PDFTron PDF2SVG SDK™
User Manual

Version 3.5

LEGAL STATEMENT AND COPYRIGHT NOTICE

PDFTron PDF2SVG SDK™ User Manual
Part number: PDFTRON-3.5-PDF2SVGSDK
January 15, 2007

TABLE OF CONTENTS

1. Introduction

1.1 An Introduction to PDFTron PDF2SVG SDK

PDFTron's PDF2SVG SDK (Software Development Kit) is a software component that can be used as a building block for other client and server-based applications to convert and publish PDF documents as SVG (Scalable Vector Graphics), the open-standard W3C recommendation for high-end graphics on the web.

PDF2SVG enables high-quality conversion from PDF to SVG that maintains the original layout of the document and preserves hyperlinks, colors and fonts. The resulting self-contained and compact SVG files can be distributed, viewed, edited, stored, printed, and published onto web sites and CD-ROMs. For quick and easy document navigation and viewing, PDF2SVG can be configured to create page thumbnails as well as an XML summary describing the document components such as metadata, bookmarks, annotations, etc. Using XSLT or any other XML processor, the user can quickly generate custom HTML and JavaScript 'wrappers' that enable the user to browse multi-page documents in any web browser.

Like other PDFTron products, PDF2SVG SDK does not rely on any other third-party software for its operation.

The SDK is available as a plain 'C DLL' and can be easily accessed from any programming language (including C#, VB.NET, C/C++, Java, VB6, Perl, Python, Ruby, Delphi, etc).

Also available as a command-line application, PDF2SVG is the best way to publish high quality, XML-based graphical content. Please see http://www.pdftron.com/pdf2svg for more information.

1.1.1 Key Functions

- High quality conversion from PDF (Portable Document Format) to SVG (Scalable Vector Graphics) that maintains the original document layout and preserves hyperlinks, colors and fonts.
  - Font support: Type1, TrueType, Type3 and Type0/CID Fonts
  - Color: ICC, DeviceN, Separation, RGB, CMYK, Indexed, etc.
  - Support for encrypted PDF documents (40 and 128 bit RC4, 128 bit AES).
  - Support for all kinds of patterns, functions, and compression schemes.
  - Support for all annotation types.
- Compression and SVG optimization: Converted documents can be saved as SVGZ compressed files for fast web downloads.
- PDF2SVG allows for generation of stand-alone SVG files (i.e. SVG files with embedded resources) as well as for SVG files with shared resources.
- All text is converted to Unicode dramatically simplifying text manipulation, editing, and searching. PDF2SVG makes every attempt to map text to a Unicode public area, resulting in better repurposing and text-searching capabilities of converted documents.
- Font embedding and font substitution option: The font embedding option allows for reliable and accurate font reproduction. The font substitution option produces smaller files but may not produce 100% accurate fonts on all systems.
- Thumbnail generation.
- Option to extract document-level information that has no counterpart in SVG (e.g. metadata, bookmarks, annotations, etc).
- Batch conversion: Using PDF2SVG you can easily convert single files or whole PDF repositories.
- Efficiency: PDF2SVG SDK is based on PDFNet SDK, making it extremely fast and efficient. The conversion speed is suitable for interactive and dynamic applications.

1.1.2 What's New in Version 3.5?

- Generation of XML summary document describing important document components such as document metadata, bookmarks, annotations, etc. Using XSLT or any other XML processor, XML summary can be used to wrap separate SVG documents into a composite document for e-book style navigation and viewing or for further repurposing.
- Thumbnail generation option for fast navigation through multi-page documents.
- Improved support for PDF 1.7 and Acrobat 8 documents.
- Support for soft, explicit, and color-key masks.
- Better conversion of PDF forms and annotations.
- The conversion speed is significantly improved.
- Smaller file size of generated SVG output.
- SVG output is now faster to render in popular SVG viewers.
- Support for 128-bit AES (Advanced Encryption Standard) encryption and Crypt filters.
- Files with broken cross reference tables are now automatically repaired.
- Support for clipping user defined regions.

