated by a space.

---

## Notes

This property allows you to control this spacing. Increasing the property will tend to reduce the cohesiveness of fragments resulting in fewer spaces. Decreasing it will tend to increase the cohesiveness resulting in more spaces.

Broadly speaking the property is defined as the proportion of the font size at which distance a space will be assumed to be present. The precise mechanism is a little more complicated than this as it needs to take account of vertical distance and of varying font sizes between fragments. However for most purposes the broad definition will be acceptable.

---

**Example**

None.

# Font Property

---



Type	Default	Read Only	Description
[C#] FontObject	n/a	Yes	The font object used for drawing this text fragment.
[Visual Basic] FontObject			

The font object used for drawing this text fragment.

---

## Notes

The font object can be used to find information about the font.

---

## Example

None.

# FontColor Property

---



Type	Default	Read Only	Description
[C#] XColor			
[Visual Basic] XColor	n/a	Yes	The color used for drawing this text fragment.

The color used for drawing this text fragment.

Note that the [PageContents.IncludeColor](#) property must be set for this to be available. This is because processing color information adds a noticeable overhead that can be avoided for many text operations. If the property is not set then all text will be reported as black.

---

## Notes

None.

---

## Example

# FontSize Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The effective font size used for drawing this text fragment.

The effective font size used for drawing this text fragment.

The effective size is the normal measure that you will want to use as it describes the size the font appears on the page. It combines the effect of the specified font size with any transformation matrices which may enlarge or shrink the text.

---

## Notes

None.

---

## Example

# FontObliqueAngle Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The skew angle of the font as used for oblique styles to simulate italic.

The skew angle of the font as used for oblique styles to simulate italic. Measured in degrees clockwise from the vertical.

An italic style is not the same as an oblique style though the two are often confused. An italic style contains different glyphs to a regular style and these glyphs often appear slanted. If a true italic style is not available one can be synthesized by applying a skew to tilt the characters to the right. This is known as an oblique style.

---

## Notes

A typical skew angle is between ten and fifteen degrees. So, while it is an arbitrary judgment, any angle above about eight degrees may be regarded as an oblique style.

---

None.

**Example**

# PageID Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The ID of the Page on which this text fragment is located.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The ID of the Page on which this text fragment is located.

---

## Notes

You can assign this ID to the [Doc.Page](#) or you can use it with the [ObjectSoup](#) to retrieve a [Page](#) object.

---

## Example

None.

# StreamID Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	n/a	Yes	The Stream ID of the content stream in which this text fragment is located.

The Stream ID of the content stream in which this text fragment is located.

---

## Notes

You can use this ID with the [ObjectSoup](#) to retrieve a [StreamObject](#).

---

## Example

None.

# StreamOffset Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	n/a	Yes	The offset within the content stream to the start of the drawing operation that contains this fragment.

The offset within the content stream to the start of the drawing operation that contains this fragment.

---

## Notes

Each TextFragment spans a part of a PDF stream drawing operator. This property provides the offset within the uncompressed stream, to the start of that drawing operator.

---

## Example

None.

# StreamLength Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			The length within the content stream of the drawing operation that contains this fragment.
[Visual Basic] <code>Integer</code>	n/a	Yes	

The length within the content stream of the drawing operation that contains this fragment.

---

## Notes

Each TextFragment spans a part of a PDF stream drawing operator. This property provides the offset within the uncompressed stream, to the start of that drawing operator.

---

## Example

None.

# TextSpanIndex Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	n/a	Yes	The zero based index of the drawing operator text array item that contains this fragment. For non text array operators this value is zero.

The zero based index of the drawing operator text array item that contains this fragment. For non text array operators this value is zero.

Each TextFragment spans a part of a PDF stream drawing operator. The part may be the entire of the text drawn by the operator or it may be a section of the text within that operator.

Some text drawing operators allow multiple items of text to be drawn. This provides a useful shorthand to allow character placement to be adjusted within a string. For example the following command will draw two items of text with a wide space (specified by the number) between the words.

---

## Notes

```
[ (Breaking) 4000 (Bad) ] TJ
```

Each TextFragment only corresponds to one of the strings within such an array. This property is a zero based index which tells you which one. For example in the above example, if the TextFragment corresponded to "Bad", then the TextSpanIndex would be equal to one.

For text drawing operations which do not involve arrays this value will always be zero.

---

**Example**

None.

# Rect Property

---



Type	Default	Read Only	Description
[C#] XRect			
[Visual Basic] XRect	n/a	Yes	The rectangle that contains the text of the fragment.

The rectangle that contains the text of the fragment.

Each TextFragment spans a part of a PDF stream drawing operator. The part may be the entire of the text drawn by the operator or it may be a section of the text within that operator.

---

## Notes

This property reflects the area covered by the TextFragment.

This rectangle is encoded in PDF coordinates rather than any abstracted coordinate space.

None.

---

## Example

# Rotation Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The angle of rotation of the fragment in degrees.

The angle of rotation of the fragment in degrees.

---

Notes

None.

---

Example

# Text Property

---



Type	Default	Read Only	Description
[C#] string [Visual Basic] String	n/a	Yes	The text of the fragment. A fragment may span only a portion of the complete text drawing operator.

The text of the fragment. A fragment may span only a portion of the complete text drawing operator.

Each TextFragment spans a part of a PDF stream drawing operator. The part may be the entire of the text drawn by the operator or it may be a section of the text within that operator.

---

## Notes

This property reflects the text of the TextFragment - the selected portion within the drawing operator.

None.

---

## Example



# RawText Property

---



Type	Default	Read Only	Description
[C#] string			The text specified in the drawing operator or the text array item of the drawing operator.
[Visual Basic] String	n/a	Yes	

The text of the drawing operator.

Each TextFragment spans a part of a PDF stream drawing operator. The part may be the entire of the text drawn by the operator or it may be a section of the text within that operator.

---

## Notes

This property reflects the entire text of the drawing operator. If the drawing operator is a text array operator then this property reflects the text of the portion of the array referenced by the [TextSpanIndex](#) property.

None.

---

## Example



# FontColorSpace Property

---



Type	Default Value	Read Only	Description
[C#] ColorSpace	n/a	Yes	The ColorSpace used for drawing this text fragment
[Visual Basic] ColorSpace			

The ColorSpace used for drawing this text fragment.

---

## Notes

None.

---

## Example

# PageID Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The ID of the Page on which this group is located.

The ID of the Page on which this group is located.

---

## Notes

You can assign this ID to the [Doc.Page](#) or you can use it with the [ObjectSoup](#) to retrieve a [Page](#) object.

---

## Example

None.

# Rect Property

---



Type	Default	Read Only	Description
[C#] <code>XRect</code>			
[Visual Basic] <code>XRect</code>	n/a	Yes	The rectangle that contains the group.

The rectangle that contains the group.

---

## Notes

This rectangle is encoded in PDF coordinates rather than any abstracted coordinate space.

---

## Example

None.

# Text Property

---



Type	Default	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The text of the group.

The text of the group.

---

## Notes

The text of the group is not necessarily the same as the concatenated text of the TextFragments as join characters such as spaces may be inserted where characters are separated by large gaps.

---

## Example

None.

# TextFragments Property

---



Type	Default	Read Only	Description
[C#] IList<TextFragment>	n/a	Yes	The text fragments in the group.
[Visual Basic] IList<TextFragment>			

The text fragments in the group.

---

## Notes

.

None.

---

## Example



# ImageOperation Constructor

---

ImageOperation Constructor.

## [C#]

```
ImageOperation(Doc doc)
```

---

### Syntax

## [Visual Basic]

```
Sub New(doc As Doc)
```

---

### Params

Name	Description
doc	The PDF Document

---

### Notes

Create a ImageOperation for text analysis. If the doc is null then an exception will be raised.

---

### Example

None.

# GetImageProperties Function

---



Get the image information for all the raster images.

## [C#]

```
ICollection<ImageProperties>  
GetImageProperties()
```

## [Visual Basic]

### Syntax

```
Function GetImageProperties() As  
ICollection<ImageProperties>
```

- may throw Exception()

### Params

Name	Description
Return	A collection of all the image properties..

Get the image information for all the raster images. If the [PageContents](#) is null then an exception will be raised.

After retrieving image properties you can iterate through them to find out information about what images there are and how they are placed in the document. Much like HTML, PDF allows a

### Notes

divorce between the image and the placement of that image. So you may have one PixMap which is drawn multiple times at multiple sizes in different locations within the document.

Here we highlight a set of images in a source document by drawing a red rectangle around each one.

### [C#]

```
string theSrc =
Server.MapPath("Acrobat.pdf");
string theDst =
Server.MapPath("HighlightedImages.pdf");
using (Doc theDoc = new Doc()) {
    theDoc.Read(theSrc);
    theDoc.Color.SetRgb(255, 0, 0);
    theDoc.Width = 0.1;
    ImageOperation op = new
ImageOperation(theDoc);
    op.PageContents.AddPages();
    ICollection<ImageProperties> images =
op.GetImageProperties();
    foreach (ImageProperties pl in images)
    {
        foreach (ImageRendition plc in
pl.Renditions) {
            plc.Focus();
            theDoc.FrameRect();
        }
    }
    theDoc.Save(theDst);
}
```

---

Example

### [Visual Basic]

---

```
Dim theSrc As String =
Server.MapPath("Acrobat.pdf")
Dim theDst As String =
Server.MapPath("HighlightedImages.pdf")
Using theDoc As New Doc()
    theDoc.Read(theSrc)
    theDoc.Color.SetRgb(255, 0, 0)
    theDoc.Width = 0.1
    Dim op As New ImageOperation(theDoc)
    op.PageContents.AddPages()
    Dim images As ICollection(Of
ImageProperties) =
op.GetImageProperties()
    For Each pl As ImageProperties In
images
        For Each plc As ImageRendition In
pl.Renditions
            plc.Focus()
            theDoc.FrameRect()
        Next
    Next
    theDoc.Save(theDst)
End Using
```

# IncludeAll Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to include all images in the analysis to allow the detection of orphans.

Whether to include all images in the analysis to allow the detection of orphans.

By default the ImageOperation will go through the pages defined in the [PageContents](#) looking at all the images that are used. However it is possible to include an image in a document yet never use it. This kind of orphan will not be detected using this method.

---

## Notes

To allow the detection of orphans this property can be used to search the entire document and reference any image that exists. If such an image is an orphan this will be revealed because it will appear in the analysis as an [ImageProperties](#) item with no [ImageRenditions](#).

---

None.

**Example**

# PageContents Property

---



Type	Default	Read Only	Description
[C#] PageContents	n/a	No	The pages to be operated upon.
[Visual Basic] PageContents			

This property specifies the pages to be operated upon.

Adding pages to a PageContents object can be a costly procedure taking a noticeable amount of time.

---

## Notes

So if you are performing a set of analysis operations on the same pages it can be more efficient to assign the PageContents from one to another rather than repeatedly re-populate from the original document.

None.

---

## Example

# PixelFormat Property

---



Type	Default	Read Only	Description
[C#] <a href="#">PixelFormat</a>			
[Visual Basic] <a href="#">PixelFormat</a>	n/a	Yes	The <a href="#">PixelFormat</a> object associated with the image.

The [PixelFormat](#) object associated with the image.

---

## Notes

None.

---

## Example

# Renditions Property

---



Type	Default	Read Only	Description
[C#] <code>IList&lt;ImageRendition&gt;</code>	n/a	Yes	The set of renditions of the image at different locations within the document.
[Visual Basic] <code>IList&lt;ImageRendition&gt;</code>			

The set of renditions of the image at different locations within the document.

---

## Notes

Much like HTML, PDF allows a divorce between the image and the placement of that image. So you may have one bitmap which is drawn multiple times at multiple sizes in different locations within the document.

---

## Example

None.

# Dpi Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The resolution of the image in dots per inch.

The resolution of the image in dots per inch.

---

Notes

None.

---

Example

# DpiX Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The horizontal resolution of the image in dots per inch.

---

## Notes

The horizontal resolution of the image in dots per inch.

---

## Example

None.

# DpiY Property

---



Type	Default	Read Only	Description
[C#] double			The vertical resolution of the image in dots per inch.
[Visual Basic] Double	n/a	Yes	

The vertical resolution of the image in dots per inch.

---

Notes

None.

---

Example



## Focus Function

---

Focus the document on the location of the image placement.

### [C#]

```
void Focus()
```

---

### Syntax

### [Visual Basic]

```
Sub Focus()
```

---

### Params

Name	Description
none	

Focus the document on the location of the image placement.

After calling this function you can add content overlaying the image rendition. For example you might call [Doc.FrameRect](#) to frame it.

---

### Notes

Because images are always drawn using a transformation matrix rather than a Rect you will probably need to reduce the size of any lines or text to values smaller than one.

---

**Example**

None.

# PageID Property

---



Type	Default	Read Only	Description
[C#] <code>int</code>			
[Visual Basic] <code>Integer</code>	n/a	Yes	The ID of the Page on which the image is placed.

The ID of the Page on which the image is placed.

---

## Notes

You can assign this ID to the [Doc.Page](#) or you can use it with the [ObjectSoup](#) to retrieve a [Page](#) object.

---

## Example

None.

# Matrix Property

---



Type	Default	Read Only	Description
[C#] XTransform	n/a	Yes	The transformation matrix for the placement of the image.
[Visual Basic] XTransform			

The transformation matrix for the placement of the image.

---

## Notes

Images in PDFs are always drawn at the origin with a width and height of zero. It is the transformation matrix which defines their placement on the page.

---

## Example

None.

# BoundingBox Property

---



Type	Default	Read Only	Description
[C#] XRect			The bounding rectangle for the placement of the image.
[Visual Basic] XRect	n/a	Yes	

The bounding rectangle for the placement of the image.

---

## Notes

Note that images may be rotated or otherwise transformed. For this reason consider whether the Matrix property or the Focus method might be more appropriately used in place of the BoundingBox property.

---

## Example

None.

# Dpi Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The resolution of the image in dots per inch.

The resolution of the image in dots per inch.

---

Notes

None.

---

Example

# DpiX Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The horizontal resolution of the image in dots per inch.

---

## Notes

The horizontal resolution of the image in dots per inch.

---

## Example

None.

# DpiY Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	n/a	Yes	The vertical resolution of the image in dots per inch.

The vertical resolution of the image in dots per inch.

---

Notes

None.

---

Example

# StreamObject Property

---



Type	Default	Read Only	Description
[C#] StreamObject	n/a	Yes	The StreamObject in which the image draw operation is contained.
[Visual Basic] StreamObject			

The StreamObject in which the image draw operation is contained.

---

## Notes

The combination of the StreamObject, StreamOffset and StreamLength allows you to precisely locate the image draw command sequence in the PDF content stream. This can then be used to modify or delete this particular command.

---

## Example

None.

# StreamOffset Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Int32</code>	n/a	Yes	The offset within the uncompressed StreamObject to the start of the drawing operation that contains this image draw operation.

The offset within the uncompressed StreamObject to the start of the drawing operation that contains this image draw operation.

---

## Notes

The combination of the StreamObject, StreamOffset and StreamLength allows you to precisely locate the image draw command sequence in the PDF content stream. This can then be used to modify or delete this particular command.

---

## Example

None.



# StreamLength Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Int32</code>	n/a	Yes	The length within the uncompressed StreamObject of the drawing operation that contains this image draw operation.

The length within the uncompressed StreamObject of the drawing operation that contains this image draw operation..

---

## Notes

The combination of the StreamObject, StreamOffset and StreamLength allows you to precisely locate the image draw command sequence in the PDF content stream. This can then be used to modify or delete this particular command.

---

## Example

None.



# StreamID Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code>			The ID of the Stream in which the image draw operation is contained
[Visual Basic] <code>Integer</code>	n/a	Yes	

The ID of the Stream in which the image draw operation is contained.

---

## Notes

None.

---

## Example



# FlattenTransparencyOperation Constructor

---

FlattenTransparencyOperation Constructor.

## [C#]

```
FlattenTransparencyOperation()
```

---

## Syntax

## [Visual Basic]

```
Sub New()
```

---

## Params

Name	Description
none	

---

## Notes

Create a FlattenTransparencyOperation.

See the [FlattenTransparency](#) method.

---

## Example

Note that this feature is only available under the ABCpdf Professional License.



# FlattenTransparency Function

---

Flatten the transparency of pages in a document.

## [C#]

```
void FlattenTransparency(Doc doc)
void FlattenTransparency(Pages pages)
void FlattenTransparency(Page page)
```

## [Visual Basic]

### Syntax

```
Sub FlattenTransparency(doc As Doc)
Sub FlattenTransparency(pages As Pages)
Sub FlattenTransparency(page As Page)
```

- may throw Exception()

### Params

Name	Description
doc	The document containing transparency to be flattened.
pages	The pages whose transparency is to be flattened as referenced by a Pages IndirectObject.
page	The page whose transparency is to be flattened as referenced by a Page IndirectObject.

Flattens the transparent objects on pages in the document.

When transparent objects overlap other objects, all or a portion of the overlapped object shows through. FlattenTransparency will create new objects that represent the portion of the two objects that do not overlap. Portions of transparent objects that don't overlap other objects are composited with the backdrop color (white) to create a fully opaque object that appears the same.

---

**Notes**

Note that this feature is only available under the ABCpdf Pro License.

Here we flatten all the transparent objects in a document.

**[C#]**

---

**Example**

```
FlattenTransparencyOperation transOp = new  
FlattenTransparencyOperation();  
transOp.DotsPerInch = 144;  
transOp.ColorSpace = XRendering.ColorSpaceType.DeviceRGB;  
  
Doc theDoc = new Doc();  
theDoc.Read(Server.MapPath("../mypics/sample.pdf"));  
transOp.Flatten(theDoc);  
theDoc.Save(Server.MapPath("../Flattened/sample.pdf"));
```

# Alpha Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	1.0	No	Sets the backdrop alpha

When objects are composited with the backdrop, this alpha value will be the minimum alpha of objects after compositing

---

## Notes

See the [Flatten](#) method.

---

## Example

# AntiAliasPolygons Property

---



Type	Default Value	Read Only	Description
[C#] bool			
[Visual Basic] Boolean	true	No	Whether to anti-alias polygons when creating synthetic image objects.

Determines whether polygons will be rendered with anti-aliased edges when creating synthetic image objects.

---

## Notes

Anti-aliasing is a technique for using gradients of color to eliminate jagged edges when objects are drawn. The object edges are blurred to reduce pixelation.

---

## Example

None

# AntiAliasText Property

---



Type	Default Value	Read Only	Description
[C#] bool			Whether to anti-alias text when creating a synthetic image object.
[Visual Basic] Boolean	true	No	

Determines whether text will be rendered with anti-aliased edges when creating a synthetic image object.

---

## Notes

Anti-aliasing is a technique for using gradients of color to eliminate jagged edges when objects are drawn. The object edges are blurred to reduce pixelation.

---

## Example

None

# ColorSpace Property

---



Type	Default Value
<b>[C#]</b> XRendering.ColorSpaceType	XRendering.ColorSpaceTyp
<b>[Visual Basic]</b> XRendering.ColorSpaceType	

All the objects will be converted to this colorspace. Also, compos during the creation of new objects will take place in this colorspa

---

## Notes

See the [Flatten](#) method.

---

## Example

# ConvertAnnotations Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	true	No	Gets or sets a value indicating whether annotations are to be flattened

Gets or sets a value indicating whether annotations are to be flattened.

---

## Notes

None.

---

## Example

# DotsPerInch Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	72.0	No	Sets the resolution of generated images.

When two complex objects overlap, such as a partially transparent image with a backdrop image, an image representing the overlap is generated to this resolution.

---

## Notes

See the [Flatten](#) method.

---

## Example

# IccCmyk Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	"standard"	No	The path to the default CMYK ICC color profile.

A path to the default CMYK ICC color profile.

The profile that will be used to convert any device CMYK specified in the PDF file to the device independent working color space. This may be necessary when colors in different color spaces need to be blended together.

This property can take a path to an icm file. However there are also two special values you can use. If the property takes the value "device" then the device color space will be used. If the property takes the value "standard" then a built in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set to "standard" or paths to color profiles. All color spaces are assumed to be device independent color spaces.

If this property is set to "device" then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set

to "device". All color spaces are all assumed to be device color spaces.

If this property is set to a file name with no path information, then the folder "`<windows>\system32\spool\drivers\color`" will be searched to locate the file.

---

**Example**

None

# IccGray Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	"standard"	No	The path to the default Gray ICC color profile.

A path to the default Gray ICC color profile.

The profile that will be used to convert any device Gray specified in the PDF file to the device independent working color space. This may be necessary when colors in different color spaces need to be blended together.

This property can take a path to an icm file. However there are also two special values you can use. If the property takes the value "device" then the device color space will be used. If the property takes the value "standard" then a built in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set to "standard" or paths to color profiles. All color spaces are assumed to be device independent color spaces.

If this property is set to "device" then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set

to "device". All color spaces are all assumed to be device color spaces.

If this property is set to a file name with no path information, then the folder "`<windows>\system32\spool\drivers\color`" will be searched to locate the file.

---

**Example**

None.

# IccRgb Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code>			
[Visual Basic] <code>String</code>	"standard"	No	The path to the default RGB ICC color profile.

A path to the default RGB ICC color profile.

The profile that will be used to convert any device RGB specified in the PDF file to the device independent working color space. This may be necessary when colors in different color spaces need to be blended together.

This property can take a path to an icm file. However there are also two special values you can use. If the property takes the value "device" then the device color space will be used. If the property takes the value "standard" then a built in default color profile will be used.

---

## Notes

If this property is set to "standard" or a path to a color profile then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set to "standard" or paths to color profiles. All color spaces are assumed to be device independent color spaces.

If this property is set to "device" then [IccRgb](#), [IccCmyk](#), [IccGray](#) and [IccOutput](#) should also be set

to "device". All color spaces are all assumed to be device color spaces.

If this property is set to a file name with no path information, then the folder "`<windows>\system32\spool\drivers\color`" will be searched to locate the file.

---

**Example**

None.

# Log Property

---



Type	Default Value	Read Only	Description
[C#] <code>string</code> [Visual Basic] <code>String</code>	None	Yes	Returns information from the last FlattenTransparency operation.

If for some reason an object was not able to be flattened, a reason will be indicated here.

---

## Notes

See the [Flatten](#) method.

---

## Example



# ReduceSizeOperation Constructor

---

ReduceSizeOperation Constructor.

## [C#]

```
ReduceSizeOperation(Doc doc)
```

## [Visual Basic]

### Syntax

```
Sub New(doc As Doc)
```

- may throw Exception()

### Params

Name	Description
doc	The PDF Document

### Notes

Create a ReduceSizeOperation to compact and compress a PDF document. If the doc is null then an exception will be raised.

### Example

See the [Compact](#) function.





# Compact Function

---

Compact and compress the document.

## [C#]

```
void Compact(bool wholeDocument)
```

## [Visual Basic]

Syntax

```
Function Compact(wholeDocument As Boolean)
```

- may throw `Exception()`

Params

Name	Description
wholeDocument	Whether to compact the entire document selected pages.

Compact and compress the document.

The two main types of resources which take up file space are images and embedded fonts. As such this method allows you to resize images to a lower resolution, compress them using compression methods and quality settings and remove embedded fonts. Particular options appropriate to your documents can be seen in the properties on this operation.

During processing [ProcessingObject](#) and [ProcessedObject](#)

---

**Notes**

will be fired for each font or image object that is being operated upon. Using the arguments included with these events you can get finer control over the application of this operation.

If you have previously added content to the document it will be frozen in place. This may result in changes to the [Doc.HtmlOptions](#), [Doc.Rendering](#) and [Doc.SaveOptions](#) properties.

You may also wish to set the [SaveOptions.CompressObject](#) property to true, to further reduce the output size on save.

This type of operations is fairly complex and can take some time on larger documents.

The following example shows how to compress a document.

**[C#]**

```
using (Doc doc = new Doc()) {
    doc.Read(Server.MapPath("../mypics/samplepics/sample.doc"));
    using (ReduceSizeOperation op = new ReduceSizeOperation(doc))
        op.Compact(true);
    doc.Save(Server.MapPath("ReduceSizeOperation.doc"));
}
```

---

**Example****[Visual Basic]**

```
Using doc As New Doc()
    doc.Read(Server.MapPath("../mypics/samplepics/sample.doc"))
    Using op As New ReduceSizeOperation(doc)
        op.Compact(True)
    End Using
    doc.Save(Server.MapPath("ReduceSizeOperation.doc"))
End Using
```



# CompressImages Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to resize and recompress images where possible.

Whether to resize and recompress images where possible.

---

## Notes

This option is used in conjunction with the image quality, resolution and compression settings.

---

## Example

None.

# CompressStreams Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to compress uncompressed streams where possible.

Whether to compress uncompressed streams where possible.

---

## Notes

Uncompressed streams are relatively uncommon in real world PDF documents but checking and compressing them is a simple and effective way to reduce size in the unusual situations in which they are found.

---

## Example

None.

# ColorImageCompression Property

---



Type	Default	Read Only	Description
[C#] CompressionType	Jpeg	No	The target compression type for the re-encoding of color images.
[Visual Basic] CompressionType			

The target compression type for the re-encoding of color images.

When the [CompressImages](#) setting is used this option is used to determine the type of compression to be used for images in this color space. Images are recompressed if they are resampled or if the target compression type is lossy. They are not recompressed if they already use the target compression and the compression type is lossless.

---

## Notes

None.

---

## Example

# ColorImageDpi Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	72.0	No	The target resolution for the resampling of color images.

The target resolution for the resampling of color images.

When the [CompressImages](#) setting is set this option is used to determine the resolution at images in this color space should be targeted. Images are scaled down if they are too large but not up if they are too small.

---

## Notes

None.

---

## Example

# ColorImageQuality Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	50	No	The target compression quality for the re-encoding of color images.

The target compression quality for the re-encoding of color images.

---

## Notes

When the [CompressImages](#) setting is set to a lossy type of compression such as JPEG, this option is used to determine the level of compression to be used when images are recompressed.

---

## Example

None.

# GrayImageCompression Property

---



Type	Default	Read Only	Description
[C#] CompressionType	Jpeg	No	The target compression type for the re-encoding of grayscale images.
[Visual Basic] CompressionType			

The target compression type for the re-encoding of grayscale images.

When the [CompressImages](#) setting is used this option is used to determine the type of compression to be used for images in this color space. Images are recompressed if they are resampled or if the target compression type is lossy. They are not recompressed if they already use the target compression and the compression type is lossless.

---

## Notes

None.

---

## Example

# GrayImageDpi Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	150.0	No	The target resolution for the resampling of grayscale images.

The target resolution for the resampling of grayscale images.

---

## Notes

When the [CompressImages](#) setting is set this option is used to determine the resolution at images in this color space should be targeted. Images are scaled down if they are too large but not up if they are too small.

---

## Example

None.

# GrayImageQuality Property

---



Type	Default	Read Only	Description
[C#] int [Visual Basic] Integer	50	No	The target compression quality for the re-encoding of grayscale images.

The target compression quality for the re-encoding of grayscale images.

---

## Notes

When the [CompressImages](#) setting is set to a lossy type of compression such as JPEG, this option is used to determine the level of compression to be used when images are recompressed.

---

## Example

None.

# MonochromeImageCompressionProperty

---



Type	Default	Read Only	Description
[C#] CompressionType	Ccitt	No	The target compression type for the re-encoding of monochrome images.
[Visual Basic] CompressionType			

The target compression type for the re-encoding of monochrome images. Most monochrome images are black and white.

When the [CompressImages](#) setting is used this option is used to determine the type of compression to be used for images in this color space. Images are recompressed if they are resampled or if the target compression type is lossy. They are not recompressed if they already use the target compression and the compression type is lossless.

---

## Notes

None.

---

## Example



# MonochromeImageDpi Property

---



Type	Default	Read Only	Description
[C#] double [Visual Basic] Double	150.0	No	The target resolution for the resampling of monochrome images.

The target resolution for the resampling of monochrome images. Most monochrome images are black and white.

---

## Notes

When the [CompressImages](#) setting is set this option is used to determine the resolution at images in this color space should be targeted. Images are scaled down if they are too large but not up if they are too small.

---

## Example

None.

# MonochromeImageQuality Property

---



Type	Default	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	50	No	The target compression quality for the re-encoding of monochrome images.

The target compression quality for the re-encoding of monochrome images.

---

## Notes

When the [CompressImages](#) setting is set to a lossy type of compression such as JPEG, this option is used to determine the level of compression to be used when images are recompressed.

---

## Example

None.

# PageContents Property

---



Type	Default	Read Only	Description
[C#] PageContents	n/a	No	The pages to be operated upon.
[Visual Basic] PageContents			

This property specifies the pages to be operated upon.

Adding pages to a PageContents object can be a costly procedure taking a noticeable amount of time.

So if you are performing a set of analysis operations on the same pages it can be more efficient to assign the PageContents from one to another rather than repeatedly re-populate from the original document.

---

## Notes

Note that the [UnembedComplexFonts](#) option normally results in document-wide changes. If this happens the PageContents may be invalidated and will have to be re-built if required again. While this happens invisibly it is a relatively expensive operation and for this reason it is a good idea to perform this type of operation last rather than first if you are performing a chain of operations.

---

**Example**

None.

# PalettizationTolerance Property

---



Type	Default	Read Only	Description
[C#] double			
[Visual Basic] Double	0.01	No	The amount of divergence from the target palette which will be allowed.

The amount of divergence from the target palette which will be allowed.

Currently the only palette that is supported is RGB black and white. If an RGB image contains only grayscale components, and those grayscale pixels fit into the black and white palette with less than the amount of divergence specified, then it will be converted to one bit grayscale.

---

## Notes

The tolerance is measured in terms of the average distance between the colors of pixels and the color they would be converted to in the palette. Because the maximum distance varies depending on the number of components in the color space, this distance is then rescaled to a value between zero and one. Zero means that the pixels and the palette need to match exactly. One means all pixels will always match.

---

**Example**

None.

# RefactorImages Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to refactor and remove duplicate images where possible.

Whether to refactor and remove duplicate images where possible.

---

## Notes

PDF documents sometimes contain duplicate images. This method allows you to detect such duplicates and resolve them into one reference..

---

## Example

None.

# UnembedComplexFonts Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to unembed complex Unicode based fonts where possible.

Whether to unembed complex Unicode based fonts where possible.

---

## Notes

This option is relatively complex and expensive but this type of embedded font tends to be large so the size reduction obtained can be significant.

---

## Example

None.

# UnembedCorruptFonts Property

---



Type	Default Value	Read Only	Description
[C#] <code>bool</code> [Visual Basic] <code>Boolean</code>	true	No	Whether to unembed embedded fonts that appear to be corrupt or non-standard

Whether to unembed embedded fonts that appear to be corrupt or non-standard.

---

## Notes

None.

---

## Example

# UnembedSimpleFonts Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to unembed simple Latin based fonts where possible.

Whether to unembed simple Latin based fonts where possible.

---

## Notes

This option is relatively fast and low impact but embedded Latin fonts tend not to be too large.

---

## Example

None.

# UnembedUnusualFonts Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	false	No	Whether to unembed embedded fonts that do not have an obvious substitute on the local machine.

Whether to unembed embedded fonts that do not have an obvious substitute on the local machine.

The difficulty of providing an appropriate substitute font depends greatly on how unusual the font is. Substitutes are typically rather normal fonts so if you have a font like "Transport Medium" (the font used for all UK road signs) then a PDF display package should be able to provide a fairly sensible substitute. However if you have a font like "Fire and Ice" it is likely that the substitute will not be terribly appropriate. The most extreme example of this comes with barcode fonts which of course look completely different from the characters that they represent.

---

## Notes

It is difficult to know what fonts are unusual. However if the font does not appear on the local machine then it is a reasonable assumption that it may not substitute well. As such this property allows you to

avoid unembedding such fonts.

---

**Example**

None.



# AccessibilityOperation Constructor

---

AccessibilityOperation Constructor.

## [C#]

```
AccessibilityOperation(Doc doc)
```

## [Visual Basic]

Syntax

```
Sub New(doc As Doc)
```

- may throw Exception()

Params

Name	Description
doc	The PDF Document

Notes

Create an AccessibilityOperation for accessibility. If the doc is null then an exception will be raised.

Here we read an existing PDF and make it accessible.

We produce output conformant to the PDF specification and compliant with use within Acrobat. However please note that Accessibility is not well supported outside Acrobat.

in particular some versions of software such as NVDAccess not necessarily handle all constructs correctly. In particular have problems with form XObjects and so one needs to avoid type of construct.

The code below calls Page.StampFormXObjects on all pages of the document to ensure that all form XObjects are removed before the document is made accessible.

The process of stamping form XObjects will not normally make much difference. However in some cases you may find that the level of expansion in size and very occasionally you may see subtle differences in transparency blending. Nevertheless if you want compatibility, there is no other choice.

### [C#]

Example

```
Doc doc = new Doc();
doc.Read("spaceshuttle.pdf");
Page[] pages =
doc.ObjectSoup.Catalog.Pages.GetPageArrayAll();
foreach (Page page in pages)
    page.StampFormXObjects(true); // see note
NVDA above
AccessibilityOperation op = new
AccessibilityOperation(doc);
op.PageContents.AddPages();
op.MakeAccessible();
doc.Save("accessible.pdf");
```

### [Visual Basic]

```
Dim doc as New Doc
doc.Read("spaceshuttle.pdf")
Page() pages =
doc.ObjectSoup.Catalog.Pages.GetPageArrayAll()
foreach (Page page in pages)
    page.StampFormXObjects(true) ' see notes
NVDA above
```

```
Dim op As New AccessibilityOperation(doc)
op.PageContents.AddPages()
op.MakeAccessible()
doc.Save("accessible.pdf")
```



# MakeAccessible Function

---

Tags the document for accessibility.

## [C#]

```
void MakeAccessible()
```

## [Visual Basic]

```
Sub MakeAccessible()
```

- may throw Exception()

Syntax

Params

Name	Description
Return	The text.

This function scans the document, performs a semantic analysis on it and tags it for accessibility purposes using techniques broadly based around the PDF/UA standard and Section 508 compliance.

User accessibility standards like PDF/UA rely on Tagged PDF. This standard was initially put forward by the Association for Information and Image Management (AIIM) but was later adopted via the ISO in the form of ISO 14289-1

Document management applications --  
Electronic document file format enhancement  
for accessibility.

Strictly speaking Section 508 refers to the application rather than the document but of course this means that the producers of applications which consume documents are in a good position to mandate that documents must conform to certain standards so that the application can provide appropriate information. This is what people generally refer to when they talk about PDFs as being Section 508 compliant. Appropriate tagging achieves that aim.

Tagging allows people who have disabilities to have the content in a PDF presented to them via different mechanisms, For example an accessible PDF would provide information on page structure to allow a PDF reader to speak the content of the document. However there are a variety of assistive technologies available, ranging from readers to magnifiers to navigational aids.

Tagged PDFs are the same as normal PDFs but they have been annotated with metadata in the form of PDF tags. This metadata is required because PDF documents contain good layout information but little semantic structure. The tags that are required supply this semantic structure. The way they are inserted and operate is defined in the Adobe PDF Specification. The types of tags that are used and the way they are used are defined by accessibility standards such as PDF/UA.

---

Notes

The semantic analysis provided by ABCpdf is

based around reading order and results in the the logical structure of the PDF being determined. Content that is regarded as irrelevant is tagged as being an artifact in line with the PDF/UA standard.

Images present a particular challenge as automated processes do not find it easy to generate descriptions from bitmaps. However if you know what the different images represent you can tag each [PixMap](#) object dictionary with a "XXAlt" entry referring to a [StringAtom](#). When the MakeAccessible function is called these entries will be picked up and used and then deleted.

The MakeAccessible function will result in any existing tagged content being discarded. This is necessary because, while it is possible to determine if a document is already tagged, it is not possible to determine if it has been correctly tagged. Many PDF consumers do not understand tags and will not update the metadata appropriately if they operate on such documents.

To determine whether a document has been tagged you can find the "MarkInfo" entry in the [Doc.Catalog](#) as a [DictAtom](#) and then get the "Marked" entry of that as a [BoolAtom](#). The default is false.

You may wish to set the [SaveOptions.CompressObects](#) property to true, to reduce output size on save.

---

**Example**

See the [AccessibilityOperation Constructor](#).

# PageContents Property

---



Type	Default	Read Only	Description
[C#] PageContents	n/a	No	The pages to be operated upon.
[Visual Basic] PageContents			

This property specifies the pages to be operated upon.

Adding pages to a PageContents object can be a costly procedure taking a noticeable amount of time.

---

## Notes

So if you are performing a set of analysis operations on the same pages it can be more efficient to assign the PageContents from one to another rather than repeatedly re-populate from the original document.

None.

---

## Example

# FixFonts Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	n/a	No	Whether to attempt to fix font settings that may be required for accessibility.

Whether to attempt to fix font settings that may be required for accessibility.

Specifications like PDF/UA mandate certain requirements. One of the core requirements is that fonts are embedded rather than referenced.

---

## Notes

This property controls whether an attempt will be made to embed any fonts that are referenced rather than embedded in the document.

None.

---

## Example

# FixMetadata Property

---



Type	Default	Read Only	Description
[C#] bool [Visual Basic] Boolean	n/a	No	Whether to attempt to fix or add metadata that may be required for accessibility.

Whether to attempt to fix or add metadata that may be required for accessibility.

Specifications like PDF/UA mandate certain requirements. One of the core requirements is that XMP metadata is embedded in the document.

---

## Notes

This property controls whether an attempt will be made to add any such data. Information for the XMP section will be extracted from the info section of the PDF. A PDF/UA tag will be added as will other required features like a title.

None.

---

## Example





# EffectOperation Function

---

EffectOperation Constructor

**[C#]**

```
EffectOperation(string name)
```

---

Syntax

**[Visual Basic]**

```
New(name As String)
```

---

Params

Name	Description
name	The name of the effect that is required.

---

Notes

Create a EffectOperation.

The installed effects can be obtained using the [Names](#) static variable.

If the name is not one of the valid names then an exception will be thrown.

---

Example

None.





# Apply Function

---

Apply the effect to an image.

## [C#]

```
void Apply(PixMap pixMap)
```

## [Visual Basic]

```
Sub Apply(pixMap As PixMap)
```

- may throw `Exception()`

Syntax

Params

Name	Description
pixMap	The PixMap to which the image should be applied.

Apply the effect to an image specified as a [PixMap](#).

Effects normally involve wholesale changes and so it may well be necessary to change the color space and depth in order to apply the effect. If the [AutoRestore](#) property is set then the PixMap will be restored to a similar color space after processing and it will be recompressed at an appropriate quality. This default is generally

what is required.

---

**Notes**

If the `AutoRestore` property is not set then the final output may well have different characteristics such as color space and depth. In general the final image will be uncompressed eight bit RGB no matter what type of `PixelFormat` was supplied.

If you are relying on particular characteristics present in the original `PixelFormat` then you should set the `AutoRestore` property to false and then convert or recolor it after the effect has been applied. Since the final image will be left uncompressed you will likely want to recompress it using an appropriate schema.

---

**Example**

None.

# Names Property

---



Type	Default Value	Read Only	Description
[C#] static string[]	See below	Yes	Names of all the installed effects
[Visual Basic] Shared String()			

The names of all the installed effects.

The default effect names are:

- "None"
- "Median"
- "Sharpen"
- "Convolution"
- "Gaussian Blur"
- "Laplacian"
- "Unsharp Mask"
- "Despeckle"
- "Brightness"
- "Equalize"
- "Contrast"
- "Levels"
- "Auto Levels"
- "Histogram"
- "Twirl"

- "Pinch"
- "Ripple"
- "Wave"

---

**Example**

None.

# Name Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	The name of the Effect

The unique name by which the effect is known.

---

Notes

None.

---

Example

# Description Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	n/a	Yes	A description of what the effect does

A description of what the effect does.

---

## Notes

This human language description of the effect can be used to supplement the [Name](#).

---

## Example

None.

# Parameters Property

---



Type	Default Value	Read Only	Description
[C#] IDictionary<string, EffectParameter>	n/a	No	The parameters associated with the effect
[Visual Basic] IDictionary<string, EffectParameter>			

The parameters associated with the effect.

---

## Notes

Each item in the dictionary is identified by a name and takes a Parameter. So for a brightness effect there might be an intensity entry with a Parameter containing the value 25.

---

## Example

None.

# AutoQuality Property

---



Type	Default Value	Read Only	Description
[C#] <code>int</code> [Visual Basic] <code>Integer</code>	75	No	The quality of compression to use when automatically compressing after the effect is applied

---

## Notes

The quality of compression to use when automatically compressing after the effect is applied.

---

## Example

None.

# AutoRestore Property

---



Type	Default Value	Read Only	Description
[C#] bool [Visual Basic] Boolean	true	No	Whether to automatically restore the image color space and apply compression after the effect is applied

---

## Notes

Whether to automatically restore the image color space and apply compression after the effect is applied.

---

## Example

None.

# Name Property

---



Type	Default Value	Read Only	Description
[C#] string			
[Visual Basic] String	0?	No?	The name of the parameter

The name of the parameter.

---

Notes

None.

---

Example

# Description Property

---



Type	Default Value	Read Only	Description
[C#] string			A description of what the parameter does
[Visual Basic] String	0?	No?	

A description of what the parameter does.

---

Notes

None.

---

Example

# Maximum Property

---



Type	Default Value	Read Only	Description
[C#] double			The maximum recommended value (if applicable)
[Visual Basic] Double	n/a	Yes	

The maximum recommended value (if applicable).

---

Notes

None.

---

Example

# Minimum Property

---



Type	Default Value	Read Only	Description
[C#] double			The minimum recommended value (if applicable)
[Visual Basic] Double	n/a	Yes	

The minimum recommended value (if applicable).

---

Notes

None.

---

Example

# Value Property

---



Type	Default Value	Read Only	Description
[C#] double			The value of the parameter, or the first value if there are multiple
[Visual Basic] Double	n/a	No	

The value of the parameter, or the first value if there are multiple.

---

## Notes

None.

---

## Example

# Values Property

---



Type	Default Value	Read Only	Description
[C#] <code>double[]</code>			The values of the parameter
[Visual Basic] <code>Double()</code>	n/a	No	

---

## Notes

The values of the parameter. Most parameters have one value only.

---

## Example

None.

# Introduction to Effects

---



Effects are an operation which can be applied to images. They include common and useful functions like blur and sharpen filters.

Effects are applied using the [EffectOperation](#) class. To apply an effect you can use code as simple as this.