1.1.3 Why SVG?

There are many benefits of converting your documents to SVG:

- SVG is a W3C (Web Standards Consortium) standard format and is backed by a large number of companies and non-profit organizations.
- Free SVG viewers are widely available on major platforms and operating systems.
- Because SVG is based on XML, the document can be easily edited in a text editor. SVG XML content can be linked to back-end business processes such as databases, application servers, and other rich sources of real-time information.
- There are a growing number of affordable and powerful SVG authoring and editing solutions.
- Enhanced search capabilities. All text in SVG is stored in standard XML syntax and Unicode encoding that makes searching operations within a document or across large collections of documents a breeze.

1.1.4 Common Use Case Scenarios

- Server-based, on-demand encryption and decryption of PDF documents based on specific security requirements.
- Batch processing of PDF collections that require a uniform security and permission settings across many documents. PDF2SVG is particularly useful in assembling product catalogues, brochures, and forms.
- PDF content extraction and repurposing through SVG and XML.

1.1.5 Operating Systems Supported

- Windows Vista, 2003, 2000, XP, NT, 98; Mac OSX; Linux

1.1.6 System Requirements

- At least 10 MB of free disk space.
- Memory requirement is heavily dependent on the nature of the document being converted into an image file.

1.2 About This Manual

This manual is intended as a guide to the installation and use of PDF2SVG SDK.

- Section 1 introduces PDF2SVG SDK and describes the manual.
- Section 2 explains how to install, register and uninstall PDF2SVG SDK.
- Section 3 covers basic use of the PDF2SVG SDK and integration with third-party applications.
- Section 4 contains the PDF2SVG Command String Syntax, FAQ and Usage Examples.
- Section 5 contains the XML Summary Document
- Section 6 covers general PDF2SVG related questions.
- Section 7 is where you will find all the support information you may require, such as how to report a problem with the software.

2. Installing, Registering and Uninstalling PDF2SVG SDK

2.1 PDF2SVG Installation

PDF2SVG SDK is supplied as a download from a distributor or directly from www.pdftron.com. The release is packaged as a .zip file (pdf2svg_sdk.zip). Unzip the archive in the desired location and make sure to preserve the directory (folder) structure when extracting the archive. In order to register the software, copy the license file provided to you into the "pdf2svg_sdk" folder.

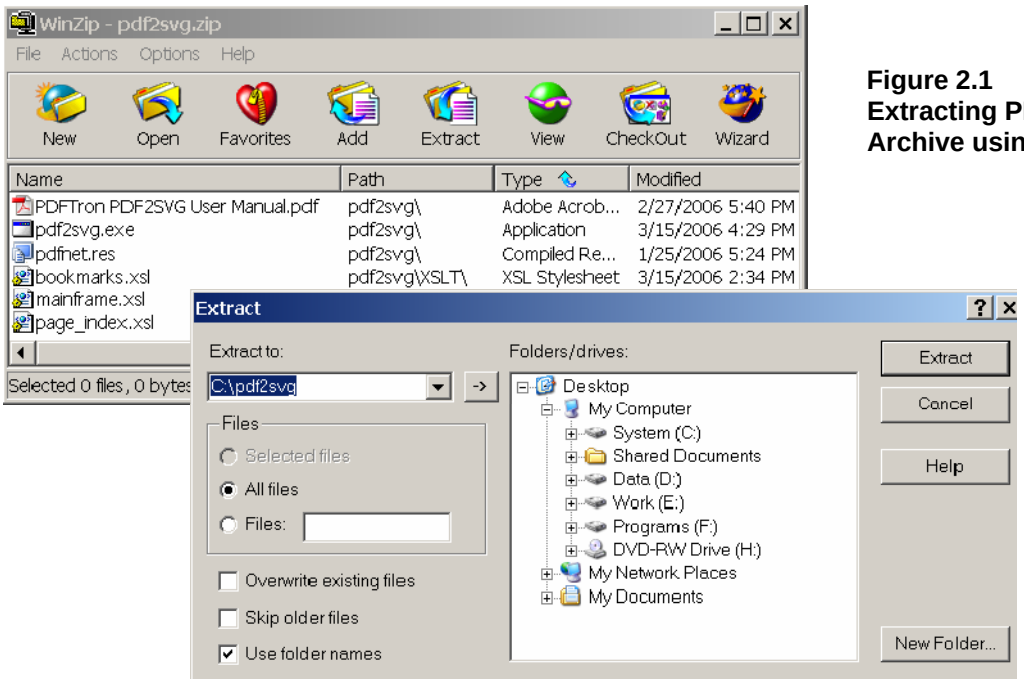


Figure 2.1
Extracting PDF2SVG Archive using WinZip

2.2 Product Registration

After purchasing a license of PDF2SVG SDK, you will receive additional registration and license information via email.

To successfully register PDF2SVG SDK and remove evaluation restrictions, you only need to pass the license information during the call to PDF2SVGInit(). For example:

PDFSVGInit("Joe Doe", "MyCompany", "sRLL4qr555sd63dnd");

The first parameter is your user name, the second parameter is your company name, and the last parameter is the license key supplied with your registration information.

2.3 Demo Version Installation

If you wish to evaluate the product, you can download the demo version of the product without any serial number or license key.

To do this, go to PDFTron's **Downloads** page at www.pdftron.com/downloads.html. Click on the appropriate product version/name, which will bring you to the product and the appropriate link for the demo download. Simply Download the zip file (pdf2svg_sdk.zip) and extract the archive in the desired location, while making sure to preserve the directory (folder) structure when extracting the archive. This will provide you with a working copy of the component. The limitation of the evaluation version is that all pages in processed documents will be missing random characters and that some images will have a demo stamp.

2.4 Uninstalling PDF2SVG

To remove PDF2SVG SDK from a computer, simply delete the "pdf2svg_sdk" folder.

3. Overview

PDFTron PDF2SVG SDK is a software component designed to convert PDF files to SVG, the open-standard W3C recommendation for high-end graphics on the web. The flawless conversion process creates web-ready SVG documents.

The SDK is available as a plain 'C DLL' and can be easily accessed from any programming language (including C#, VB.NET, C/C++, Java, VB6, Perl, Python, Ruby, Delphi, etc).

The entire API consists of only two functions and is very simple to use.

This section covers the basic use of PDFSDK SDK explaining all the available options.

3.1 Working with PDF2SVG API

The API consists of only two functions: PDF2SVGInit and PDF2SVGRun.

PDF2SVGInit is called only once per process session to initialize the library and register the component. PDF2SVGRun can be called many times to process PDF documents or folders with PDF documents.

The following is the simplest application that can be built using PDF2SVG SDK:

```
// Using C# or C/C++
void main() {
  PDF2SVGInit("username", "company", "lic_key");
  PDF2SVGRun("-o c:/out c:/test/tiger.pdf", MyCallback, 0);
}
```

This application essentially executes a hardcoded operation. This operation converts 'tiger.pdf' to 'tiger.svg'. To convert all PDF documents in 'test' folder simply delete 'tiger.pdf' from the command string.

The first parameter of the PDF2SVGRun() function is a command string which is exactly the same as the general syntax used for the PDF2SVG Command-Line application. For a detailed explanation of all options, please refer to section 4 of this manual. The PDF2SVG Command-Line application is a great tool to get to know all the available options. In fact, building a command-line application using PDF2SVG SDK is as simple as the following listing:

```
// Using C#
static void Main(string[] args) {
  PDF2SVGInit("username", "company", "lic_key");

  String s = "";
  foreach (string arg in args) {
    s += arg + " ";
  }
  PDF2SVGRun(s, 0, 0);
}
```