## [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img5);
        using (ImageLayer layer =
AddImagePage(doc, img5)) {
            using (EffectOperation effect = new
EffectOperation("Sharpen")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectSharpen1b.jpg");
    }
}
```

---

## Basics

## [Visual Basic]

```
Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img5)
        Using layer As ImageLayer =
AddImagePage(doc, img5)
            Using effect As New
EffectOperation("Sharpen")
                effect.Apply(layer.PixMap)
            End Using
        End Using
    End Using
End Sub
```

```
End Using
End Using
doc.Rendering.Save("EffectSharpen1b.jpg")
End Sub
```

The examples make use of a function named `AddImagePage`. This function simply creates an appropriate `Doc` object, adds the image and returns the `ImageLayer` which was added.

### [C#]

```
private ImageLayer AddImagePage(Doc doc,
string path) {
    using (XImage img =
XImage.FromFile(path, null)) {
        doc.MediaBox.SetSides(0, 0,
img.Width, img.Height);
        doc.Page = doc.AddPage();
        doc.Rect.String =
doc.MediaBox.String;
        int id = doc.AddImageObject(img);
        return
(ImageLayer)doc.ObjectSoup[id];
    }
}
```

### Examples

### [Visual Basic]

```
Private Function AddImagePage(doc As Doc,
path As String) As ImageLayer
    Using img As XImage =
XImage.FromFile(path, Nothing)
        doc.MediaBox.SetSides(0, 0,
```

```
img.Width, img.Height)
    doc.Page = doc.AddPage()
    doc.Rect.[String] = doc.MediaBox.
[String]
    Dim id As Integer =
doc.AddImageObject(img)
    Return DirectCast(doc.ObjectSoup(id),
ImageLayer)
    End Using
End Function
```

# AutoLevels Effect

---



AutoLevels automatically adjusts brightness and contrast to produce a balanced image with a good range of color intensities. In doing so it performs a function very much like the Levels effect, automatically deriving settings based on the image provided.

---

## Settings

Name	Default	Description
Clip	0.5 %	The percentage of extreme color values to ignore. This applies to the top and the bottom extremes.

Well defined images span an entire range of color intensities. However it is common to find images that do not. If a photo has been overexposed it will be too bright - there will be few colors at the low ends of intensity and many at the high end. Similarly if a photograph has been underexposed it will be very dark - all the colors will be at the low end of the range and virtually none at the high end.

---

## Workings

The AutoLevels effect detects and fixes this kind of imbalance. It scans through the levels of intensity within the image and chooses a level that should be regarded as black (low intensity) and another that should be regarded as white (high intensity). It then stretches the levels in the image so that all the intensities present lie between the black and the white points. This results in an image with a good span of color intensities.

To mitigate the effect of outliers - small numbers of pixels at extreme values of intensity - a clipping percentage is used. By default the value is 0.5% which means that the bottom and top 0.5% of pixels will be ignored when determining the black and white points.

The following example images show the effect of AutoLevels at different clipping levels.

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img1); // original image
        doc.Rendering.Save("EffectAutoLevels5.jpg")
        using (ImageLayer layer = AddImagePage(doc,
img1)) {
            using (EffectOperation effect = new
EffectOperation("Auto Levels")) {
                effect.Parameters["Clip"].Value = 0.5;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectAutoLevels50.jpg")
        using (ImageLayer layer = AddImagePage(doc,
img1)) {
            using (EffectOperation effect = new
EffectOperation("Auto Levels")) {
                effect.Parameters["Clip"].Value = 5.0;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectAutoLevels.jpg");
    }
}
```

## [Visual Basic]

```
Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img1)
        ' original image
        doc.Rendering.Save("EffectAutoLevels5.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img1)
            Using effect As New EffectOperation("Auto
Levels")
                effect.Parameters("Clip").Value = 0.5
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectAutoLevels50.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img1)
            Using effect As New EffectOperation("Auto
Levels")
                effect.Parameters("Clip").Value = 5.0
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectAutoLevels.jpg")
    End Using
End Sub
```

---

Example



Original Image before AutoLevels



After AutoLevels Clip 0.5%



After AutoLevels Clip 5.0%

# Brightness Effect

---



The Brightness effect can be used to lighten the image or to darken it.

---

## Settings

Name	Default	Description
Amount	0 %	Amount to lighten or darken the image.

---

## Workings

The value given is added to every pixel in the image. The value may be negative in which case the result is to darken rather than lighten the image.

The following show the effect of brightening or darkening an image.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img2); // original image
        doc.Rendering.Save("EffectBrightness.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img2)) {
            using (EffectOperation effect = new
EffectOperation("Brightness")) {
                effect.Parameters["Amount"].Value = 20;
                effect.Apply(layer.PixMap);
            }
        }
    }
}
```

```

    }
  }
  doc.Rendering.Save("EffectBrightness20.jpg"
  using (ImageLayer layer = AddImagePage(doc,
img2)) {
    using (EffectOperation effect = new
EffectOperation("Brightness")) {
      effect.Parameters["Amount"].Value = -20;
      effect.Apply(layer.PixMap);
    }
  }
  doc.Rendering.Save("EffectBrightness-
20.jpg");
  using (ImageLayer layer = AddImagePage(doc,
img2)) {
    using (EffectOperation effect = new
EffectOperation("Brightness")) {
      effect.Parameters["Amount"].Value = 60;
      effect.Apply(layer.PixMap);
    }
  }
  doc.Rendering.Save("EffectBrightness60.jpg"
  }
}

```

## [Visual Basic]

```

Sub ...
  Using doc As New Doc()
    AddImagePage(doc, img2)
    ' original image
    doc.Rendering.Save("EffectBrightness.jpg")
    Using layer As ImageLayer = AddImagePage(doc,
img2)
      Using effect As New
EffectOperation("Brightness")
        effect.Parameters("Amount").Value = 20

```

Example

```
        effect.Apply(layer.Pixmap)
    End Using
End Using
doc.Rendering.Save("EffectBrightness20.jpg"
Using layer As ImageLayer = AddImagePage(doc
img2)
    Using effect As New
EffectOperation("Brightness")
    effect.Parameters("Amount").Value = -20
    effect.Apply(layer.Pixmap)
    End Using
End Using
doc.Rendering.Save("EffectBrightness-20.jpg"
Using layer As ImageLayer = AddImagePage(doc
img2)
    Using effect As New
EffectOperation("Brightness")
    effect.Parameters("Amount").Value = 60
    effect.Apply(layer.Pixmap)
    End Using
End Using
doc.Rendering.Save("EffectBrightness60.jpg"
End Using
End Sub
```



Original Image

Brightness Amount 20



Brightness Amount -20 Brightness Amount 60

# Contrast Effect

---



The Contrast effect can be used to increase or decrease the contrast of an image.

---

## Settings

Name	Default	Description
Amount	0 %	Amount to increase or decrease contrast by.

---

## Workings

The value given is used to stretch the contrast within the image. Values greater than zero increase the amount of contrast while those less than zero decrease the contrast.

The following show the effect of increasing and decreasing contrast.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img2); // original image
        doc.Rendering.Save("EffectContrast.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img2)) {
            using (EffectOperation effect = new
EffectOperation("Contrast")) {
                effect.Parameters["Amount"].Value = 25;
```

```

        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectContrast25.jpg");
using (ImageLayer layer = AddImagePage(doc,
img2)) {
    using (EffectOperation effect = new
EffectOperation("Contrast")) {
        effect.Parameters["Amount"].Value = 50;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectContrast50.jpg");
using (ImageLayer layer = AddImagePage(doc,
img2)) {
    using (EffectOperation effect = new
EffectOperation("Contrast")) {
        effect.Parameters["Amount"].Value =
-50;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectContrast-
50.jpg");
}
}

```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img2)
        ' original image
        doc.Rendering.Save("EffectContrast.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img2)
            Using effect As New

```

---

Example

```
EffectOperation("Contrast")
    effect.Parameters("Amount").Value = 25
    effect.Apply(layer.PixMap)
End Using
End Using
doc.Rendering.Save("EffectContrast25.jpg")
Using layer As ImageLayer =
AddImagePage(doc, img2)
    Using effect As New
EffectOperation("Contrast")
        effect.Parameters("Amount").Value = 50
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectContrast50.jpg")
Using layer As ImageLayer =
AddImagePage(doc, img2)
    Using effect As New
EffectOperation("Contrast")
        effect.Parameters("Amount").Value = -50
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectContrast-50.jpg")
End Using
End Sub
```



Original Image



Contrast Amount 25



Contrast Amount 50



Contrast Amount -50

# Convolution Effect

---



The Convolution Effect allows you to produce a range of effects by specifying a set of convolution kernels. A simple explanation is given here but you may wish to refer to other sources for complete descriptions of convolution and how you can use it.

---

## Settings

Name	Default	Description
Width	3 pixels	The width of the filter to be applied.
Height	3 pixels	The height of the filter to be applied.
X	2 pixels	The horizontal center of the filter in pixels from left.
Y	2 pixels	The vertical center of the filter in pixels from top.
Number	2	The number of kernels to apply.
Values	See desc.	The values for the kernels in a comma delimited list. The items in the list may be integers or floating point numbers.  The default value is: "-1,0,1,-2,0,2,-1,0,1,1,2,1,0,0,0,-1,-2,-1"

Convolution is a general purpose filter effect for images. It works by determining the value of a central pixel by adding the weighted values of all its neighbors together. The weights applied to each pixel are determined by what is called a convolution kernel.

So if you want to take the average of all the immediate neighbors central pixel you would specify an equally weighted convolution kernel. Note that the total sum of all the weights is one so that the overall brightness of the image is not affected by the convolution. Note that the anchor point is highlighted to show which pixel should be regarded as central.

0.1111	0.1111	0.1111
0.1111	<b>0.1112</b>	0.1111
0.1111	0.1111	0.1111

If you wanted to take the average of the pixels immediately above, below, and to the sides of the central pixel, and you wanted to exclude the central pixel itself, you would specify the following kernel.

Workings

0.0000	0.2500	0.0000
0.2500	<b>0.0000</b>	0.2500
0.0000	0.2500	0.0000

Although a three square kernel with the anchor at the center is most common, you can use other shapes of kernel. For example the following convolution will shift the entire image left by one pixel.

<b>0.0000</b>	1.0000
---------------	--------

When you specify values you specify them from left to right and top to bottom. You can specify more than one filter at a time and the results will be added together.

The default kernel is a standard Sobel edge detector. This contains two filters - one vertical, one horizontal - to be applied and then added together. The two kernel values are:

-1	0	1	1	2	1
-2	<b>0</b>	2	0	<b>0</b>	0

-1 0 1

-1 -2 -1

The following examples show the effect of the default kernel - a Sobel Edge Detector.

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectConvolution.jpg");
        using (ImageLayer layer = AddImagePage(doc,
        {
            using (EffectOperation effect = new
EffectOperation("Convolution")) {
                // the default is a Sobel Edge Detector
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectConvolutionDefault.jpg");
        using (ImageLayer layer = AddImagePage(doc,
        {
            using (EffectOperation effect = new
EffectOperation("Convolution")) {
                // the following is a high intensity sharpness
filter
                double[] k = new double[] { 0, -1, 0, -1, 0, -1, 0, -1, 0 };
                effect.Parameters["Number"].Value = 1;
                effect.Parameters["Values"].Values = k;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectConvolutionSharpened.jpg");
    }
}
```

```
}
```

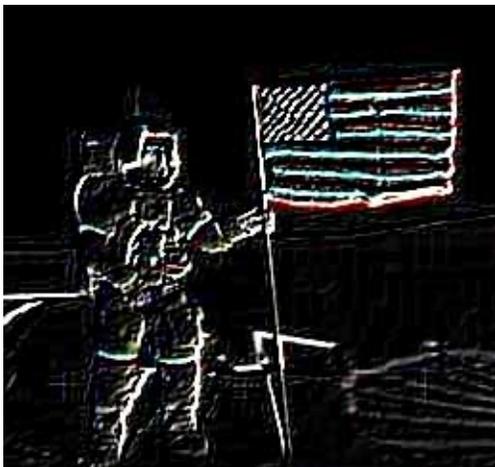
## [Visual Basic]

```
Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img3)
        ' original image
        doc.Rendering.Save("EffectConvolution.jpg")
        Using layer As ImageLayer = AddImagePage(doc)
            Using effect As New
                EffectOperation("Convolution")
                ' the default is a Sobel Edge Detector
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectConvolutionDefault.jpg")
        Using layer As ImageLayer = AddImagePage(doc)
            Using effect As New
                EffectOperation("Convolution")
                ' the following is a high intensity sharpness
                filter
                Dim k As Double() = New Double() {0, -1, 1,
                5, -1, 1, -5, 1, 0}
                effect.Parameters("Number").Value = 1
                effect.Parameters("Values").Values = k
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectConvolutionSharpened.jpg")
    End Using
End Sub
```

Example



Original Image



After Convolution Applied

# Despeckle Effect

---



The Despeckle filter removes noise from images without blurring edges. It attempts to detect complex areas and leave these intact while smoothing areas where noise will be noticeable.

---

## Settings

Name	Default	Description
Threshold	20	Threshold of complexity above which the image should not be smoothed.

The Despeckle filter smooths areas in which noise is noticeable while leaving complex areas untouched. The effect is that grain or other noise is reduced without severely affecting edges.

---

## Workings

The standard deviation of each pixel and its neighbors is calculated to determine if the area is one of high complexity or low complexity. If the complexity is lower than the threshold then the area is smoothed using a simple mean filter.

The following examples show the effect of a Despeckle filter applied to a portion of detail from a picture of the Hubble Space Telescope. The image is characterized by a relatively smooth background but a complex foreground. The image is very grainy

but the grain is obscured in areas of high complexity and is only really visible in the background.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img4); // original image
        doc.Rendering.Save("EffectDespeckle.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img4)) {
            using (EffectOperation effect = new
EffectOperation("Despeckle")) {
                effect.Parameters["Threshold"].Value =
20;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectDespeckle10.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img4)) {
            using (EffectOperation effect = new
EffectOperation("Despeckle")) {
                effect.Parameters["Threshold"].Value =
20;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectDespeckle20.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img4)) {
            using (EffectOperation effect = new
EffectOperation("Despeckle")) {
                effect.Parameters["Threshold"].Value =
30;
                effect.Apply(layer.PixMap);
            }
        }
    }
}
```

```
    doc.Rendering.Save("EffectDespeckle30.jpg")
  }
}
```

## [Visual Basic]

```
Sub ...
  Using doc As New Doc()
    AddImagePage(doc, img4)
    ' original image
    doc.Rendering.Save("EffectDespeckle.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img4)
      Using effect As New
EffectOperation("Despeckle")
        effect.Parameters("Threshold").Value =
10
          effect.Apply(layer.PixMap)
        End Using
      End Using
    doc.Rendering.Save("EffectDespeckle10.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img4)
      Using effect As New
EffectOperation("Despeckle")
        effect.Parameters("Threshold").Value =
20
          effect.Apply(layer.PixMap)
        End Using
      End Using
    doc.Rendering.Save("EffectDespeckle20.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img4)
      Using effect As New
EffectOperation("Despeckle")
        effect.Parameters("Threshold").Value =
30
```

---

Example

```
effect.Apply(layer.Pixmap)
End Using
End Using
doc.Rendering.Save("EffectDespeckle30.jpg")
End Using
End Sub
```



Original Image

The sea is very grainy while the the grain is obscured in areas of high complexity like the telescope.



Despeckle Radius 10

The sea has been appreciably degrained while retaining detail in areas of complexity.



Despeckle Radius 20

The sea has been further degrained.



Despeckle Radius 30

The telescope

Although the telescope remains unblurred  
remains unblurred, the but now the  
coastline is becoming a coastline has lost a  
little indistinct. lot of distinction.

# Equalize Effect

---



Equalize modifies an image to ensure that all levels of brightness are equally well represented. This function is very similar to the AutoLevels effect but is designed to modify brightness rather than color levels.

---

## Settings

Name	Default	Description
None		

---

## Workings

Most images span an entire range of brightness with all levels well represented. However sometimes images are too light or too dark and some levels are overpopulated leaving others underpopulated.

The Equalize effect modifies an image so that all levels of brightness are equally well represented within the image.

The following example images show the effect of Equalize.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img2); // original image
        doc.Rendering.Save("EffectEqualize1.jpg");
        using (ImageLayer layer = AddImagePage(doc,
```

```

img2)) {
    using (EffectOperation effect = new
EffectOperation("Equalize")) {
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectEqualizeAfter1.jpg");
AddImagePage(doc, img6); // original image
doc.Rendering.Save("EffectEqualize2.jpg");
using (ImageLayer layer = AddImagePage(doc,
img6)) {
    using (EffectOperation effect = new
EffectOperation("Equalize")) {
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectEqualizeAfter2.jpg");
}
}

```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img2)
        ' original image
        doc.Rendering.Save("EffectEqualize1.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img2)
            Using effect As New
EffectOperation("Equalize")
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectEqualizeAfter1.jpg");
        AddImagePage(doc, img6)
        ' original image
    End Using
End Sub

```

Example

```
doc.Rendering.Save("EffectEqualize2.jpg")
Using layer As ImageLayer = AddImagePage(doc,
img6)
    Using effect As New
EffectOperation("Equalize")
        effect.Apply(layer.Pixmap)
    End Using
End Using
doc.Rendering.Save("EffectEqualizeAfter2.jpg")
End Using
End Sub
```



Original Image



After Equalize



Original Image

After Equalize

# Gaussian Blur Effect

---



A Gaussian Blur is a general purpose blur filter. This removes fine image detail and noise leaving only larger scale changes. Gaussian Blurs produce a very pure smoothing effect without side effects.

---

## Settings

Name	Default	Description
Radius	1.5 pixels	<p>Determines the scale of fine detail that will be removed. Low values remove only very fine detail while high values remove larger levels of detail.</p> <p>This value represents the standard deviation of the Gaussian function.</p>

A Gaussian Blur is distinct from other blurs in that it has a well defined effect on different levels of detail within an image. As the level of detail becomes smaller the filter lets through less and less. With other types of blur (e.g. Mean Filter) the amount let through may vary considerably.

---

## Workings

As well as having this well defined and consistent frequency response, certain characteristics of the Gaussian function mean that large blurs can be applied much faster than other similar kinds of filters.

The following example images show the effect of Gaussian Blur.

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectGaussianBlur.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Gaussian Blur")) {
                effect.Parameters["Radius"].Value = 1.2;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectGaussianBlur12.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Gaussian Blur")) {
                effect.Parameters["Radius"].Value = 2.5;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectGaussianBlur25.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Gaussian Blur")) {
                effect.Parameters["Radius"].Value = 5.0;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectGaussianBlur50.jpg");
    }
}
```

## [Visual Basic]

```
Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img3)
        ' original image
        doc.Rendering.Save("EffectGaussianBlur.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img3)
            Using effect As New EffectOperation("Gaus
Blur")
                effect.Parameters("Radius").Value = 1.2
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectGaussianBlur12.jp
Using layer As ImageLayer = AddImagePage(doc,
img3)
            Using effect As New EffectOperation("Gaus
Blur")
                effect.Parameters("Radius").Value = 2.5
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectGaussianBlur25.jp
Using layer As ImageLayer = AddImagePage(doc,
img3)
            Using effect As New EffectOperation("Gaus
Blur")
                effect.Parameters("Radius").Value = 5.0
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectGaussianBlur50.jp
    End Using
End Sub
```

---

Example



Original Image before Gaussian Blur



After Gaussian Blur Radius 1.2 pixels



After Gaussian Blur Radius 2.5 pixels



After Gaussian Blur Radius 5.0 pixels

# Histogram Effect

---



Histogram is an image diagnostic rather than an image effect. When you apply the Histogram effect color levels are calculated and returned via the settings. The image is unaffected.

Name	Default	Description
Red	""	A histogram of red levels as a comma delimited string.
Green	""	A histogram of green levels as a comma delimited string.
Blue	""	A histogram of blue levels as a comma delimited string.
RGB	""	A histogram of RGB levels as a comma delimited string.

---

## Settings

Image histograms show how color levels are distributed within an image. Each color channel has a value between 0 and 255 and the histogram simply returns the number of pixels with each of these values. A light image will have many values at the higher end of the histogram while a dark one will have many values at the lower end.

---

## Workings

After the Histogram effect has been applied to an image the settings contain the histograms of each of the color channels and also of a combined RGB channel. These values are held as comma delimited strings.

The following example images show the effect of Histogram..

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img1); // original image
        doc.Rendering.Save("EffectHistogram.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img1)) {
            using (EffectOperation effect = new
EffectOperation("Histogram")) {
                effect.Apply(layer.PixMap);
                var pars = effect.Parameters;
                doc.Color.SetRgb(255, 0, 0);
                DrawGraph(doc, pars["Red"].Values);
                doc.Color.SetRgb(0, 255, 0);
                DrawGraph(doc, pars["Green"].Values);
                doc.Color.SetRgb(0, 0, 255);
                DrawGraph(doc, pars["Blue"].Values);
            }
        }
        doc.Rendering.Save("EffectHistogramGraph.jp
    }
}

private void DrawGraph(Doc doc, double[] values)
{
    double max = 0;
    for (int i = 0; i < values.Length; i++)
        max = Math.Max(max, values[i]);
}
```

```

double[] points = new double[values.Length *
for (int i = 0; i < values.Length; i++) {
    double x = i * (doc.Rect.Width /
values.Length);
    double y = values[i] * (doc.Rect.Height / n
points[(i * 2) + 0] = x;
points[(i * 2) + 1] = y;
}
doc.AddPoly(points, false);
}

```

### [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img1)
        ' original image
        doc.Rendering.Save("EffectHistogram.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img1)
            Using effect As New
EffectOperation("Histogram")
                effect.Apply(layer.PixMap)
                Dim pars = effect.Parameters
                doc.Color.SetRgb(255, 0, 0)
                DrawGraph(doc, pars("Red").Values)
                doc.Color.SetRgb(0, 255, 0)
                DrawGraph(doc, pars("Green").Values)
                doc.Color.SetRgb(0, 0, 255)
                DrawGraph(doc, pars("Blue").Values)
            End Using
        End Using
        doc.Rendering.Save("EffectHistogramGraph.jp
End Using
End Sub

```

```

Private Sub DrawGraph(doc As Doc, values As

```

```
Double()  
    Dim max As Double = 0  
    For i As Integer = 0 To values.Length - 1  
        max = Math.Max(max, values(i))  
    Next  
    Dim points As Double() = New Double(values.Length  
* 2 - 1) {}  
    For i As Integer = 0 To values.Length - 1  
        Dim x As Double = i * (doc.Rect.Width /  
values.Length)  
        Dim y As Double = values(i) * (doc.Rect.Hei  
/ max)  
        points((i * 2) + 0) = x  
        points((i * 2) + 1) = y  
    Next  
    doc.AddPoly(points, False)  
End Sub
```

Example



Original Image





# Laplacian Effect

---



The Laplacian filter is used for detection of edges in an image. It highlights areas in which intensity changes rapidly producing a picture of all the edges in an image.

---

## Settings

Name	Default	Description
Radius	0.8 pixels	This value determines the size of the edges that are detected. This value represents the standard deviation of the Laplacian of Gaussian function.

---

## Workings

The Laplacian filter is a standard Laplacian of Gaussian convolution. This is a second derivative function designed to measure changes in intensity without being overly sensitive to noise. The function produces a peak at the start of the change in intensity and then at the end of the change.

Because the Laplacian of Gaussian produces a fairly wide convolution for a small radius this filter can become quite computationally expensive as radius is increased.

The following examples show the effect of a Laplacian filter applied with a number of different settings.

## [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectLaplacian.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Laplacian")) {
                effect.Parameters["Radius"].Value = 0.8;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectLaplacian08.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Laplacian")) {
                effect.Parameters["Radius"].Value = 1.6;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectLaplacian16.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Laplacian")) {
                effect.Parameters["Radius"].Value = 3.2;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectLaplacian32.jpg");
    }
}
```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img3)
        ' original image
        doc.Rendering.Save("EffectLaplacian.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img3)
            Using effect As New
EffectOperation("Laplacian")
                effect.Parameters("Radius").Value = 0.8
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectLaplacian08.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img3)
            Using effect As New
EffectOperation("Laplacian")
                effect.Parameters("Radius").Value = 1.6
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectLaplacian16.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img3)
            Using effect As New
EffectOperation("Laplacian")
                effect.Parameters("Radius").Value = 3.2
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectLaplacian32.jpg")
    End Using
End Sub

```

---

Example



Original Image before Laplacian Filter



Laplacian Radius 0.8 Pixels



Laplacian Radius 1.6 Pixels



Laplacian Radius 3.2 Pixels

# Levels Effect

---



The Levels effect allows you fine control over brightness and contrast.

---

## Settings

Name	Default	Description
Black Input	0 %	The level in the current image that should be regarded as black.
White Input	100 %	The level in the current image that should be regarded as white.
Black Output	0 %	The level in the final image that should be regarded as black.
White Output	100 %	The level in the final image that should be regarded as white.

Well defined images span an entire range of color intensities. However it is common to find images that do not. If a photo has been overexposed it will be too bright - there will be few colors at the low ends of intensity and many at the high end. Similarly if a photograph has been underexposed it will be very dark - all the colors will be at the low end of the range and virtually none at the high end.

The Levels effect allows you fine control over brightness and contrast to let you correct this kind of problem. The basic method of adjustment is to set the black and white points on the input image. Normally the black point will be at 0 and the white point at 255. This simply means that black is represented by the value 0 and white is represented by the value 255.

---

### Workings

However if an image is too dark there may be no pixels at all with a value of 255. In this case what was white on the original image might be represented by a value of only 200. By setting the white input point to 200 and then applying the effect, the levels in between will be stretched to try and restore balance to the image. A similar operation setting the black input point would apply if an image was too light.

As well as being able to specify input points you can also specify output points. This lets you tell the effect what value should be regarded as white and black on the final output image.

The levels effect is often used in conjunction with an image histogram so that the exact representation of different color levels can be seen in the image.

The following example images show the effect of the Levels effect.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img6); // original image
        doc.Rendering.Save("EffectLevels.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img6)) {
```

```

        using (EffectOperation effect = new
EffectOperation("Levels")) {
            effect.Parameters["White Input"].Value
= 80;
            effect.Apply(layer.PixMap);
        }
    }
    doc.Rendering.Save("EffectLevelsWI80.jpg");
    using (ImageLayer layer = AddImagePage(doc,
img6)) {
        using (EffectOperation effect = new
EffectOperation("Levels")) {
            effect.Parameters["Black Input"].Value
= 20;
            effect.Apply(layer.PixMap);
        }
    }
    doc.Rendering.Save("EffectLevelsBI20.jpg");
    using (ImageLayer layer = AddImagePage(doc,
img6)) {
        using (EffectOperation effect = new
EffectOperation("Levels")) {
            effect.Parameters["White Output"].Value
= 80;
            effect.Apply(layer.PixMap);
        }
    }
    doc.Rendering.Save("EffectLevelswO80.jpg");
}
}

```

### [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img6)
        ' original image
    End Using
End Sub

```

---

Example

```
doc.Rendering.Save("EffectLevels.jpg")
Using layer As ImageLayer =
AddImagePage(doc, img6)
    Using effect As New
EffectOperation("Levels")
        effect.Parameters("White Input").Value
= 80
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectLevelsWI80.jpg")
Using layer As ImageLayer =
AddImagePage(doc, img6)
    Using effect As New
EffectOperation("Levels")
        effect.Parameters("Black Input").Value
= 20
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectLevelsBI20.jpg")
Using layer As ImageLayer =
AddImagePage(doc, img6)
    Using effect As New
EffectOperation("Levels")
        effect.Parameters("White Output").Value
= 80
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectLevelsW080.jpg")
End Using
End Sub
```



Original Image



White Input = 80



Black Input = 20



White Output = 80

# Median Effect

---



The Median filter smoothes images using a fast median algorithm. The median algorithm is particularly good at removing "salt and pepper" noise from images without removing too much fine detail.

---

## Settings

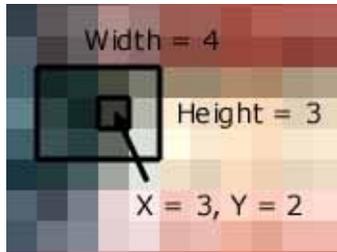
Name	Default	Description
Width	3 pixels	The width of the median filter to be applied.
Height	3 pixels	The height of the median filter to be applied.
X	2 pixels	The horizontal center of the filter in pixels from top left.
Y	2 pixels	The vertical center of the filter in pixels from top left.

The median filter replaces each pixel with the median of its neighbors. The Median has a number of advantages over the Mean. Firstly unrepresentative pixels do not unduly influence the outcome of the final pixel level - this is why the filter is good at removing "salt and pepper" noise. Secondly, because the final pixel must actually be the value of one of its neighbors, edges are preserved more faithfully.

---

## Workings

The image below show the neighboring pixels polled in a 4 by 3 Median Filter.



The following example images show the effect of a Median filter applied to a noisy picture at different width and height settings.

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img5); // original
        image
        doc.Rendering.Save("EffectMedian.jpg");
        using (ImageLayer layer =
        AddImagePage(doc, img5)) {
            using (EffectOperation effect = new
            EffectOperation("Median")) {
                effect.Parameters["Width"].Value =
                3;
                effect.Parameters["Height"].Value =
                3;
                effect.Parameters["X"].Value = 2;
                effect.Parameters["Y"].Value = 2;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectMedian3.jpg");
        using (ImageLayer layer =
        AddImagePage(doc, img5)) {
            using (EffectOperation effect = new
            EffectOperation("Median")) {
```

```

        effect.Parameters["Width"].Value =
5;
        effect.Parameters["Height"].Value =
5;
        effect.Parameters["X"].Value = 3;
        effect.Parameters["Y"].Value = 3;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectMedian5.jpg");
using (ImageLayer layer =
AddImagePage(doc, img5)) {
    using (EffectOperation effect = new
EffectOperation("Median")) {
        effect.Parameters["Width"].Value =
9;
        effect.Parameters["Height"].Value =
9;
        effect.Parameters["X"].Value = 4;
        effect.Parameters["Y"].Value = 4;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectMedian9.jpg");
}
}

```

### [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img5)
        ' original image
        doc.Rendering.Save("EffectMedian.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img5)
            Using effect As New

```

---

Example

```
EffectOperation("Median")
    effect.Parameters("Width").Value = 3
    effect.Parameters("Height").Value =
3
        effect.Parameters("X").Value = 2
        effect.Parameters("Y").Value = 2
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectMedian3.jpg")
Using layer As ImageLayer =
AddImagePage(doc, img5)
    Using effect As New
EffectOperation("Median")
        effect.Parameters("Width").Value = 5
        effect.Parameters("Height").Value =
5
            effect.Parameters("X").Value = 3
            effect.Parameters("Y").Value = 3
            effect.Apply(layer.PixMap)
        End Using
    End Using
    doc.Rendering.Save("EffectMedian5.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img5)
        Using effect As New
EffectOperation("Median")
            effect.Parameters("Width").Value = 9
            effect.Parameters("Height").Value =
9
                effect.Parameters("X").Value = 4
                effect.Parameters("Y").Value = 4
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectMedian9.jpg")
    End Using
```

End Sub



Original Image before Median Filter



Width = 3, Height = 3, X = 2, Y = 2



Width = 5, Height = 5, X = 3, Y = 3



Width = 9, Height = 9, X = 4, Y = 4

# Pinch Effect

---



The Pinch effect distorts the image as if it had been pinched.

---

## Settings

Name	Default	Description
Amount	50 %	The amount that the image should be pinched.
Extent	50 %	How far the effect should extend.
Speed	3	There is a general speed versus quality tradeoff. Higher values produce faster results at the expense of quality.

---

## Workings

The effect distorts the image as if it had been pinched.

The following example images show the effect of a Pinch filter applied to a picture with different settings.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectPinch.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
```

```

EffectOperation("Pinch")) {
    effect.Apply(layer.PixMap);
}
}
doc.Rendering.Save("EffectPinchDefault.jpg"
using (ImageLayer layer = AddImagePage(doc,
img3)) {
    using (EffectOperation effect = new
EffectOperation("Pinch")) {
        effect.Parameters["Amount"].Value = 75;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectPinchAmount75.jpg"
using (ImageLayer layer = AddImagePage(doc,
img3)) {
    using (EffectOperation effect = new
EffectOperation("Pinch")) {
        effect.Parameters["Extent"].Value = 100;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectPinchExtent100.jpg"
}
}
}

```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img3)
        ' original image
        doc.Rendering.Save("EffectPinch.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img3)
            Using effect As New EffectOperation("Pinch")
                effect.Apply(layer.PixMap)
            End Using
        End Using
    End Using
End Sub

```

```

    End Using
End Using
doc.Rendering.Save("EffectPinchDefault.jpg"
Using layer As ImageLayer = AddImagePage(doc
img3)
    Using effect As New EffectOperation("Pinch"
effect.Parameters("Amount").Value = 75
effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectPinchAmount75.jpg"
Using layer As ImageLayer = AddImagePage(doc
img3)
    Using effect As New EffectOperation("Pinch"
effect.Parameters("Extent").Value = 100
effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectPinchExtent100.jpg"
End Using
End Sub

```

Example



Original Image before Pinch



After Pinch with default settings



Amount = 75



Extent = 100



# Ripple Effect

---



The Pinch effect distorts the image as if it had been rippled.

---

## Settings

Name	Default	Description
Height	10 pixels	The amplitude of the wave determines how high the ripples are on the surface of the images.
Length	30 pixels	The length of the wave determines how far apart the ripples are placed.
Phase	0 degrees	The phase determines if the center of the ripple is a peak or a trough.
Speed	6	There is a general speed versus quality tradeoff. Higher values produce faster results at the expense of quality.

---

## Workings

The effect is very similar to a pond ripple. The image is distorted as if it was projected onto the surface of a pond and then a stone had been dropped into the middle.

The following example images show the effect of a Ripple filter a to a picture with different settings.

**[C#]**

---

```

void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectRipple.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Ripple")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectRippleDefault.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Ripple")) {
                effect.Parameters["Height"].Value = 15;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectRippleHeight15.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Ripple")) {
                effect.Parameters["Length"].Value = 10;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectRippleLength10.jpg");
    }
}

```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc()

```

```

AddImagePage(doc, img3)
' original image
doc.Rendering.Save("EffectRipple.jpg")
Using layer As ImageLayer = AddImagePage(doc,
img3)
    Using effect As New EffectOperation("Ripple")
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectRippleDefault.jpg")
Using layer As ImageLayer = AddImagePage(doc,
img3)
    Using effect As New EffectOperation("Ripple")
        effect.Parameters("Height").Value = 15
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectRippleHeight15.jpg")
Using layer As ImageLayer = AddImagePage(doc,
img3)
    Using effect As New EffectOperation("Ripple")
        effect.Parameters("Length").Value = 10
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectRippleLength10.jpg")
End Using
End Sub

```

---

**Example**



Original Image before Ripple



After Ripple with default settings



Height = 15



Length = 10

# Sharpen Effect

---



The Sharpen filter enhances edges using a simple algorithm. This is very fast to compute but can produce artificially over-sharp or over-noisy images if not used carefully.

---

## Settings

Name	Default	Description
None		

The Sharpen Filter uses a simple three square convolution to enhance edges.

The matrix for this convolution is:

---

## Workings

-0.125	-0.125	-0.125
-0.125	<b>+2.000</b>	-0.125
-0.125	-0.125	-0.125

The following examples show the effect of a Sharpen filter applied to a number of different images. Note that because JPEG compression has been used to compress these images some of the fine detail applied by the effect is not visible.

**[C#]**

```

void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img5); // original image
        doc.Rendering.Save("EffectSharpen1a.jpg");
        using (ImageLayer layer =
AddImagePage(doc, img5)) {
            using (EffectOperation effect = new
EffectOperation("Sharpen")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectSharpen1b.jpg");
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectSharpen2a.jpg");
        using (ImageLayer layer =
AddImagePage(doc, img3)) {
            using (EffectOperation effect = new
EffectOperation("Sharpen")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectSharpen2b.jpg");
        AddImagePage(doc, img6); // original image
        doc.Rendering.Save("EffectSharpen3a.jpg");
        using (ImageLayer layer =
AddImagePage(doc, img6)) {
            using (EffectOperation effect = new
EffectOperation("Sharpen")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectSharpen3b.jpg");
    }
}

```

**[Visual Basic]**

```
Sub ...
  Using doc As New Doc()
    AddImagePage(doc, img5)
    ' original image
    doc.Rendering.Save("EffectSharpen1a.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img5)
      Using effect As New
EffectOperation("Sharpen")
        effect.Apply(layer.Pixmap)
      End Using
    End Using
    doc.Rendering.Save("EffectSharpen1b.jpg")
    AddImagePage(doc, img3)
    ' original image
    doc.Rendering.Save("EffectSharpen2a.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img3)
      Using effect As New
EffectOperation("Sharpen")
        effect.Apply(layer.Pixmap)
      End Using
    End Using
    doc.Rendering.Save("EffectSharpen2b.jpg")
    AddImagePage(doc, img6)
    ' original image
    doc.Rendering.Save("EffectSharpen3a.jpg")
    Using layer As ImageLayer =
AddImagePage(doc, img6)
      Using effect As New
EffectOperation("Sharpen")
        effect.Apply(layer.Pixmap)
      End Using
    End Using
    doc.Rendering.Save("EffectSharpen3b.jpg")
  End Using
End Sub
```

---

Example



Original Image before Sharpen Filter



After Sharpen Filter Applied.



Original Image before Sharpen Filter



After Sharpen Filter Applied.

Note that as well as enhancing edges the effect has also enhanced 'mosquito noise' artifacts from the JPEG compression used on the original image. These are especially visible around the legs of the astronaut.



Original Image before Sharpen Filter



After Sharpen Filter Applied.

Note that the main effect here has been to enhance noise rather than improve quality.

# Twirl Effect

---



The Twirl effect distorts the image as if it had been twirled.

Name	Default	Description
Angle	120 degrees	The amount that the image should be twirled.
Extent	50 %	How far the effect should extend.
Speed	8	There is a general speed versus quality tradeoff. Higher values produce faster results at the expense of quality.

---

## Settings

The effect distorts the image as if it had been twisted.

---

## Workings

The following examples show the effect of a Twirl applied with a number of different settings.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectTwirl.jpg");
        using (ImageLayer layer = AddImagePage(doc,
            img3)) {
```

```

        using (EffectOperation effect = new
EffectOperation("Twirl")) {
            effect.Apply(layer.PixMap);
        }
    }
    doc.Rendering.Save("EffectTwirlDefault.jpg"
using (ImageLayer layer = AddImagePage(doc,
img3)) {
        using (EffectOperation effect = new
EffectOperation("Twirl")) {
            effect.Parameters["Angle"].Value = 360;
            effect.Apply(layer.PixMap);
        }
    }
    doc.Rendering.Save("EffectTwirlAngle360.jpg"
using (ImageLayer layer = AddImagePage(doc,
img3)) {
        using (EffectOperation effect = new
EffectOperation("Twirl")) {
            effect.Parameters["Extent"].Value = 100;
            effect.Apply(layer.PixMap);
        }
    }
    doc.Rendering.Save("EffectTwirlExtent100.jpg"
    }
}

```

## [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img3)
        ' original image
        doc.Rendering.Save("EffectTwirl.jpg")
        Using layer As ImageLayer = AddImagePage(doc,
img3)
            Using effect As New EffectOperation("Twirl")

```

```

        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectTwirlDefault.jpg"
Using layer As ImageLayer = AddImagePage(doc
img3)
    Using effect As New EffectOperation("Twirl"
        effect.Parameters("Angle").Value = 360
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectTwirlAngle360.jpg"
Using layer As ImageLayer = AddImagePage(doc
img3)
    Using effect As New EffectOperation("Twirl"
        effect.Parameters("Extent").Value = 100
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectTwirlExtent100.jpg"
End Using
End Sub

```

Example



Original Image before Twirl



After Twirl with default settings



Angle = 360



Extent = 100



# Unsharp Mask Effect

---



The Unsharp Mask filter is a simple method of sharpening a photo. Areas of complexity and fine detail within the image become crisper and better defined. An Unsharp Mask takes longer to perform than a Sharpen but gives more control and produces a more natural appearance.

---

## Settings

Name	Default	Description
Radius	1.5 pixels	Determines the scale of fine detail that will be enhanced. Low values enhance only very fine detail while high values enhance larger level detail.
Amount	60 %	The level of enhancement to be applied to the detail. Values greater than 100% will super-enhance any complex areas.
Threshold	0	If there is little fine detail then you can choose to enhance it by setting a threshold. Any detail less than the threshold will not be enhanced.

An Unsharp Mask is essentially a Blur in reverse. A Gaussian Blur is applied to a copy of the original image to produce an image with less detail. The blurred image is subtracted from the original to extract fine detail. This fine detail is then added to the original image to highlight complex areas.

---

## Workings

The radius parameter determines the radius of the Gaussian Blur and lets you choose the level of scale of detail that should be enhanced. The difference between color levels on the blurred and

original image is determined at each point on the image. If the difference is greater than the Threshold parameter then the Amount percentage of the difference is added back to the original image.

The following example shows the basic effect of Unsharp Mask on a photo and how the parameters can change the effect produced. Because JPEG compression has been used to compress these images, some of the fine detail applied by the effect is not visible.

**[C#]**

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img5); // original image
        doc.Rendering.Save("EffectUnsharpMask1a.jpg");
        using (ImageLayer layer = AddImagePage(doc, img5)) {
            using (EffectOperation effect = new
                EffectOperation("Unsharp Mask")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectUnsharpMask1b.jpg");
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectUnsharpMask2a.jpg");
        using (ImageLayer layer = AddImagePage(doc, img3)) {
            using (EffectOperation effect = new
                EffectOperation("Unsharp Mask")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectUnsharpMask2b.jpg");
        AddImagePage(doc, img6); // original image
        doc.Rendering.Save("EffectUnsharpMask3a.jpg");
        using (ImageLayer layer = AddImagePage(doc, img6)) {
            using (EffectOperation effect = new
```



```

Sub ...
  Using doc As New Doc()
    AddImagePage(doc, img5)
    ' original image
    doc.Rendering.Save("EffectUnsharpMask1a.jpg")
    Using layer As ImageLayer = AddImagePage(doc, img5)
      Using effect As New EffectOperation("UnsharpMask")
        effect.Apply(layer.PixMap)
      End Using
    End Using
    doc.Rendering.Save("EffectUnsharpMask1b.jpg")
    AddImagePage(doc, img3)
    ' original image
    doc.Rendering.Save("EffectUnsharpMask2a.jpg")
    Using layer As ImageLayer = AddImagePage(doc, img3)
      Using effect As New EffectOperation("UnsharpMask")
        effect.Apply(layer.PixMap)
      End Using
    End Using
    doc.Rendering.Save("EffectUnsharpMask2b.jpg")
    AddImagePage(doc, img6)
    ' original image
    doc.Rendering.Save("EffectUnsharpMask3a.jpg")
    Using layer As ImageLayer = AddImagePage(doc, img6)
      Using effect As New EffectOperation("UnsharpMask")
        effect.Apply(layer.PixMap)
      End Using
    End Using
    doc.Rendering.Save("EffectSharpen3b.jpg")
    Using layer As ImageLayer = AddImagePage(doc, img6)
      Using effect As New EffectOperation("UnsharpMask")
        effect.Parameters("Amount").Value = 140
        effect.Apply(layer.PixMap)
      End Using
    End Using
  End Using
End Sub

```

```
    End Using
End Using
doc.Rendering.Save("EffectUnsharpMaskSetting")
Using layer As ImageLayer = AddImagePage(doc, "EffectUnsharpMask")
    Using effect As New EffectOperation("UnsharpMask")
        effect.Parameters("Amount").Value = 140
        effect.Parameters("Radius").Value = 6
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectUnsharpMaskSetting")
Using layer As ImageLayer = AddImagePage(doc, "EffectUnsharpMask")
    Using effect As New EffectOperation("UnsharpMask")
        effect.Parameters("Amount").Value = 140
        effect.Parameters("Radius").Value = 6
        effect.Parameters("Threshold").Value = 100
        effect.Apply(layer.PixMap)
    End Using
End Using
doc.Rendering.Save("EffectUnsharpMaskSetting")
End Using
End Sub
```



Original Image

---

Example



Unsharp Mask Default Settings



Original Image



Unsharp Mask Default Settings



Original Image



Unsharp Mask Default Settings

The following examples show how the parameters can change the effect produced.



Unsharp Mask Default Settings



Unsharp Mask Amount =140



Unsharp Mask Amount =140 Radius = 6



Unsharp Mask Amount =140 Radius = 6 Threshold = 40

# Wave Effect

---



The Wave effect distorts the image as if it had been disturbed by a number of random waves.

---

## Settings

Name	Default	Description
Number	3	The number of random waves that should be generated.
LengthMin	40 pixels	The minimum wavelength for a random wave.
LengthMax	60 pixels	The maximum wavelength for a random wave.
HeightMin	5 pixels	The minimum amplitude for a random wave.
HeightMax	15 pixels	The maximum amplitude for a random wave.
Speed	6	There is a general speed versus quality tradeoff. Higher values produce faster results at the expense of quality.

---

## Workings

The effect distorts the image as if it had been disturbed by a number of waves. By choosing appropriate values you can make images look like they are underwater or are being seen through a heat haze.

The following examples show the effect of a Wave applied with a number of different settings.