It is also possible to build the command string dynamically (e.g. based on the user or dynamic input), as illustrated in the following code snippet:

```
// Using C#
static void Main(string[] args)
{
	PDF2SVGInit(username, company, lic_key);
	PDF2SVGCallback mycallback = new PDF2SVGCallback(MyCallback);

	string output_folder = "../samples/TestOutput/test1";
	string open_password = "secret";
	bool compress_using_svgz = true;
	bool embedimages = true;
	bool generate_xml_masterdoc = false;
	bool generate_thumbnails = false;
	int thumbsize = 150;
	string filename_prefix = "";
	int filename_digits = 4;

	// Given the above settings build a command string.
	String s = "";

	if (output_folder != "") s += "-o " + output_folder + " ";
	if (open_password != "") s += "-p " + open_password + " ";
	if (compress_using_svgz) s += "--svgz ";
	if (embedimages) s += "--embedimages ";
	if (generate_xml_masterdoc == false) s += "--noxmldoc ";
	if (generate_thumbnails)
	{
		s += "--thumbsize " + thumbsize + " ";
	}
	else
	{
		s += "--nothumbs ";
	}

	if (filename_prefix != "") s += "--prefix " + filename_prefix + " ";
	if (filename_digits>0)  s += "--digits " + filename_digits + " ";

	// specify input PDF files and folders...
	s += "../samples/TestFiles/tiger.pdf ";
	s += "../samples/TestFiles/tai962.pdf ";
	// s += "c:/my_pdf_folder "; etc ...

	// Execute the command string.
	int error_code = PDF2SVGRun(s, mycallback, 0);
}
```

3.2 Reporting Progress Messages and Errors

To find out if PDF2SVG processing was successful, the application can query the status code returned by PDF2SVGRun().

For example,

```
int ret = PDF2SVGRun(...);
if (ret == PDF2SVG_OK) {
      // No errors...
}
else if (ret == PDF2SVG_ERR_BADKEY ) {
      // bad license key...
}
else if (ret == PDF2SVG_ERR_DIRCREATE ) {
      // Failed to create the output directory
}
else {
      // Unspecified error
}
```

A non-zero value indicates that an error was encountered. You can find the listing for all error code in 'include/PDF2SVG.h' header.

For more detailed error and message reporting, you can pass a pointer to the custom callback function in the second parameter of PDF2SVGRun(). The last parameter in PDF2SVGRun is a pointer to custom data that you may want to pass to the callback function.

A sample callback function may look as follows:

```
// Using C/C++
char* MyCallback(int mode, char* msg, void* user_data) {
  if (mode == PDF2SVG_ERROR) {
    cout << "Error: " << msg << endl;
  }
  else if (mode == PDF2SVG_MSG) {
    cout << msg;
  }
  else if (mode == PDF2SVG_GETPASS) {
    static string gl_pass;
    cin >> gl_pass;
    return (char*)gl_pass.c_str();
  }
  return 0;
}

' or in VB.NET...
Public Function MyCallback(ByVal mode As Integer, ByVal msg As String,
ByVal user_data As Int32) As Int32
  If mode = PDF2SVG_ERROR Then
   Console.WriteLine("Error: {0}", msg)
  ElseIf mode = PDF2SVG_MSG Then
   Console.Write("{0}", msg)
  End If
  Return 0
End Function    'MyCallback
```

4. PDF2SVG Command String Syntax

This Section includes the command string syntax, used both in the PDF2SVG Command-Line Applications as well as in PDF2SVG SDK.

4.1 Command String Syntax

The basic command string syntax is:

```
[options] file1 file2 folder1 file3 ...
```

4.2 Command String Summary

The following command string arguments are available for PDF2SVG.