### [C#]

```
void function() {
    using (Doc doc = new Doc()) {
        AddImagePage(doc, img3); // original image
        doc.Rendering.Save("EffectWave.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Wave")) {
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectWaveDefault.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Wave")) {
                effect.Parameters["LengthMin"].Value =
15;
                effect.Parameters["LengthMax"].Value =
30;
                effect.Apply(layer.PixMap);
            }
        }
        doc.Rendering.Save("EffectWaveLength.jpg");
        using (ImageLayer layer = AddImagePage(doc,
img3)) {
            using (EffectOperation effect = new
EffectOperation("Wave")) {
                effect.Parameters["HeightMin"].Value =
0;
```

```

        effect.Parameters["HeightMax"].Value =
5;
        effect.Apply(layer.PixMap);
    }
}
doc.Rendering.Save("EffectWaveHeight.jpg");
}
}

```

### [Visual Basic]

```

Sub ...
    Using doc As New Doc()
        AddImagePage(doc, img3)
        ' original image
        doc.Rendering.Save("EffectWave.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img3)
            Using effect As New
EffectOperation("Wave")
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectWaveDefault.jpg")
        Using layer As ImageLayer =
AddImagePage(doc, img3)
            Using effect As New
EffectOperation("Wave")
                effect.Parameters("LengthMin").Value =
15
                effect.Parameters("LengthMax").Value =
30
                effect.Apply(layer.PixMap)
            End Using
        End Using
        doc.Rendering.Save("EffectWaveLength.jpg")
        Using layer As ImageLayer =

```

---

Example

```
AddImagePage(doc, img3)
    Using effect As New
EffectOperation("Wave")
    effect.Parameters("HeightMin").Value =
    effect.Parameters("HeightMax").Value =
    effect.Apply(layer.PixMap)
    End Using
    End Using
    doc.Rendering.Save("EffectWaveHeight.jpg")
    End Using
End Sub
```



Original Image before Wave



After Wave with default settings



LengthMin = 15, LengthMax = 30



HeightMin = 0, HeightMax = 5

# EmbeddedFile Class

---

An EmbeddedFile represents a file stream embedded inside a PDF document.

It includes a small amount of metadata such as, optionally, the file creation and modification date. However it is primarily related to file data..



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
  WebSupergoo.ABCpdf10.Objects.StreamObject
  WebSupergoo.ABCpdf10.Objects.EmbeddedFile
```

Method	Description
EmbeddedFile	EmbeddedFile Constructor.
• UpdateMetadata	Update metadata for the embedded file.
	inherited methods...

Property	Description
Checksum	The 16 byte checksum for the embedded file.
CreationDate	The creation date of the embedded file.

MacCreator	The Macintosh file creator - a four char code represented as an integer.
MacResFork	The Macintosh resource forks stream for this file.
MacType	The Macintosh file type - a four char code represented as an integer.
ModificationDate	The modification date of the embedded file.
Size	The size of the embedded file.
Subtype	The subtype of the embedded file.
	inherited properties...

# FileSpecification Class

---

A file specification represents the name and location of a file and optionally may also include references to embedded data.

The crucial properties of this class are Uri and EmbeddedFile. All methods and properties are organized around backwards compatibility with obsolete entries so you are unlikely to need them when creating new documents.



If you have an old PDF which contains obsolete entries, the Rationalize method will attempt to make the entries compliant. So again you only need the Rationalize method and the Uri and EmbeddedFile properties.

```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
    WebSupergoo.ABCpdf10.Objects.FileSpecific
```

Method	Description
<a href="#">FileSpecification</a>	FileSpecification Constructor.
<a href="#">GetPath</a>	Get the path to the file for the specified platform.
<a href="#">Rationalize</a>	Removes any obsolescent and redundant entries.
<a href="#">SetPath</a>	Set the path to the file for the specified platform.

---

<b>Property</b>	<b>Description</b>
Description	A description of the file for use in a user interface.
EmbeddedFile	The embedded file if one has been embedded in the PDF.
EmbeddedFiles	The set of embedded files if a set has been embedded in the PDF.
Platform	The default platform being used for the file specification.
• Uri	The URI to the file.
Volatile	Whether the file changes frequently.

# XColor Class

---

This object represents a color. Colors may be expressed in RGB, CMYK, Grayscale or generic unspecified color components.

When first created, the color defaults to RGB black - "0 0 0".



System.Object

**WebSupergoo.ABCpdf10.XColor**

Implements: IDisposable,  
IEquatable<XColor>,  
IComparable<XColor>

Method	Description
<b>S»</b> <a href="#">FromGray</a>	Create an XColor from a grayscale value.
<b>S»</b> <a href="#">FromRgb</a>	Create an XColor from a set of RGB component values.
<b>S»</b> <a href="#">FromCmyk</a>	Create an XColor given a set of CMYK component values.
	Create an XColor

<b>S»</b> FromComponents	from a set of PDF components in the generic ColorSpace color space.
<b>S»</b> • FromArrayAtom	Create an XColor from an ArrayAtom of NumAtoms containing PDF color values.
<b>S»</b> • FromOperator	Create an XColor given a PDF color operator and a set of Atoms containing the arguments for that operator.
SetColor	Sets the color.
SetGray	Set the color to a grayscale value.
SetRgb	Set the color to an RGB value.
Equals	Test whether the two colors are effectively the same.
GetHashCode	A hash code for the XColor.

SetCmyk	Set the color to an CMYK value.
SetComponents	Set the color to a set of ColorSpace PDF components.
SetRandom	Set the color to a random opaque value in the current color space.
ToArrayAtom	An ArrayAtom representation of the components of the color.
ToString	Returns a string representation of the object.

Property	Description
Alpha	The alpha opacity.
Black	The black component.
Blue	The blue component.
Color	The System.Drawing.Color.

ColorSpace	The native color space for the color.
Components	The components of the color in native PDF format.
Cyan	The cyan component.
Gray	The gray component.
Green	The green component.
Magenta	The magenta component.
Name	Any name that may be associated with this color.
Red	The red component.
String	The color as a string.
Yellow	The yellow component.

# OpAtom Class

---

An atom which represents a content stream operator.



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.OpAtom
```

Method	Description
<a href="#">OpAtom</a>	Create an operator atom for a string.
<a href="#">Equals</a>	Test whether the two OpAtoms are the same.
<a href="#">Find</a>	Finds specified types of OpAtom entries in an array.
● <a href="#">GetParameter</a>	Gets the parameter associated with the OpAtom at the specified index and validates that the atom is of the correct type.
	Gets the parameters associated with the

GetParameters	OpAtom at the specified index and validates that the atoms are of the correct type.
	inherited methods...

Property	Description
Text	The text of the operator.
	inherited properties...

# FormXObject Class

---

A Form XObject is a self contained stream of graphics operations which can be referenced as a group.



System.Object

WebSupergoo.ABCpdf10.Objects.IndirectObject

WebSupergoo.ABCpdf10.Objects.StreamObject

**WebSupergoo.ABCpdf10.Objects.FormXobje**

Method	Description
AddResource	Add a particular type of resource.
GetResourceMap	Get a dictionary mapping the names of a particular type of resource to Atoms.
	inherited methods...

Property	Description
BBox	The rect defining the bounding box of the graphic.
Matrix	The transformation matrix used for mapping the Form XObject space into user space.

inherited properties...

# Pixmap Class

---

A bitmap based image.



System.Object

WebSupergoo.ABCpdf10.Objects.IndirectObject

WebSupergoo.ABCpdf10.Objects.StreamObject

**WebSupergoo.ABCpdf10.Objects.Pixmap**

Method	Description
<b>S»</b> FromXImage	Pixmap static constructor.
• CompressCcitt	Compresses the image using CCITT compression.
• CompressJbig2	Compresses the image using JBIG2 compression.
• CompressJpeg	Compresses the image using JPEG compression.
• CompressJpx	Compresses the image using JPEG 2000 compression.
Decompress	Decompress the data in the stream using on-the-

	fly resizing.
Flip	Flip the image horizontally or vertically.
● GetBitmap	Get the QPixmap image as a System.Drawing.Bitmap.
● Realize	Converts the image to component color.
● Recolor	Converts the image from one color space to another.
● Resample	Changes the number of bits per color component.
● Resize	Resizes the image.
● Rotate	Rotate the image clockwise.
Save	Saves the QPixmap to stream attempting to preserve resolution, color space and depth as far as the output format allows.
SetAlpha	Sets a constant alpha value (0-255) for this image.
	Set the content of the

SetBitmap	object as a Bitmap.
SetChromaKey	Sets a chromakey transparent color for this image.
ToGrayscale	Converts the image to grayscale.

Property	Description
AutoFix	Whether to automatically fix corrupt images.
BitsPerComponent	The number of bits per color component.
ColorSpace	The ColorSpace for this image.
ColorSpaceType	The ColorSpace for this image.
Components	The number of color components for each pixel.
Height	The height of the image in pixels.
ImageMask	Whether this image is a one bit image mask.

Mask	Any one bit image mask associated with this image.
Matte	Any matte associated with this soft mask.
SMask	Any soft image mask associated with this image.
Width	The width of the image in pixels.

# XPoint Class

---

Represents a point in two-dimensional space. When first created the object defaults to the origin "0 0" which is at the bottom left of the space.

ABCpdf uses the standard Adobe PDF coordinate space. The origin of this space is at the bottom left of the document. Distances are measured up and right in points. Points are a traditional measure for print work and there are 72 points in an inch. For further details see the [Coordinates](#) section of the documentation.



System.Object

**WebSupergoo.ABCpdf10.XPoint**

Implements: IDisposable,  
IEquatable<XPoint>,  
IComparable<XPoint>

Method	Description
<a href="#">XPoint</a>	XPoint Constructor.
<a href="#">Copy</a>	Copies a series of X and Y coordinates between arrays of doubles and arrays of XPoints.
<a href="#">Equals</a>	Test whether the two points are effectively

	the same.
GetHashCode	A hash code for the XPoint.
SetPoint	Sets the point.
ToString	Returns a string representation of the object.

<b>Property</b>	<b>Description</b>
Point	The System.Drawing.Point.
String	The point as a string.
X	The horizontal coordinate.
Y	The vertical coordinate.

# XRect Class

---

Represents a rectangular area in two-dimensional space. When first created the object defaults to an empty rectangle "0 0 0 0".

ABCpdf uses the standard Adobe PDF coordinate space. The origin of this space is at the bottom left of the document. Distances are measured up and right in points. Points are a traditional measure for print work and there are 72 points in an inch. For further details see the [Coordinates](#) section of the documentation.



You can change the coordinate system used by the [Doc.Rect](#) and the [Doc.MediaBox](#) using the [Doc.Units](#) and [Doc.TopDown](#) properties.

```
System.Object  
    WebSupergoo.ABCpdf10.XRect
```

```
Implements: IDisposable,  
IEquatable<XRect>, IComparable<XRect>
```

Method	Description
<b>S»</b> <a href="#">FromLbwh</a>	Creates an XRect from a bottom left corner, a width and a height.
<b>S»</b> <a href="#">FromLtw</a>	Creates an XRect from a top left corner, a width and a height.

<b>S»</b> FromSides	Creates an XRect from the coordinates of two diagonally opposite corners.
<b>S»</b> FromPoints	Create the smallest XRect that encloses all the points supplied.
XRect	XRect Constructor.
Contains	Determine if this rectangle contains a specified point or rectangle.
FitIn	Fits the rectangle as content inside another rectangle.
GetCorners	Get the four corners of the rectangle.
Inset	Insets the edges of the rectangle.
Intersect	Intersects this rectangle with another rectangle.
IntersectsWith	Determine if this rectangle intersects with another.
Magnify	Magnifies the rectangle.

Move	Translate the rectangle.
Position	Position the bottom left of the rectangle.
Resize	Resizes the rectangle.
Union	Union this rectangle with another rectangle.
SetRect	Sets the location and size of the rectangle.
SetSides	Sets the sides of the rectangle.
ToString	Returns a string representation of the object.
Equals	Test whether the two rectangles are effectively the same.
GetHashCode	A hash code for the XRect.

Property	Description
Bottom	The bottom coordinate.
Height	The height of the rectangle.
Left	The left coordinate.

Pin	The corner of the rectangle to pin.
Rectangle	The System.Drawing.Rectangle.
Right	The right coordinate.
String	The rect as a string.
Top	The top coordinate.
Width	The width of the rectangle.
HasArea	Whether the rectangle has area.

# Doc Class

---

This is the top level object that represents a PDF document.



System.Object  
**WebSupergoo.ABCpdf10.Doc**

Implements: IDisposable

Method	Description
AddArc	Adds an arc to the current page.
AddBookmark	Adds a bookmark pointing to a page.
• AddColorSpaceFile	Adds an ICC color space to the document.
AddColorSpaceSpot	Adds a spot color space to the document.
AddFont	Adds a font to the document.
AddGrid	Adds a grid to the current page.
AddHtml	Adds a block of styled text to the current page.

	current pa
● AddImage	Adds an in current pa
● AddImageBitmap	Adds a System.D to the curr
● AddImageCopy	Adds a co existing in Doc, to the page.
● AddImageData	Extract an data and a current pa
AddImageDoc	Draw a pa PDF docu current pa document
AddImageFile	Extract an file and ac current pa
● AddImageHtml	Renders a specified a
● AddImageObject	Adds an X image to t page.
● AddImageToChain	Adds a ne a paged F

	PostScript
• AddImageUrl	Renders a specified image
AddLine	Adds a line to the current page
AddObject	Adds a named object to the document
AddOval	Adds an oval to the current page
AddPage	Adds a page to the current document
AddPie	Adds a pie slice to the current page
AddPoly	Adds a polygon to the current page
AddRect	Add a rectangle to the current page
AddText	Adds a block of text to the current page
AddXObject	Add a Font or Image XObject to the page.
Append	Appends content to the end of the page
Chainable	Determine if a method is chainable

Clear	Clears the
ClearCachedDecompressedStreams	Clears the decompre stream ob
Delete	Deletes an previously document
EmbedFont	Embeds a document
FillRect	Adds a pa rectangle page.
FitHtml	Fit a block styled text current re
FitText	Fit a block the current the current
Flatten	Flattens a compresso page.
FrameRect	Adds a reo frame to th page.
● GetData	Saves a d memory.
	Gets string

GetInfo	about an c
GetInfoDate	Gets date about an c
GetInfoDouble	Gets num information object.
GetInfoInt	Gets num information object.
GetInfoInt64	Gets num information object.
● GetStream	Gets a do raw data s
GetText	Extract co current pa specified f
MeasureText	Measure t block of te adding it t
● Read	Reads an document
RemapPages	Remaps p reordering deletion.
Save	Saves the

	PDF.
SetInfo	Sets information for an object.
ToString	A string representation of the graph of the document.

Property	Description
Bookmark	The top level bookmark for the document.
Color	The current drawing and filling color.
ColorSpace	The current ColorSpace ID.
CropBox	The current document visible page crop box.
Encryption	The current encryption settings.
Font	The current Font ID.
FontSize	The current text size.
Form	The document form and fields.
HtmlOptions	The HTML and URL rendering options.
Layer	The insertion layer for new content.
LayerCount	The number of layers on the current page.
MediaBox	The current document page size.
ObjectSoup	The collection of objects that make up the PDF.
Options	The state options for low level drawing control.

Page	The current Page ID.
PageCount	The number of pages in the document.
PageNumber	The page number of the current page.
Pos	The current drawing position.
Rect	The current rectangle used for drawing operations.
Rendering	The rendering options and control.
Root	The root catalog object.
SaveOptions	The save options and control.
String	A string representation of the graph of the document.
TextStyle	The current style for drawing text.
TopDown	The current position of the origin.
Transform	The current transformation for drawing.
Units	The current measurement units.
Width	The current line width.

# XImage Class

---

Allows access to images stored in files or in data. XImages can be added to a document using the [Doc.AddImageObject](#) method.



System.Object  
**WebSupergoo.ABCpdf10.XImage**

Implements: IDisposable

Method	Description
<b>S»</b> ● <a href="#">FromFile</a>	Creates an XImage from a file path.
<b>S»</b> ● <a href="#">FromData</a>	Creates an XImage from an array of bytes.
<b>S»</b> ● <a href="#">FromStream</a>	Creates an XImage from a Stream.
<a href="#">Dispose</a>	Dispose of the object.
<a href="#">Clear</a>	Clears the image.
● <a href="#">SetData</a>	Load an image from data.
● <a href="#">SetFile</a>	Load an image from a file.
● <a href="#">SetMask</a>	Assign a soft mask to the image.

- **SetStream** Load an image from stream.

<b>Property</b>	<b>Description</b>
<b>BoundingBox</b>	The physical bounds of the image in points.
<b>Frame</b>	The currently selected frame.
<b>FrameCount</b>	The number of frames in the image.
<b>FrameRate</b>	The default frame rate for a moving image.
<b>HasRealRes</b>	Whether the image specifies the resolution.
<b>Height</b>	The height of the current frame (pixels).
<b>HRes</b>	The horizontal resolution of the current frame (DPI).
<b>Indirect</b>	Whether the image will be added using indirect mode.
<b>NeedsFile</b>	Whether the file needs to exist.
	Whether the stream

NeedsStream	needs be kept open.
Selection	The current selection rectangle.
Type	The type of image.
VRes	The vertical resolution of the current frame (DPI).
Width	The width of the current frame (pixels).

# XTransform Class

---

Represents a world space transform. When first created the object defaults to the identity - no transformation.

A world space transform is not same an object transform. You are changing the coordinate system - not the objects you're inserting.



System.Object

**WebSupergoo.ABCpdf10.XTransform**

Implements: IDisposable,  
IEquatable<XTransform>,  
IComparable<XTransform>

Method	Description
XTransform	XTransform Constructor.
Invert	Invert the transform.
Equals	Determines if two transforms are effectively the same.
Magnify	Scale about a locked anchor point.
PreMultiply	Pre-multiplies this transformation matrix

	by the supplied transform.
PostMultiply	Post-multiplies this transformation matrix by the supplied transform.
Reset	Reset to the identity.
Rotate	Rotate about a locked anchor point (angle in degrees).
Skew	Skew horizontally and vertically about a locked anchor point.
SetTransform	Set the transform.
ToString	Returns a string representation of the object.
TransformPoint	Applies this transform to a specified point.
TransformPoints	Applies this transform to a specified array of points.
Translate	Translate horizontally and vertically.

GetHashCode	A hash code for the XTransform.
-------------	---------------------------------

Property	Description
String	The transform as a string.
AngleUnit	The angle unit, degrees or radians.
Elements	The transform as an array of floating-point values.
Matrix	The transform as a System.Drawing.Drawing2D Matrix.
MediaMatrix	The transform as a System.Windows.Media Matrix.
OffsetX	The x translation.
OffsetY	The y translation.

# XReadOptions Object

---

Represents settings for read operations.

By creating a XReadOptions object and passing it in during a read operation you can take complete control of the way in which your content is imported. There can be different read modules which can import content in different ways. For example a Microsoft Word Document can be converted to PDF either using OpenOffice.org or by using Microsoft Word. A read options object allows you to precisely target one or more modules using particular settings.



System.Object

**WebSupergoo.ABCpdf10.XReadOptions**

Implements: IDisposable,  
IEquatable<XReadOptions>

Method	Description
<a href="#">XReadOptions</a>	XReadOptions Constructor.
<a href="#">ClearCache</a>	Clear cached data and terminate worker threads and worker processes for a read module.
<a href="#">Dispose</a>	Dispose of the object.

Equals	Test whether the two objects are effectively the same.
GetHashCode	A hash code for the XReadOptions.

Property	Description
ReadModule	Gets or sets the module to use.
AddForms	Whether form fields should be live.
Bookmarks	Bookmark creation strategy.
ContentItem	Gets or sets the content items to process.
DefaultFont	The font to use for text that does not specify a font.
	The default

DefaultRect	document size for documents that do not specify a size.
DotsPerInch	The default resolution to use for the import.
EnableMSOfficeMacros	Whether to enable macros when opening MS Office documents.
ErrorHandling	The error handling behavior.
ExtraChecks	Whether to apply extra processing to enable certain types of corrupt document to be read.
Frame	The frame to be read from a multiple frame

	image such as a movie.
FileExtension	Gets or sets the file extension for data sources that do not have file names.
Log	The log for the read operation.
MakeFieldNamesUnique	Whether field names should be changed to make them unique.
OpenOfficeParameters	OpenOffice.org PDF conversion control parameters.
Operation	Gets or sets the Operation to use.
Password	Gets or sets the password needed to read

	the source.
PreserveTransparency	Gets or sets a value that indicates whether the transparency of raster images should be preserved.
Timeout	Gets or sets the timeout in milliseconds for external components.
SkipRevisions	Skip back a number of revisions when reading an incrementally saved PDF document.
OpenPortfolios	Whether to automatically open the default document inside a PDF

portfolio rather  
than the  
portfolio itself.

# SwfImportOperation Class

---

An operation used to import Flash movies (SWF files).



```
System.Object  
  WebSupergoo.ABCpdf10.Operations.Operation  
    WebSupergoo.ABCpdf10.Operations.SwfImport
```

Method	Description
SwfImportOperation	SwfImportOperation Const
• Import	Imports selected frames of movie.

Property	Description
Doc	The target PDF document.
BackgroundRegion	Gets or sets the background
ClipRegion	Gets or sets the clip rectang
ResetRegions	Gets or sets a value that inc whether BackgroundRegion ClipRegion are reset when t Operation.ProcessingObjec ProcessingSourceType.Mul or of ProcessingSourceType.Ima

	generated.
Timeout	Gets or sets the time-out for execution.
ContentAlign	Gets or sets the content align
ContentScaleMode	Gets or sets the content scale
Parameters	The parameters for initializing the machine.

# GraphicLayer Class

---

A generic graphic layer appearing on a page of the document.



System.Object

WebSupergoo.ABCpdf10.Objects.IndirectObject

WebSupergoo.ABCpdf10.Objects.StreamObject

WebSupergoo.ABCpdf10.Objects.Layer

**WebSupergoo.ABCpdf10.Objects.Graphi**

Method	Description
--------	-------------

	inherited methods...
--	----------------------

Property	Description
----------	-------------

	inherited properties...
--	-------------------------

# XHtmlOptions Object

---

Represents the current HTML rendering settings.

The properties of this object may be used to control the way that the [Doc.AddImageUrl](#) and [Doc.AddImageHtml](#) methods work.

The methods of this object operate on objects added using the the [Doc.AddImageUrl](#) and [Doc.AddImageHtml](#) methods. Some operations change the document. Others provide information about the content which has been added.



HTML documents can be imported using different HTML engines. The current supported ones are the MSHTML engine (used in Microsoft Internet Explorer) and a modified version of Mozilla Firefox's Gecko engine tailored to work with ABCpdf. Different engines interpret HTML documents in different ways and support a different set of HTML options, please see the [Engine](#) property for details.

System.Object

**WebSupergoo.ABCpdf10.XHtmlOptions**

Method	Description
<a href="#">EndTasks</a>	Ends any HTML Engine worker threads or processes.

GetHttpStatusCode	Retrieves the HTTP status code.
GetScriptReturn	Retrieves the client side onload script return value.
GetTagIDs	Gets an array of the HTML IDs of tagged visible items.
GetTagRects	Gets an array of the locations of tagged visible items.
GetTagUntransformedRects	Gets an array of the locations of tagged visible items before Doc.Transform is applied.
LinkDestinations	Convert a restricted selection of external links to internal

	links.
LinkPages	Convert external links to internal links wherever possible.
PageCacheClear	Clears the HTML page cache.
PageCachePurge	Purges the HTML page cache.
SetTheme	Specify whether to use Windows themes or not.

Property	Description
Engine	The engine to use for HTML import operations.
	An object that provides access to only

ForMSHtml	the HTML options supported by the MSHTML engine.
ForGecko	An object that provides access to only the HTML options supported by the Gecko HTML engine.
AddForms	Whether form fields should be live.
AddLinks	Whether hyperlinks should be live.
AddMovies	Whether active content such as movies should be added.
AddTags	Whether location of certain tags should be

	noted.
AdjustLayout	Whether HTML layout is checked and adjusted for optimal output to PDF.
AutoTruncate	Whether to automatically clip redundant content at the end of the page.
BreakMethod	The page break logic for HTML.
BreakZoneSize	The percentage of the current drawing area in which HTML breaks can occur.
BrowserWidth	The width of the virtual browser in pixels.

CoerceVector	The conditions under which to coerce a vector output.
ContentCount	The minimum number of content items required for a page to be valid.
DeactivateWebBrowser	Whether to deactivate the WebBrowser.
• DisableVectorCoercion	The conditions under which to disable vector-output coercion.
DoMarkup	Whether HTML pages are marked up before conversion to PDF.
FontEmbed	Whether fonts should be embedded

	rather than referenced.
FontProtection	Whether fonts should be protected.
FontSubset	Whether embedded fonts should be subsetted or not.
FontSubstitute	Whether font substitution should be used to reduce file size.
HideBackground	Whether to hide the background color of a page.
HostWebBrowser	Whether to host a WebBrowser control.
	The multicast delegate to be

HtmlCallback	called while HTML rendering is taking place.
HtmlEmbedCallback	The multicast delegate to be called when embedding an object while HTML rendering is taking place.
HttpAdditionalHeaders	Additional HTTP headers to send in the request.
ImageQuality	The quality of compression acceptable for continuous tone images such as JPEGs.
ImprovePageBreakAvoid	Whether to improve the support for page-break-inside of

	avoid.
ImprovePageBreakInTable	Whether to improve the support for page break in table to prevent missing rows.
InitialWidth	The minimum width to be used for auto-sized pages.
LogonName	A user name to be used for authentication.
LogonPassword	A password to be used for authentication.
MakeFieldNamesUnique	Whether field names should be changed to make them unique.
MaxAtomicImageSize	The maximum size at which an image may be regarded

	as unbreakable.
Media	The CSS media type to use.
NoCookie	Whether to disable automatic cookies.
NoDefaultBackground	Whether to disable the default background color.
NoSnapRounding	Whether to disable the snap rounding of coordinates and lengths.
NoTheme	Whether themes should be disabled.
OnLoadScript	A script to be run after the page is loaded.
	Whether the

PageCacheEnabled	page cache should be searched before rendering the page.
PageCacheExpiry	The length of time that pages can be held in the cache (ms).
PageCacheSize	The number of pages that can be held in the cache.
Paged	Whether content should be rendered in multipaged format.
PageLoadMethod	The method for loading URI/HTML.
ProcessOptions	The options for new worker processes.

ReloadPage	Whether to reload page.
RequestMethod	The request method for URL.
RetryCount	The number of times a page should be retried if unavailable or invalid.
TargetLinks	Whether hyperlinks should be allowed to open new windows.
Timeout	The maximum amount of time allowed for obtaining a page (ms).
TransferModule	The module for obtaining URI/HTML data.
	Whether to

UseActiveX	enable ActiveX.
UseJava	Whether to enable Java.
UseNoCache	Whether any proxy servers should re-request items of content.
UseResync	Whether to resynchronize pages.
UseScript	Whether to enable JavaScript and VBScript.
UseTheme	Whether to use themes.
UseVideo	Whether to enable Video.

# XForm Object

---

Represents the form and fields associated with this document.

The PDF specification makes a distinction between a field (in terms of a named value) and the visible appearance of the field on the document.

In general one field will have one visible appearance on the document. However it is possible for a field to have multiple or indeed no appearances on the document.

For example a Radio Button group encapsulates one value and is represented by one field. However each radio button has a separate appearance on the document. So one Radio Button field is likely to have many appearances.



For this reason fields exist within a hierarchy. Each field may or may not have a visible appearance. Each field may or may not have children. These children may be fields themselves and again may have their own children.

Fields are generally referenced by fully qualified name. A full name describes a path down through the field hierarchy - using periods as delimiters - to a specific field object. For example the name 'person.address.city' would reference a field with a partial field name called 'city' which has a parent called 'address' which has a parent at the top level called 'person'. You can resolve a fully qualified name using the [Item](#) function.

Once a field has been obtained you can query or change its values. If you wish to convert the fields into

a permanent part of the document you can use the [Stamp](#) method to permanently emboss them.

```
System.Object  
WebSupergoo.ABCpdf10.XForm
```

Method	Description
<a href="#">GetFieldNames</a>	Gets the full names of all the fields in the document.
<a href="#">MakeFieldsUnique</a>	Makes shared XForm fields unique.
<a href="#">AddResource</a>	Add a particular type of resource to the form.
<a href="#">Item</a>	Returns a particular field referenced by full name.
<a href="#">Refresh</a>	Refresh and reload the document fields.
<a href="#">Stamp</a>	Stamp all fields into the document.

<b>Property</b>	<b>Description</b>
<code>DateTimeFormat</code>	The format provider for formatting dates and times.
<code>Fields</code>	All top level fields in the form.
<code>FormatFields</code>	Whether values should be formatted before insertion into fields.
<code>GenerateAppearances</code>	Whether field appearances should be pre-generated.
<code>NeedAppearances</code>	Whether the viewer should automatically regenerate field appearances.



# Bookmark Class

---

A bookmark or outline item.

PDF documents typically provide a list of bookmarks for easy navigation between pages. In Acrobat this navigation structure is available under the Bookmarks tab. The PDF specification refers to this structure as the document outline.

The document outline comprises of a hierarchy of bookmarks. The bookmark at the top of the hierarchy is available via the [Doc.Bookmark](#) property. Typically a bookmark triggers navigation to a particular page but in some cases it may trigger a different and more complex type of action.



Every bookmark holds a collection of all its child bookmarks. As with all collections you can use the Count property to determine the number of items contained and you can iterate through the collection using the standard methods appropriate to the language you are coding in.

System.Object

[WebSupergoo.ABCpdf10.Objects.IndirectObject](#)

**WebSupergoo.ABCpdf10.Objects.Bookmark**

Implements: `IList<Bookmark>`,  
`ICollection<Bookmark>`, `IEnumerable<Bookmark>`,  
`IList`, `ICollection`, `IEnumerable`

Method	Description
<a href="#">CopyTo</a>	Copies the Bookmarks into an array.

Add	Adds a Bookmark to the end of the list.
Clear	Removes all Bookmarks from the list.
Contains	Determines whether the list contains a specific Bookmark.
IndexOf	Determines the index of a specific Bookmark.
• Insert	Inserts a Bookmark into the list at the specified position.
Remove	Removes a Bookmark from the list.
RemoveAt	Remove the Bookmark at the specified location.
GetEnumerator	Gets an enumerator for the Collection.
• Adopt	Adopt a specified Bookmark.
Refresh	Refresh and reload the document Bookmarks.
	inherited methods...

Property	Description
----------	-------------

Count	The number of Bookmarks in the collection.
• Item	Get or set the Bookmark at the specified index.
Open	Whether the bookmark appears open or closed.
Page	The destination Page associated with this bookmark.
PageID	The destination Page ID associated with this bookmark.
Parent	The parent of this Bookmark.
Title	The bookmark title to be displayed on screen.
	<a href="#">inherited properties...</a>

# XEncryption Object

---

Represents the current encryption settings.



By default encryption is not applied unless you explicitly specify it using the Type property.

```
System.Object  
WebSupergoo.ABCpdf10.XEncryption
```

Method	Description
• <a href="#">SetCryptMethods</a>	Sets the crypt methods for encryption levels of type 4 or above.

Property	Description
<a href="#">CanAssemble</a>	Whether a user can assemble the document.
<a href="#">CanChange</a>	Whether a user can modify the document.

CanCopy	Whether a user can copy from the document.
CanEdit	Whether a user can edit the document.
CanExtract	Whether a user can extract from the document.
CanFillForms	Whether a user can fill forms in the document.
CanPrint	Whether a user can print the document.
CanPrintHi	Whether a user can print a high resolution copy of the document.
	Whether to encrypt the document

EncryptMetadata	metadata for encryption levels of type 4 or above.
OwnerPassword	The owner password.
Password	The user password.
StreamCryptionMethod	The crypt method for streams for encryption levels of type 4 or above.
StringCryptionMethod	The crypt method for strings for encryption levels of type 4 or above.
Type	The level of encryption to use.

# ObjectSoup Class

---

A non-traditional collection of [Indirect Objects](#) making up the content of a PDF.

This collection has some of the characteristics of an Array and some of a Dictionary. Objects are referred to by index - starting at zero - like an Array. However they do not move within the list because they are identified by index - like a Dictionary with numeric keys.



We call this type of hybrid object a Soup. We assign it traditional interfaces to make it easier to use.

```
System.Object  
    WebSupergoo.ABCpdf10.Objects.ObjectSoup
```

```
Implements: IList<IndirectObject>,  
            ICollection<IndirectObject>,  
            IEnumerable<IndirectObject>, IList,  
            ICollection, IEnumerable, IDisposable
```

Method	Description
<a href="#">Dispose</a>	Dispose of the object.
<a href="#">CopyTo</a>	Copies the objects in the Soup to an Array.
<a href="#">Add</a>	Adds an object to the Soup.
<a href="#">Clear</a>	Removes all objects from the Soup.

Contains	Determines whether the Soup contains a specific object.
IndexOf	Determines the index of a specific object.
● Insert	Inserts an object into the Soup at the specified position.
Remove	Removes an object from the Soup.
● RemoveAt	Removes an object at a specified position from the Soup.
GetEnumerator	Gets an enumerator for the Soup.

Property	Description
Count	The number of items in the Soup.
● Item	Gets or sets the object at the specified index.
Catalog	The Catalog for the document.
Trailer	The Trailer or XRef for the document.

**Revisions** The number of incremental updates.

# XRendering Object

---

Provides control over PDF rendering and rendering options.

Note: rendering is only available under the ABCpdf Professional License.



Rendering is initiated by using the [Save](#) method. Properties of this object allow finer control over the way that rendering is performed.

System.Object

**WebSupergoo.ABCpdf10.XRendering**

Method	Description
• <a href="#">GetBitmap</a>	Renders the current area of the current page to a Bitmap.
<a href="#">GetData</a>	Renders the current area of the current page to memory.
<a href="#">Save</a>	Renders and saves the current area of the current page.

Property	Description
----------	-------------

AntiAliasImages	Whether to anti-alias images.
AntiAliasPolygons	Whether to anti-alias polygons.
AntiAliasScene	Whether to apply entire scene anti-aliasing.
AntiAliasText	Whether to anti-alias text.
AutoRotate	Whether pages should be automatically rotated.
BitsPerChannel	The output bits per color channel.
ColorSpace	The name of the output color space.
DefaultHalftone	Halftone type and options.
	The output resolution in

DotsPerInch	dots per inch (DPI).
DotsPerInchX	The horizontal output resolution in dots per inch (DPI).
DotsPerInchY	The vertical output resolution in dots per inch (DPI).
DrawAnnotations	Whether to render fields and annotations.
IccCmyk	The path to the default CMYK ICC color profile.
IccGray	The path to the default Gray ICC color profile.
IccOutput	The path to the default output ICC

	color profile.
IccRgb	The path to the default RGB ICC color profile.
Log	The log for the last render.
Metadata	A collection of TIFF tags that should be written to the output file.
MinimumLineWidth	The minimum stroked line width for output.
NamedSeparation	Named separations.
NumTiffStrips	Number of strips to generate when writing a tiff file.
Overprint	Whether to apply overprint.

PaletteSize	The number of colors in the palette for indexed color output.
ResizeImages	Whether to resize images for vector output.
SaveAlpha	Whether to save the alpha channel into the output.
SaveAppend	Whether to append to (rather than overwrite) existing image files.
SaveCompression	The preferred compression method.
SaveQuality	The output file quality for lossy compression.

UseEmbeddedHalftone

Whether to  
use  
embedded  
halftones.

# XSaveOptions Object

---



Represents the current save settings.

```
System.Object  
WebSupergoo.ABCpdf10.XSaveOptions
```

Method	Description
none	

Property	Description
<a href="#">CompressObjects</a>	Whether to use object stream compression to reduce file size.
<a href="#">EmbeddedGraphics</a>	The format to be used for exporting PDF vector graphics.
<a href="#">FileExtension</a>	Gets or sets the file extension for the target when it is not otherwise specified.

Folder	The folder where to save additional files (images, fonts, etc).
FontSubstitution	The font substitution type.
IDConstant	Whether to assign a constant file version identifier.
IDHexadecimal	Whether to assign non-ASCII file identifiers.
IDUpdate	Whether to update the file version identifier.
Incremental	Whether to use incremental update to preserve an audit trail.
Linearize	Whether to linearize the output for fast web viewing.

Remap	Whether to reduce size by remapping objects.
SaveQuality	The XPS output quality for lossy compression.
Template	The path to the template file.
TemplateData	The template data.
WritePageSeparator	The delegate called to write the page separator for export.

# XTextStyle Class

---

Represents the style used when adding text.



All measurements are specified using the current `Doc.Units`.

```
System.Object  
WebSupergoo.ABCpdf10.XTextStyle
```

Method	Description
<code>ToString</code>	Returns a string representation of the object.

Property	Description
<code>Ascender</code>	An adjustment to allow the text ascender to coincide with the top of the text area.
<code>Bold</code>	Whether to apply a synthetic bold effect.
<code>CharSpacing</code>	The inter-character spacing.
<code>CharUsage</code>	The usage of

	characters.
Direction	The default text direction.
Font	The current Font ID.
HPos	The current horizontal positioning factor (0 to 1).
Indent	The first line of paragraph indent.
Italic	Whether to apply a synthetic italic effect.
Justification	The horizontal justification factor.
Kerning	The kerning method.
LeftMargin	The paragraph indent.
LineSpacing	The inter-line spacing.
Outline	The width of character outlining.
ParaSpacing	The inter-paragraph spacing.
PreserveSpace	Whether to preserve white space characters for plain text.
Size	The current text size.

Strike	Whether to apply a strikethrough effect.
Strike2	Whether to apply a double strikethrough effect.
String	The text style as a string.
Underline	Whether to underline text.
VPos	The current vertical positioning factor (0 to 1).
WordSpacing	The inter-word spacing.

# ArrayAtom Class

---

An Atom containing an array of other Atoms.



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.ArrayAtom
```

```
Implements: IList<Atom>,
             ICollection<Atom>, IEnumerable<Atom>,
             IList, ICollection, IEnumerable
```

Method	Description
<b>S»</b> FromXRect	Create an ArrayAtom from a XRect representation.
<b>S»</b> FromXTransform	Create an ArrayAtom from a XTransform representation.
<b>S»</b> FromContentStream	Create an array of Atoms from a byte array containing a sequence of PDF objects.

ArrayAtom	ArrayAtom Constructor.
CopyTo	Copies the Atoms into an array.
Add	Add an item to the end of the array.
Clear	Removes all Atoms from the array.
Contains	Determines whether the array contains a specific Atom.
IndexOf	Determines the index of a specific Atom.
• Insert	Inserts an Atom into the array at the specified position.
Remove	Removes an Atom from the array.
	Removes an Atom at a

● RemoveAt	specified position from the array.
● AddRange	Adds the elements in the supplied array at the end of this array.
Equals	Test whether the two ArrayAtoms are the same.
GetEnumerator	Gets an enumerator for the Collection.
● GetRange	Creates a shallow copy of a range of elements in the source array.
● InsertRange	Inserts the elements in the supplied array into this array at the specified index.
● RemoveRange	Removes a range of elements from

	the source array.
	inherited methods...

<b>Property</b>	<b>Description</b>
Count	The number of Atoms in the array.
• Item	Get or set the Atom at the specified index.
	inherited properties...

# NumAtom Class

---

An Atom representing a numeric value.



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.NumAtom
```

Method	Description
<b>S»</b> Encode	Encode a number into a PDF string. This format may be needed for direct insertion into a content stream.
Equals	Test whether the two NumAtoms are the same.
NumAtom	Construct a NumAtom.
	<a href="#">inherited methods...</a>

Property	Description
Num	The integer value of the number.
Num64	The 64-bit integer value of the number.

Real	The floating point value of the number.
	inherited properties...

# Atom Class

---

A PDF atomic object comprising a basic chunk of PDF data.

If you are going to be using Atoms then you will want to download the [Adobe PDF Specification](#). This document explains the names and types that can be used and how they are interpreted.

## Common Operations.

Get a named property from the dictionary of an IndirectObject. In the example below we use the Type property which is typically a name but the principles are similar for other entries of other types:

### [C#]

```
NameAtom type =  
io.Resolve(Atom.GetItem(io.Atom,  
"Type")) as NameAtom;
```

### [Visual Basic]

```
Dim type As NameAtom =  
io.Resolve(Atom.GetItem(io.Atom,  
"Type"))
```

Get an value out of an array atom. In the the event that the atom is not an array or does not have sufficient entries then the return value will be null. In the example below we are looking for entry two - this is the third entry since entries are zero based:

### [C#]

```
NumAtom num =  
io.Resolve(Atom.GetItem(atom, 2)) as  
NumAtom;
```

### **[Visual Basic]**

```
Dim num As NumAtom =  
io.Resolve(Atom.GetItem(atom, 2))
```

Get a stream referenced from a property of an IndirectObject. In the example below we use the FontFile2 property (a reference to an embedded TrueType font):

### **[C#]**

```
StreamObject stream =  
io.ResolveObj(Atom.GetItem(io.Atom,  
"FontFile2")) as StreamObject;
```



### **[Visual Basic]**

```
Dim stream As StreamObject =  
io.ResolveObj(Atom.GetItem(io.Atom,  
"FontFile2"))
```

Add a named entry to an IndirectObject. In the example below we add a V entry which is a string. We keep the returned StringAtom so we can manipulate the value:

### **[C#]**

```
StringAtom str =  
(StringAtom)Atom.SetItem(io.Atom, "V",  
new StringAtom());
```

### [Visual Basic]

```
Dim str As StringAtom =  
Atom.SetItem(io.Atom, "V", New  
StringAtom())
```

Add a named entry to an IndirectObject. In the example below we add an array entry to specify a border array. Rather than creating an ArrayAtom and specifying the individual values we just specify the raw string value of the object:

### [C#]

```
Atom.SetItem(io.Atom, "Border",  
Atom.FromString("[ 0 0 1 ]"));
```

### [Visual Basic]

```
Atom.SetItem(io.Atom, "Border",  
Atom.FromString("[ 0 0 1 ]"))
```

System.Object

**WebSupergoo.ABCpdf10.Atoms.Atom**

WebSupergoo.ABCpdf10.Atoms.ArrayAtom

WebSupergoo.ABCpdf10.Atoms.BoolAtom

WebSupergoo.ABCpdf10.Atoms.DictAtom

WebSupergoo.ABCpdf10.Atoms.NameAtom

WebSupergoo.ABCpdf10.Atoms.NullAtom

WebSupergoo.ABCpdf10.Atoms.NumAtom

WebSupergoo.ABCpdf10.Atoms.RefAtom

WebSupergoo.ABCpdf10.Atoms.OpAtom  
WebSupergoo.ABCpdf10.Atoms.StringAtom

Implements: ICloneable, IDisposable,  
IEquatable<Atom>, IComparable<Atom>

Method	Description
<b>S»</b> FromString	Create an appropriate type of Atom from a raw PDF string representation.
<b>S»</b> GetBool	Gets the Boolean value from the Atom if it is a BoolAtom.
<b>S»</b> GetDouble	Gets the double value from the Atom if it is a NumAtom.
<b>S»</b> GetID	Gets the Object ID value from the Atom if it is a RefAtom.
<b>S»</b> GetInt	Gets the integer value from the Atom if it is a NumAtom.
<b>S»</b> GetItem	Gets the specified item from the Atom if it is of a type which contains other Atoms.

<b>S»</b> GetName	Gets the Name value from the Atom if it is a NameAtom.
<b>S»</b> GetText	Gets the Text value from the Atom if it is a StringAtom.
<b>S»</b> RemoveItem	Removes the named entry from the Atom if it is a DictAtom.
<b>S»</b> SetItem	Adds a specified item to the Atom if it is of a type which contains other Atoms.
GetData	The byte array representation of the Atom as it would appear in a PDF.
Clone	Creates a deep copy of the current Atom.
Dispose	Dispose of the object.
Equals	Test whether the two Atoms are the same.
GetHashCode	A hash code for the Atom.
ToString	The string representation of the Atom as it would

appear in a PDF.

<b>Property</b>	<b>Description</b>
none	

# XFont Class

---



Provides information about the fonts available to ABCpdf.

System.Object  
**WebSupergoo.ABCpdf10.XFont**

Method	Description
<b>S»</b> <a href="#">FindAll</a>	Find all the fonts currently installed on the system.
<b>S»</b> <a href="#">FindFamily</a>	Find all the fonts belonging to a particular family.
<b>S»</b> <a href="#">FindFamilyNames</a>	Find the names of all font families.
<b>S»</b> <a href="#">FindByName</a>	Find all the fonts with a given name.
<b>S»</b> <a href="#">FindByStyle</a>	Find a font from a specific family, with a given style.
<a href="#">TextWidth</a>	Calculate the width of a string of text.
<a href="#">Unload</a>	Unloads a font so that it is no longer available.

---

---

<b>Property</b>	<b>Description</b>
FamilyName	The name of the family to which the font belongs.
FixedPitch	Whether the font is in a fixed pitch style.
Italic	Whether the font is in an italic style.
Name	The complete human readable name of the font.
Names	The complete set of names by which this font is known.
PostScriptName	The name the font will be known by on a PostScript printer.
Script	Whether the font is in a script style.
Serif	Whether the font is in a serif style.
Weight	The weight of the font.
	The baseline of the

Baseline	font in thousandths of a unit.
Widths	The widths of the characters in the font.

# Field Class

---

Represents a field in a document.

The PDF specification makes a distinction between a field (in terms of a named value) and the visible appearance of the field on the document.

So not every field has a visible appearance. Those that do can be located using the [Page](#) and [Rect](#) properties. The value of the field can be modified using the [Value](#) property.

Fields exist within a hierarchy. Fields have children and their children can have children in turn. You can drill down through the hierarchy using the [Kids](#) property.



Alternatively - given a fully qualified name - you can use the [XForm](#) level methods to obtain references directly.

Note that the Field object is an abstraction of the underlying objects defined in the document. As such the Object ID may refer to an Annotation Widget or it may refer to a Field or it may refer to a hybrid Widget Field as defined in the PDF specification.

System.Object

[WebSupergoo.ABCpdf10.Objects.IndirectObject](#)

**[WebSupergoo.ABCpdf10.Objects.Field](#)**

Method	Description
<a href="#">Focus</a>	Prepare document for drawing at the field location.

GetAnnotations	Gets all the Annotations referenced by this field or its children.
GetKids	Gets a set of Fields that are descendents of this one.
SetFont	Sets the font and font size to be used for text.
Stamp	Stamp this field into the document.
UpdateAppearance	Update the appearance of all the Annotations associated with this field.
	<a href="#">inherited methods...</a>

Property	Description
DefaultAppearance	The default appearance (DA) used for the text.
FieldType	The field type.
Flags	The Field Flags (Ff)

	entry.
Format	The field format.
Kids	All the immediate children of this field.
MultiSelect	Whether the field supports multiple selections.
Name	The fully qualified field name.
Options	The field options.
Page	The Page on which the field appears.
PageID	The ID of the page on which the field appears.
Parent	The parent of this field.
PartialName	The partial field name.
• Rect	The location and size of the field.
TextAlignment	The alignment for the text.
TextColor	The color used for the text.
TextFont	The font used for the

	text.
TextSize	The font size used for the text.
Value	The field value.
	inherited properties...

# Fields Class

---

A class encapsulating a collection of Fields.



System.Object

**WebSupergoo.ABCpdf10.Objects.Fields**

Implements: IList<Field>,  
ICollection<Field>,  
IEnumerable<Field>, IList,  
ICollection, IEnumerable

Method	Description
CopyTo	Copies the items into a collection.
• Add	Add an item to the end of the collection.
• Clear	Removes all items from the collection.
Contains	Determines whether the collection contains a specific item.
IndexOf	Determines the index of a specific item.
• Insert	Inserts an item into the collection at the specified position.

● Remove	Removes an item from the collection.
● RemoveAt	Removes an item at a specified position from the collection.
GetEnumerator	Gets an enumerator for the Collection.

<b>Property</b>	<b>Description</b>
Count	The number of items in the collection.
● Item	Get or set the item at the specified index.

# XHtmlProcessOptions Class

---



Represents the options for new HTML worker processes.

System.Object

**WebSupergoo.ABCpdf10.XHtmlProcessOptions**

Method	Description
StartPool	Start the process pool for the HTML Engine.

Property	Description
PoolHasStarted	Whether the process pool for the HTML Engine has started.
ProcessCount	The number of existing processes for the HTML Engine.
Domain	The domain or server containing the user.
IdleTimeout	The maximum time a process can be idle before being terminated (ms).

LoadUserProfile	Whether to load the user profile.
Password	The password for the user.
RetryCount	The number of times an operation should be retried if there is a problem with the process.
UserName	The user name.

# XpsImportOperation Class

---

An operation used to import XPS and OXPS documents.



```
System.Object
  WebSupergoo.ABCpdf10.Operations.Operation
    WebSupergoo.ABCpdf10.Operations.XpsImport
```

Method	Description
<b>S»</b> GetApplicationFolder	Gets the installation folder of an import application.
XpsImportOperation	XpsImportOperation Constructor.
ApplicationIsRunning	Gets a value indicating whether an import application is running for this XpsImportOperation.
EnableApplicationReuse	Enables an import application process to be reused so that it is terminated only manually or when this operation is disposed of.
• Import	Imports a portion of an XPS document.
	Imports a portion of an

ImportAny	document via an intermediate XPS print out.
KillApplication	Terminates a running import application.
ReusesApplication	Gets a value indicating whether an import application process is to be kept running and reused.

Property	Description
Application	The application used by ImportAny.
ApplicationFolder	The folder of the application used by ImportAny.
DocNumber	The one-based document number used by ImportAny.
EnableMSOfficeMacros	Whether to enable macros when opening MS Office documents.
Password	The password needed to read the source.



# Operation Class

---

An operation to be performed on a document.



System.Object

## **WebSupergoo.ABCpdf10.Operations.Operation**

WebSupergoo.ABCpdf10.Operations.Accessibi  
WebSupergoo.ABCpdf10.Operations.EffectOpe  
WebSupergoo.ABCpdf10.Operations.FlattenTr  
WebSupergoo.ABCpdf10.Operations.ImageOpe  
WebSupergoo.ABCpdf10.Operations.PdfConfor  
WebSupergoo.ABCpdf10.Operations.PdfValida  
WebSupergoo.ABCpdf10.Operations.RecolorOp  
WebSupergoo.ABCpdf10.Operations.ReduceSiz  
WebSupergoo.ABCpdf10.Operations.RenderOpe  
WebSupergoo.ABCpdf10.Operations.SwfImport  
WebSupergoo.ABCpdf10.Operations.TextOpera  
WebSupergoo.ABCpdf10.Operations.XpsImport

Implements: IDisposable

Method Description	
none	

Property Description	
none	

--	--

<b>Event</b>	<b>Description</b>
ProcessingObject	Occurs before an IndirectObject on.
ProcessedObject	Occurs after an IndirectObject operated on.

# XSaveTemplateData Class

---



Represents some format-specific data not obtained from a template file.

System.Object  
**WebSupergoo.ABCpdf10.XSaveTemplateData**

Method	Description
<a href="#">SetMeasureResolution</a>	Sets the measurement resolutions.

Property	Description
<a href="#">ImageDisplaySmoothing</a>	The smoothing for displaying images.
<a href="#">JpegQuality</a>	The JPEG quality level.
<a href="#">MeasureDpiX</a>	The horizontal measurement resolution.

MeasureDpiY	The vertical measurement resolution.
ReencodeJpeg	Whether JPEG images are re-encoded in JPEG.

# IndirectObject Class

---

A PDF indirect object.

PDF supports a number of basic types of object. Objects may be so that they can be referred to by other objects. This type of object called an indirect object.

ABCpdf uses a slightly different terminology to avoid clashes with namespaces. Objects without labels contain basic data elements are referred to as [Atoms](#).

The term 'indirect object' is often shortened to 'PDF object' or son simply 'object'. There are a variety of types of PDF objects from c layers to fonts to annotations. Every object contains an Atom whi represents the data held by that object.

Indirect objects are held in an [ObjectSoup](#). This is a non-tradition collection with some of the characteristics of an Array and some ( Dictionary.



```
System.Object
```

```
WebSupergoo.ABCpdf10.Objects.IndirectObject
```

```
WebSupergoo.ABCpdf10.Objects.Annotation
```

```
WebSupergoo.ABCpdf10.Objects.Bookmark
```

```
WebSupergoo.ABCpdf10.Objects.Catalog
```

```
WebSupergoo.ABCpdf10.Objects.ColorSpace
```

```
WebSupergoo.ABCpdf10.Objects.Field
```

```
WebSupergoo.ABCpdf10.Objects.FileSpecific
```

```
WebSupergoo.ABCpdf10.Objects.FontObject
```

```
WebSupergoo.ABCpdf10.Objects.Outline
```

```
WebSupergoo.ABCpdf10.Objects.Page
```

```
WebSupergoo.ABCpdf10.Objects.Pages
```

```
WebSupergoo.ABCpdf10.Objects.StreamObject
```

```
Implements: ICloneable, IDisposable,  
IEquatable<IndirectObject>
```

---

Method	Description
<b>S»</b> FromString	Construct an appropriate type of IndirectObject given a string value.
IndirectObject	IndirectObject Constructor.
Dispose	Dispose of the object.
Clone	Create a deep copy of the current IndirectObject.
Equals	Test whether the two IndirectObjects are the same.
GetHashCode	A hash code for the IndirectObject.
Resolve	Resolves any indirect references and returns the Atom.
ResolveRef	Resolves any indirect references and returns the final RefAtom.
ResolveObj	Resolves any indirect references and returns the final IndirectObject.
ToString	The string representation of the IndirectObject.
Transcode	Transcodes and reloads the IndirectObject.

<b>Property</b>	<b>Description</b>
ID	The ID of the PDF object.
Gen	The Generation of the PDF object.
Atom	The Atom contained by the IndirectObject.
Version	The minimum version of the PDF specification required to support this object.
Revision	The revision of the document in which this object is stored.
Alive	Whether the IndirectObject is currently alive.
Doc	The Doc containing this IndirectObject.
Soup	The ObjectSoup containing this IndirectObject.
AdbeExtLevel	The minimum extension level of Adobe Supplement to the PDF specification required to support this object.

# NullAtom Class

---

A null Atom.



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.NullAtom
```

Method	Description
<a href="#">Equals</a>	Test whether the two NullAtoms are the same.
<a href="#">NullAtom</a>	Construct a NullAtom.
	<a href="#">inherited methods...</a>

Property	Description
none	
	<a href="#">inherited properties...</a>

# NameAtom Class

---

An Atom representing a name.



```
System.Object  
  WebSupergoo.ABCpdf10.Atoms.Atom  
    WebSupergoo.ABCpdf10.Atoms.NameAtom
```

Method	Description
<a href="#">Equals</a>	Test whether the two NameAtoms are the same.
<a href="#">NameAtom</a>	Construct a NameAtom.
	<a href="#">inherited methods...</a>

Property	Description
<a href="#">Text</a>	The text of the name.
	<a href="#">inherited properties...</a>

# RefAtom Class

---

An Atom which references an [IndirectObject](#).



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.RefAtom
```

Method	Description
<a href="#">RefAtom</a>	Construct a RefAtom.
<a href="#">Assign</a>	Set the IndirectObject that the reference should point to.
<a href="#">Equals</a>	Test whether the two RefAtoms are the same.
<a href="#">Resolve</a>	Get the IndirectObject that the reference is pointing to.
	<a href="#">inherited methods...</a>

Property	Description
<a href="#">ID</a>	The ID of the referenced IndirectObject.
<a href="#">Gen</a>	The generation of the referenced IndirectObject.
	<a href="#">inherited properties...</a>



# Catalog Class

---

The Catalog for the document.



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
    WebSupergoo.ABCpdf10.Objects.Catalog
```

Method	Description
AnalyzeContent	Perform whole document analysis of page contents.
GetEmbeddedFiles	Gets all the embedded files in this document.
GetFieldNames	The names of all the eForm fields in the document.
GetFields	Get all the eForm fields in the document.
GetFonts	Gets all the fonts in this document.
	<a href="#">inherited methods...</a>

<b>Property</b>	<b>Description</b>
<b>Outline</b>	The root Outline object.
<b>Pages</b>	The root Pages object.
	<b>inherited properties...</b>

# Pages Class

---

A Pages node within the document.



```
System.Object  
  WebSupergoo.ABCpdf10.Objects.IndirectObject  
    WebSupergoo.ABCpdf10.Objects.Pages
```

Method	Description
GetPage	Performs a fast lookup to retrieve a particular Page from this node tree.
GetPageArray	Gets all the Page objects immediately under this node.
GetPageArrayAll	Gets all the Page objects under this node and descendents of this node.
• Recolor	Converts the pages from one color space to another.
	inherited methods...

<b>Property</b>	<b>Description</b>
Count	The number of visible pages under this node.
Parent	The parent of this node.
	<a href="#">inherited properties...</a>

# Outline Class

---

The top level outline item.



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
  WebSupergoo.ABCpdf10.Objects.Bookmark
  WebSupergoo.ABCpdf10.Objects.Outline
```

Method	Description
--------	-------------

	inherited methods...
--	----------------------

Property	Description
----------	-------------

	inherited properties...
--	-------------------------

# StreamObject Class

---

A PDF data stream.



System.Object

WebSupergoo.ABCpdf10.Objects.IndirectObject

**WebSupergoo.ABCpdf10.Objects.StreamObject**

WebSupergoo.ABCpdf10.Objects.EmbeddedF

WebSupergoo.ABCpdf10.Objects.FormXObject

WebSupergoo.ABCpdf10.Objects.IccProfile

WebSupergoo.ABCpdf10.Objects.Layer

WebSupergoo.ABCpdf10.Objects.Pixmap

Method	Description
• StreamObject	StreamObject Constructor.
ClearData	Clear the data and compression settings for the stream.
ClearCachedDecompressed	Clear the cached, decompressed data for the stream.
	Compress the

Compress	data in the stream.
CompressAscii85	Compress the data in the stream using ASCII 85 encoding.
CompressAsciiHex	Compress the data in the stream using the ASCII Hex encoding.
CompressFlate	Compress the data in the stream using Flate compression.
CompressRunLength	Compress the data in the stream using run length encoding.
Decompress	Decompress the data in the stream.
GetData	Get the raw binary content of the stream.

GetText	Get the content of the stream as a string.
SetData	Set the raw binary content of the stream.
• SetFile	Set the raw binary content of the stream using data from a file.
SetText	Set the content of the stream as a string.
	<a href="#">inherited methods...</a>

Property	Description
Compressed	Whether the stream data is compressed or otherwise encoded.
Compression	The primary compression type.
Compressions	All the compression types applied to the stream.
Length	The number of bytes of

encoded stream data.

inherited properties...

## DictAtom Class

---

An Atom containing a dictionary of other Atoms indexed by name.



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.DictAtom
```

```
Implements: IDictionary<string, Atom>,
ICollection<KeyValuePair<string, Atom>>,
IEnumerable<KeyValuePair<string, Atom>>,
IDictionary, ICollection, IEnumerable
```

Method	Description
DictAtom	Construct a DictAtom.
CopyTo	Copies the Atoms into an array.
• Add	Add an item to the dictionary.
Clear	Removes all elements from the dictionary.
Contains	Determines whether the dictionary contains an element with a specific name.
Remove	Remove an element

	from the dictionary.
GetKeys	Get an array of all the names in the dictionary.
GetValues	Get an array of all the Atoms in the dictionary.
GetEnumerator	Get an enumerator for the dictionary.
Equals	Test whether the two DictAtoms are the same.
	inherited methods...

Property	Description
Count	Get the number of elements in the dictionary.
• Item	Get or set the entry with the specified name.
Keys	Get the collection of the keys in the dictionary.
Values	Get the collection of the Atoms in the dictionary.
	inherited properties...



# Page Class

---

A visible page within the document.



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
    WebSupergoo.ABCpdf10.Objects.Page
```

Method	Description
• DeInline	Makes any inline images into external images.
Detach	Detaches a content stream from the page.
Flatten	Flatten and compress the page content stream.
GetAnnotations	Gets all the Annotations on the page.
GetLayers	Gets all the content Layers for the page.
	Gets all the

GetNamedSeparations	named separations referenced by the page.
GetResourcesByType	Get all the resources of a named type, optionally including any used by referenced objects.
GetResourceMap	Get a dictionary mapping the names of a particular type of resource to Atoms.
GetText	Extract content from the current page in a specified format.
● Recolor	Converts the page from one color space to another.
● VectorizeText	Replaces the text on the page with

	glyph outlines.
AddLayer	Add a content layer at the front of the page.
AddResource	Add a particular type of resource to the page.
● GetBitmap	Render one or more layers on the current page.
MakeFormXObject	Makes a FormXObject out of the page.
StampFormXObjects	Removes all FormXObjects from the page by embedding them into the page content.

Property	Description
ArtBox	The ArtBox for the page.
BleedBox	The BleedBox for the page.
● CropBox	The CropBox for the page.
● MediaBox	The MediaBox for the page.

PageNumber	The number of the page in the document.
Parent	The parent of this page.
Rotation	The number of degrees to rotate the page before display.
Thumbnail	The Thumbnail for the page.
TrimBox	The TrimBox for the page.

# Annotation Class

---

An Annotation such as a text note or hyperlink.



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
    WebSupergoo.ABCpdf10.Objects.Annotation
```

Method	Description
Focus	Prepare document for drawing at the annotation location.
GetFieldOptions	The field options for any form field associated with this annotation.
Stamp	Stamp this annotation into the page.
UpdateAppearance	Update the Appearance Stream for this annotation.
	inherited methods...

--	--

<b>Property</b>	<b>Description</b>
Border	The border appearance.
Contents	The visible text of the annotation.
FieldBackgroundColor	The background color of the field.
FieldBorderColor	The border color of the field.
FieldRotation	The rotation of the annotation in degrees counterclockwise to the page.
FieldType	The field type for any form field associated with this annotation.
FieldValue	The field value for any form field associated with this annotation.
Flags	The Annotation Flags entry.
FullName	The full name of any form field associated with

	this annotation.
Page	The Page on which this annotation is located.
• Rect	The rectangle which defines the position and area of the annotation on the page.
SubType	The sub-type of Annotation.
TextDirection	The default text direction.
	inherited properties...

# FontObject Class

---

A specific font as used in the document.



```
System.Object  
  WebSupergoo.ABCpdf10.Objects.IndirectObject  
    WebSupergoo.ABCpdf10.Objects.FontObject
```

Method	Description
<b>S»</b> • <a href="#">GetEncoding</a>	Obtain a standard encoding dictionary for use with PDF text operators.
<a href="#">EmbedFont</a>	Search for and embed a font file into this font object.
• <a href="#">EncodeText</a>	Encode text for use with PDF text operators.
<a href="#">RegenerateToUnicode</a>	Attempt to regenerate a ToUnicode map.
<a href="#">Subset</a>	Subset a previously

	embedded font.
<b>UnembedSimpleFont</b>	Unembed the font if this is a simple operation.

<b>Property</b>	<b>Description</b>
<b>BaseFont</b>	The PostScript name of the font.
<b>EmbeddedFont</b>	The embedded font file.
<b>IsComposite</b>	Whether the font is a complex composite font.
<b>IsIdentity</b>	Whether the font is a composite glyph encoded identity font.
<b>IsSubset</b>	Whether the font contains an embedded subset.
<b>IsVertical</b>	Whether the font is for vertical writing.
<b>WritingMode</b>	Gets the font writing mode.
<b>Widths</b>	The widths of the characters in the font.

CharToEncoding	The Unicode to glyph mapping table for all the characters in the font.
CheckGlyphs	Whether to exclude invalid glyphs from our lookup tables.
EncodingToChar	The glyph to Unicode mapping table for all the characters in the font.
EncodingToString	The glyph to Unicode string mapping table.
Flags	The Font Descriptor Flags entry.
FontAscender	The ascender for the glyphs in this font.
FontAscent	The ascent for the glyphs in this font.
FontBBox	The bounding box for the glyphs in this font.
FontDescender	The descender for the glyphs in this font.
FontDescent	The descent for the glyphs in this font.
FontLineGap	The line gap for the glyphs in this font.

FontLineSpacing

The line spacing for the glyphs in this font.

# IccProfile Class

---

An ICC Color Profile for a particular [ColorSpace](#).



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
  WebSupergoo.ABCpdf10.Objects.StreamObject
  WebSupergoo.ABCpdf10.Objects.IccProfile
```

Method	Description
• <a href="#">IccProfile</a>	IccProfile Constructor.
• <a href="#">SetData</a>	Set the raw binary content of the stream.
<a href="#">UpdateProfile</a>	Updates this object to reflect the values contained within the embedded ICC color profile data.
	<a href="#">inherited methods...</a>

Property	Description
<a href="#">AlternateColorSpaceType</a>	The alternate color space for this ICC color profile.

inherited  
properties...

# ReduceSizeOperation Class

---

Operation to reduce the size of a document.



System.Object

WebSupergoo.ABCpdf10.Operations.Operation

**WebSupergoo.ABCpdf10.Operations.ReduceSizeOperation**

Method	Description
• <a href="#">ReduceSizeOperation</a>	ReduceSizeOperation Constructor.
• <a href="#">Compact</a>	Compact and compress document.

Property	Description
<a href="#">CompressImages</a>	Whether to resample and recompress images where possible.
<a href="#">CompressStreams</a>	Whether to compress uncompressed streams where possible.

ColorImageCompression	The target compression t for the re-enco of color image
ColorImageDpi	The target resolution for t resampling of images.
ColorImageQuality	The target compression quality for the encoding of cc images.
GrayImageCompression	The target compression t for the re-enco of grayscale images.
GrayImageDpi	The target resolution for t resampling of grayscale ima
GrayImageQuality	The target compression quality for the encoding of grayscale ima

MonochromeImageCompression	The target compression t for the re-enco of monochrome images.
MonochromeImageDpi	The target resolution for t resampling of monochrome images.
MonochromeImageQuality	The target compression quality for the encoding of monochrome images.
PageContents	The pages to operated upon
PalettizationTolerance	The amount o divergence fro the target pale which will be allowed.
RefactorImages	Whether to ref and remove duplicate imag where possible

UnembedComplexFonts	Whether to unembed complex Unicode based fonts where possible.
UnembedCorruptFonts	Whether to unembed embedded fonts that appear to be corrupt or non-standard.
UnembedSimpleFonts	Whether to unembed simple Latin based fonts where possible.
UnembedUnusualFonts	Whether to unembed embedded fonts that do not have obvious substitutes on the local machine.

# Layer Class

---

A layer of visible content on a page.

Each time content is added to a page a new layer is created. Different kinds of layers are created for different types of content. For example the `AddText` method will result in the creation of a `TextLayer` and the `AddImage` one will produce an `ImageLayer`.

A layer corresponds to a stream object referenced from the `Page Contents` array. The `Page.Flatten` method concatenates all the streams then compresses them to save space.



Note that this type of layer is an ABCpdf construct that you cannot use using Acrobat. Acrobat layers are something completely different more precisely known as Optional Content Groups (OCGs). See the [Doc.Layer](#) property for details of how to construct and manipulate this type of optional layer.

`System.Object`

`WebSupergoo.ABCpdf10.Objects.IndirectObject`

`WebSupergoo.ABCpdf10.Objects.StreamObject`

**`WebSupergoo.ABCpdf10.Objects.Layer`**

`WebSupergoo.ABCpdf10.Objects.Graphic`

`WebSupergoo.ABCpdf10.Objects.ImageLayer`

`WebSupergoo.ABCpdf10.Objects.TextLayer`

`WebSupergoo.ABCpdf10.Objects.ViewLayer`

Method	Description
--------	-------------

	<a href="#">inherited methods...</a>
--	--------------------------------------

<b>Property</b>	<b>Description</b>
BaseRect	The untransformed rect defining the bounds of the visible content on the page.
Page	The Page on which the Layer is located.
Rect	The transformed rect defining the bounds of the visible content.
Transform	The transform which has been applied to the visible content.
	<a href="#">inherited properties...</a>

# ColorSpace Class

---

A color space for an object such as a [PixMap](#).



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
    WebSupergoo.ABCpdf10.Objects.ColorSpace
```

Method	Description
<a href="#">ColorSpace</a>	Construct a ColorSpace.
	inherited methods...

Property	Description
<a href="#">ColorSpaceType</a>	The type of color space.
<a href="#">Components</a>	The number of color components in the color space.
<a href="#">IccProfile</a>	Any ICC Color Profile associated with this color space.
<a href="#">BaseColorSpaceType</a>	The base color space of an

	Indexed color space.
Name	Any name associated with the color space.
Gamma	The gamma correction for the color space.
BlackPoint	The black point for the color space specified in CIE 1931 XYZ space.
WhitePoint	The white point for the color space specified in CIE 1931 XYZ space.
	inherited properties...

## TextLayer Class

---

A multistyled text layer appearing on a page of the document.



System.Object

WebSupergoo.ABCpdf10.Objects.IndirectObject

WebSupergoo.ABCpdf10.Objects.StreamObject

WebSupergoo.ABCpdf10.Objects.Layer

**WebSupergoo.ABCpdf10.Objects.TextLa**

Method	Description
ClearTextOperation	Removes any cached TextOperation and TextFragments associated with this object.
	inherited methods...

Property	Description
Characters	The number of characters appearing on the page.
EndPos	The point defining the end position of the text.
Lines	The number of lines appearing on the page.

Previous	The previous text object in the text chain.
Truncated	Whether the text had to be truncated.
FullText	The full text provided in the initial call to AddHtml or AddText.
TextStart	The offset to the first character drawn onto this layer.
TextEnd	The offset to the last character drawn onto this layer.
TextEnd	The offset to the character which will be drawn at the start of next item in the chain.
TextFragments	The TextFragments describing the precise layout of this text layer.
TextOperation	A TextOperation describing the precise layout of this text layer.
	inherited properties...

# TextFragment Class

---

A fragment of text as it appears in a document.

Each TextFragment spans a part of a PDF stream drawing operator. The part may be the entire of the text drawn by the operator or it may be a section of the text within that operator.



For example the PDF text drawing operator "(Once upon a time) Tj" draws that text. If you search for the word "upon" the TextFragment which is returned will reference the complete operator via the [StreamOffset](#) and [StreamLength](#) properties but properties like the [Text](#) and [Rect](#) will span only the word "upon" within that operator.

System.Object

**WebSupergoo.ABCpdf10.Operations.TextFragment**

Method	Description
none	

Property	Description
<a href="#">Font</a>	The font object used for drawing this text fragment.
<a href="#">FontColor</a>	The color used for drawing this text

	fragment.
FontSize	The effective font size used for drawing this text fragment.
FontObliqueAngle	The skew angle of the font as used for oblique styles to simulate italic.
PageID	The ID of the Page on which this text fragment is located.
StreamID	The Stream ID of the content stream in which this text fragment is located.
StreamOffset	The offset within the content stream to the start of the drawing operation that contains this fragment.
StreamLength	The length within the content stream of the drawing operation that contains this fragment.
TextSpanIndex	The zero based index of the drawing operator text array item that contains this fragment.

	For non text array operators this value is zero.
Rect	The rectangle that contains the text of the fragment.
Rotation	The angle of rotation of the fragment in degrees.
Text	The text of the fragment. A fragment may span only a portion of the complete text drawing operator.
RawText	The text specified in the drawing operator or the text array item of the drawing operator.
FontColorSpace	The ColorSpace used for drawing this text fragment.

# TextOperation Class

---



Operation to analyse and manipulate the text in a set of pages.

System.Object

**WebSupergoo.ABCpdf10.Operations.TextOperatio**

Method	Description
• <a href="#">TextOperation</a>	TextOperation Constructor.
• <a href="#">GetText</a>	Get all the text in the page contents.
• <a href="#">Select</a>	Select a range of text in the document.
<a href="#">Group</a>	Group a range of text fragments into a set of lines.

Property	Description
<a href="#">Hyphenation</a>	Whether to de-hyphenate words that appear to be split across two lines.
	Whether to provide native colors such as

NativeColors	CMYK, separations and spot colors, or whether to convert all colors to RGB.
PageContents	The pages to be operated upon.
ShowArtifactText	Whether to show text content that is marked as an artifact.
ShowClippedText	Whether to show text which is invisible because it is affected by a clip path.
ShowObscuredText	Whether to show overlapping repeated text content.
Substitutions	A set of character to string substitutions used for translating typographic characters like ligatures into more normal text.
TabAffinity	The minimum distance at which two text fragments will be assumed to be part of separate tab groups.

TabChar	The character used to separate out tab groups when GetText is called.
TextObjects	Whether to get information on the BT and ET text object markers used to contain and group text operators.
WordAffinity	The minimum distance at which two fragments will be assumed to be part of one undivided word.

# StringAtom Class

---

An Atom representing a text value.



```
System.Object
  WebSupergoo.ABCpdf10.Atoms.Atom
    WebSupergoo.ABCpdf10.Atoms.StringAtom
```

Method	Description
<b>S»</b> Encode	Encode a string into a format for use in a content stream using a normal or simple font.
<b>S»</b> EncodeDoubleByte	Encode a string into a format for use in a content stream using a composite font with a double byte CMap.
StringAtom	Construct a StringAtom.
Decode	Decode a PDF encoded string

	into a plain string format.
DecodeDoubleByte	Decode a double byte PDF encoded string into a plain string format.
Equals	Test whether the two StringAtoms are the same.
	inherited methods...

Property	Description
Text	The text of the string.
	inherited properties...

# BoolAtom Class

---

An Atom containing a Boolean value.



```
System.Object  
  WebSupergoo.ABCpdf10.Atoms.Atom  
    WebSupergoo.ABCpdf10.Atoms.BoolAtom
```

Method	Description
<a href="#">BoolAtom</a>	Construct a BoolAtom.
<a href="#">Equals</a>	Test whether the two BoolAtoms are the same.
	<a href="#">inherited methods...</a>

Property	Description
<a href="#">Truth</a>	Whether the Boolean is true or false.
	<a href="#">inherited properties...</a>

# ProcessingObjectEventArgs Class

---

Provides data for the [ProcessingObject](#) event.

This class inherits from the [CancelEventArgs](#) class to allow processing to be cancelled.



```
System.Object
  System.EventArgs
    System.ComponentModel.CancelEventArgs
      WebSupergoo.ABCpdf10.Operations.ProcessingObjectEventArgs
```

Method	Description
<a href="#">ProcessingObjectEventArgs</a>	ProcessingObjectEventArgs Constructor.

Property	Description
<a href="#">Object</a>	Gets the IndirectObject to be processed.
<a href="#">Tag</a>	Gets or sets an object which can be used to store data about the event.
<a href="#">Info</a>	Gets the ProcessingInfo containing relevant information.

# ProcessedObjectEventArgs Class

---

Provides data for the [ProcessedObject](#) event.



```
System.Object
  System.EventArgs
    WebSupergoo.ABCpdf10.Operations.Processed
```

Method	Description
<a href="#">ProcessedObjectEventArgs</a>	ProcessedObjectEventArgs Constructor.

Property	Description
<a href="#">Object</a>	Gets the IndirectObject which has just been processed.
<a href="#">Successful</a>	Gets a value indicating whether the processing was successful.
<a href="#">Tag</a>	Gets or sets an object which can be used to save data about the event.

# ProcessingInfo Class

---



Provides generic information relating to the [Operation.ProcessingObject](#) event. The information about the source of the operation is available here when [ProcessingEventArgs.Object](#) is null if the source is in a non-PDF format.

```
System.Object  
    WebSupergoo.ABCpdf10.Operations.ProcessingIn
```

Method	Description
<a href="#">ProcessingInfo</a>	ProcessingInfo Constructor.

Property	Description
<a href="#">SourceType</a>	Gets the type of the source object to be processed and the stage of operation.
<a href="#">Handled</a>	Gets or sets a value that indicates whether the event handler has handled the event so that the operation skips the default processing.

X	Gets the x-coordinate of the location of the source object.
Y	Gets the y-coordinate of the location of the source object.
Width	Gets the width of the source object.
Height	Gets the height of the source object.
DocNumber	Gets or sets the document number of the source document being/to be processed.
DocCount	Gets the number of documents in the source object.
PageNumber	Gets or sets the page number of the source page being/to be processed.
PageCount	Gets the number of pages in the source object.
FrameNumber	Gets or sets the frame number of the source frame being/to be processed.

FrameCount	Gets the number of frames in the source object.
FrameRate	Gets the number of frames per second for the source object.
BackgroundColor	Gets the background color of the source object.
StreamPosition	Gets or Sets the stream position of the source object, if applicable.
StreamLength	Gets the stream length of the source object, if applicable.
Text	Gets the unicode Text of the Source Event, if applicable.

# RecolorOperation Class

---

An operation used to recolor PDF documents.



System.Object

WebSupergoo.ABCpdf10.Operations.Operation

**WebSupergoo.ABCpdf10.Operations.RecolorOp**

Method	Description
<a href="#">RecolorOperation</a>	RecolorOperation Constructor
• <a href="#">Recolor</a>	Recolor pages in a document
	<a href="#">inherited methods...</a>

Property	Description
<a href="#">ConvertAnnotations</a>	Gets or sets a value indicating whether annotations are to be recolored.
<a href="#">DestinationColorSpace</a>	Gets or sets the destination ColorSpace.
	<a href="#">inherited properties...</a>

# SwfParameters Class

---



Parameters to initialize the SWF machine.

System.Object

**WebSupergoo.ABCpdf10.Operations.SwfParameter**

Property	Description
FlashVars	The Flash variables.
StageAlign	The stage alignment.
StageHeight	The stage height when the scale mode is NoScale.
StageScaleMode	The stage scale mode.
StageWidth	The stage width when the scale mode is NoScale.

# PageContents Class

---



Operation to analyse and manipulate the text in a set of pages.

System.Object

**WebSupergoo.ABCpdf10.Operations.PageContents**

Method	Description
• PageContents	PageContents Constructor.
AddLayers	Add a particular set of Layer objects from a particular Page.
• AddPages	Add pages to be processed.

Property	Description
• IncludeColor	Whether to include color information in the output.
• IncludeAnnotations	Whether to include annotation and field text in the output.
	Whether to attempt

- RegenerateUnicode

to regenerate missing Unicode tables from embedded fonts.

# TextGroup Class

---

A group of text fragments that belong together.

The essence of a TextFragment is that it spans only a small section of text. As such it may be necessary to group fragments together. For example take the following PDF drawing instruction sequence:

```
[ (There) 400 (was) 400 (Eru,) 300 (the One)
] TJ
```

This sequence will write out the text "There was Eru, the One". There are no space characters here and the spacing is indicated by character placement (the numbers) instead.



**Selecting** the phrase "was Eru" will result in two TextFragments the first of which will correspond to "was" and the second to "Eru". The two `TextFragment.Rect` properties are not connected because there is a 400 wide gap between them. Also there is no space character so simply concatenating the text in the fragments would result in "wasEru".

Using the `TextOperation.Group` method will group them into one TextGroup with the `Text` "was Eru" (space inserted) and a `Rect` corresponding to the complete selected phrase.

```
System.Object
  WebSupergoo.ABCpdf10.Operations.TextGroup
```

Method	Description
none	

<b>Property</b>	<b>Description</b>
PageID	The ID of the Page on which this group is located.
Rect	The rectangle that contains the group.
Text	The text of the group.
TextFragments	The text fragments in the group.

# ImageOperation Class

---



Operation to analyze the placement of raster images (bitmaps) in document.

System.Object

**WebSupergoo.ABCpdf10.Operations.ImageOperati**

Method	Description
<a href="#">ImageOperation</a>	ImageOperation Constructor.
<ul style="list-style-type: none"><li>• <a href="#">GetImageProperties</a></li></ul>	Get the image information for all the raster images.

Property	Description
<a href="#">IncludeAll</a>	Whether to include all images in the analysis to allow the detection of orphans.
<a href="#">PageContents</a>	The pages to be operated upon.

# ImageProperties Class

---



A class that represents one raster image (bitmap) in a document.

System.Object

**WebSupergoo.ABCpdf10.Operations.ImagePropert**

Method	Description
none	

Property	Description
PixMap	The PixMap object associated with the image.
Renditions	The set of renditions of the image at different locations within the document.
Dpi	The resolution of the image in dots per inch.
DpiX	The horizontal resolution of the image in dots per inch.
DpiY	The vertical resolution of the image in dots per inch.



# ImageRendition Class

---



A class that represents one image placement in a document.

System.Object

**WebSupergoo.ABCpdf10.Operations.ImageRenditi**

Method	Description
Focus	Focus the document on the location of the image placement.

Property	Description
PageID	The ID of the Page on which the image is placed.
Matrix	The transformation matrix for the placement of the image.
BoundingBox	The bounding rectangle for the placement of the image.
Dpi	The resolution of the image in dots per inch.
DpiX	The horizontal resolution of the image in dots per inch.
DpiY	The vertical resolution of the image in dots per inch.

<b>StreamObject</b>	The StreamObject in which the image draw operation is contained.
<b>StreamOffset</b>	The offset within the uncompressed StreamObject to the start of the drawing operation that contains this image draw operation.
<b>StreamLength</b>	The length within the uncompressed StreamObject of the drawing operation that contains this image draw operation.
<b>StreamID</b>	The ID of the Stream in which the image draw operation is contained.

# FlattenTransparencyOperation Class

---

An operation used to flatten the transparency in PDF documents.



```
System.Object  
  WebSupergoo.ABCpdf10.Operations.Operation  
    WebSupergoo.ABCpdf10.Operations.FlattenTr
```

Method	Description
<a href="#">FlattenTransparencyOperation</a>	FlattenTransparencyOperation Constructor.
● <a href="#">FlattenTransparency</a>	Flatten the transparency on the pages in a document.
	<a href="#">inherited methods</a>

Property	Description
<a href="#">Alpha</a>	Sets the backdrop alpha.
<a href="#">AntiAliasPolygons</a>	Whether to anti-alias polygons by creating synthetic image objects.
<a href="#">AntiAliasText</a>	Whether to anti-alias text with synthetic image objects.
<a href="#">ColorSpace</a>	Sets the target and composition colorspace for objects being flattened.
	Gets or sets a value indicating whether to flatten the transparency on the pages in a document.

ConvertAnnotations	annotations are to be flattened
DotsPerInch	Sets the resolution of generated PDF
IccCmyk	The path to the default CMYK profile.
IccGray	The path to the default Gray profile.
IccRgb	The path to the default RGB profile.
Log	Returns information from the FlattenTransparency operation
	inherited properties...

# AccessibilityOperation Class

---

Operation to make a document accessible.

Accessible or Tagged PDFs are the same as normal PDFs but have been annotated with metadata in the form of PDF tags. This metadata is required because PDF documents contain good layout information but little semantic structure. The tags that are required to supply this semantic structure are inserted and their operation is defined in the Adobe PDF Specification.



See the [MakeAccessible](#) function for further information on accessibility standards and tagged PDF.

System.Object

**WebSupergoo.ABCpdf10.Operations.AccessibilityOperation**

Method	Description
• <a href="#">AccessibilityOperation</a>	AccessibilityOperation Constructor.
• <a href="#">MakeAccessible</a>	Tags the document for accessibility.

Property	Description
<a href="#">PageContents</a>	The pages to be operated upon.
<a href="#">FixFonts</a>	Whether to attempt to fix font sets that may be required for accessibility.

## FixMetadata

Whether to attempt to fix or add metadata that may be required for accessibility

# EffectOperation Class

---

Operation to apply a visual effect.



```
System.Object  
  WebSupergoo.ABCpdf10.Operations.Operation  
    WebSupergoo.ABCpdf10.Operations.EffectOpe
```

Method	Description
<a href="#">EffectOperation</a>	EffectOperation Constructor.
• <a href="#">Apply</a>	Apply the effect to an image.
	<a href="#">inherited methods...</a>

Property	Description
<b>S»</b> <a href="#">Names</a>	Names of all the installed effects.
<a href="#">Name</a>	The name of the Effect.
<a href="#">Description</a>	A description of what the effect does.
<a href="#">Parameters</a>	The parameters associated with the effect.
<a href="#">AutoQuality</a>	The quality of compression to use when automatically compressing after the effect is applied.

AutoRestore

Whether to automatically restore the image color space and apply compression after the effect is applied.

# EffectParameter Class

---



A named parameter to be applied to an effect.

```
System.Object  
WebSupergoo.ABCpdf10.Operations.EffectParame
```

Method	Description
	None

Property	Description
Name	The name of the parameter.
Description	A description of what the parameter does.
Maximum	The maximum recommended value (if applicable).
Minimum	The minimum recommended value (if applicable).
Value	The value of the parameter, or the first value if there are multiple.
Values	The values of the parameter.

# IdleTimeout Property

---



Type	Default Value	Read Only	Description
[C#] long? [Visual Basic] Nullable(Of Long)	600,000 (10 minutes)	No	The maximum time a process can be idle before being terminated (ms).

Set this property to null to specify infinity.

---

## Notes

After a process has been idle continuously for the specified time, the process shall be terminated.

---

## Example

None.

# RetryCount Property

---



Type	Default Value	Read Only	Description
[C#] int [Visual Basic] Integer	3	No	The number of times an operation should be retried if there is a problem with the process.

The problems concerned are not related to a page's content.

---

## Notes

The process pool manages the communication with the processes. If the communication is problematic or a process becomes unstable, the process is terminated and a retry may take place.

---

## Example

None.

# PdfConformityOperation Class

---

Operation to produce conforming PDF files.

Note that this feature is only available under the ABCpdf Professional



```
System.Object
  WebSupergoo.ABCpdf10.Operations.PdfConformityOperation
  Implements: IDisposable
```

Method	Description
• <a href="#">GetData</a>	Get the conforming document as raw data.
• <a href="#">GetStream</a>	Get the conforming document as raw data stream.
• <a href="#">Save</a>	Write the conforming PDF document to a file.

Property	Description
<a href="#">Conformance</a>	The PDF conformance.
<a href="#">Messages</a>	The messages for writing in the conforming PDF format.
<a href="#">Conformity</a>	The PDF conformity.
<a href="#">Errors</a>	The errors for writing in the conforming PDF format.



# PdfValidationOperation Class

---

Operation to produce conforming PDF files.

Note that this feature is only available under the ABCpdf Professional Edition.



```
System.Object  
    WebSupergoo.ABCpdf10.Operations.PdfValidationOperation
```

```
Implements: IDisposable
```

Method	Description
--------	-------------

• Read	Read and validate an existing document.
--------	---

Property	Description
Conformance	The PDF conformance.
Doc	The Doc object into which the document is read.
Errors	The validation errors.
Warnings	The validation warnings.

# RenderOperation Class

---

An operation used to render pages of PDF documents.



```
System.Object
  WebSupergoo.ABCpdf10.Operations.Operation
    WebSupergoo.ABCpdf10.Operations.RenderOpe
```

Method	Description
<a href="#">RenderOperation</a>	RenderOperation Constructor
• <a href="#">Save</a>	Renders a page into a file.
• <a href="#">GetBitmap</a>	Renders a page into a bitmap
• <a href="#">GetData</a>	Renders a page into an array of bytes.

# ImageLayer Class

---

A generic bitmap layer appearing on a page of the document.

The actual bitmap data is held separately in a PixMap object. The ImageLayer merely references the bitmap and describes where and how it should be drawn. This is analogous to the way that images are embedded in HTML.



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
  WebSupergoo.ABCpdf10.Objects.StreamObject
  WebSupergoo.ABCpdf10.Objects.Layer
  WebSupergoo.ABCpdf10.Objects.ImageL
```

Method	Description
	inherited methods...

Property	Description
ImageName	The name used to identify the PixMap in the Page resources.
PixMap	The PixMap containing the image data.
	inherited properties...



# ViewLayer Class

---

A view onto a larger underlying graphic a portion of which is appearing on a page of the document.



```
System.Object
  WebSupergoo.ABCpdf10.Objects.IndirectObject
  WebSupergoo.ABCpdf10.Objects.StreamObject
  WebSupergoo.ABCpdf10.Objects.Layer
  WebSupergoo.ABCpdf10.Objects.ViewLa
```

Method	Description
	inherited methods...

Property	Description
ContentHeight	The content height of the image in pixels.
ContentWidth	The content width of the image in pixels.
PageHeight	The height of the content on the current page in pixels.
PageOffset	The offset to the top of the current page in pixels.
PageWidth	The width of the content on the current page in pixels.

ScrollHeight	The scroll height of the image in pixels.
ScrollWidth	The scroll width of the image in pixels.
Truncated	Whether the image is truncated.
	inherited properties...