Option	Parameter	Description
-o or --output	e.g. -o myfolder -o "C:\My Folder1\F2"	The output folder used to store converted files. By default, the currently selected working folder will be used to store converted SVG files.
--prefix	--prefix myprefix	The prefix for the output SVG file. The output filename will be constructed by appending the prefix string, the page number, and the appropriate extension (e.g. myprefix1.svg, myprefix2.svg, etc). The prefix option should be used only for conversion of individual documents. By default, the each input filename will be used as a prefix.
--digits	--digits 4	The number of digits used in the page counter portion of the output filename. By default, new digits are added as needed; however, this parameter could be used to format the page counter field to a uniform width (e.g. myfile0001.svg, myfile0002.svg, etc).
--subfolders		Process all sub-directory for every directory specified in the argument list. By default, sub-directories are not processed.
-a or --pages	Convert page 1,3, and 10: -a 1,3,10 Convert all even pages: -a even Convert pages in the range from 3-11 and page 50: --pages 3-11,50 Convert all odd pages and all pages in the range from 100 to the last page: -a odd,100-	Specifies the list of pages to convert. By default, all pages are converted.
--svgz	--svgz	Compress output SVG files using GZIP/SVGZ compression. The default extension for compressed SVG is 'svgz'. By default, generated SVG output is not compressed.
-i or --embedimages	-i	Embeds all images. Using this option it is possible to create self contained SVG files (i.e. files without any references to external resources). Although it is sometimes desirable to create self contained files, this option can result in slower rendering in some viewers. The files with embedded images may also be slower to download over the Net, and because images can't be shared among different pages the total file size for the entire document may increase.

		By default, all images are saved as external files.
--nofonts	--nofonts	Disables conversion of font data to SVG. This option will usually result in the smaller SVG file size, but due to font substitution the text may not render accurately. By default, all available fonts are converted to SVG.
--nounicode	--nounicode	Disables mapping of text to public Unicode region. Instead text will be converted using a custom encoding. By default, all text is mapped to Unicode.
-b or --box	-b media	Specifies the page box/region to use for clipping. Possible values are: ▪ **media** ▪ **crop** ▪ **trim** ▪ **bleed** ▪ **art** The default is page crop region.
-c or --crop	-c 216,522,330,600	User definable crop box to be used as a top level clip region in the output SVG. By default, the clip region is identical to currently selected page 'box'.
--noclip	--noclip	Disables page clipping. Any content outside of page boundaries will be visible. By default, all pages are clipped using the crop region for the page.
--noannots	--noannots	Disables conversion of form fields and annotations.
-- noxmldoc	--noxmldoc	Disables generation of the XML wrapper document.
--thumbsize	--thumbsize 150	The dimension of thumbnail image in pixels. By default, PDF2SVG will generate 100x100 thumbnails.
--nothumbs	--nothumbs	Disables generation of thumbnail images.
--asv_css_patch	--asv_css_patch	Some versions of the Adobe SVG viewer fail to read valid CSS font properties. As a result, this option was created as a workaround for this bug in the Adobe SVG Viewer. Please note that using this option may cause problems with other SVG viewers.
--noprompt		Disables any user input. By default, the application will ask for a valid password if the password is incorrect.
-p or --pass	e.g. secret or “my pass”	The password for the input file. Not required if the input document is not secured.
--extension	--extension “.pdf”	The default file extension used to process PDF documents. The default is “.pdf”.
-h or --help		Print a listing of available options.
-v or --version		Print the version information.
--verb	--verb 2	Set the verbosity level. Valid parameter values are 0, 1, and 2. The higher number results in more feedback. The default is 1.

4.3 PDF2SVG Command String FAQ

4.3.1 How to save converted files in a given folder?

By default, PDF2SVG saves converted files in the current working folder. To specify another output location, use the '-o' (or --output) parameter. For example:

```
–o “c:\My Output” 1.pdf 2.pdf 3.pdf
```

Note: If the specified path does not exist, PDF2SVG will attempt to create the necessary folders.

4.3.2 How can I control the output names of generated files?

By default, PDF2SVG creates a separate SVG file for every page in the document. The output filename is constructed using the name of the input PDF file, the page number, and appropriate file extension (i.e. svg or svgz). For example, the following command-line generates a sequence of SVG files starting with mydoc_1.svg, mydoc_2.svg, etc.:

```
mydoc.pdf
```

PDF2SVG allows output filename customizations using the '--prefix' and '--digits' options. For example, the following command-line generates a sequence of SVG files starting with newname_0001.svg, newname_0002.svg, etc.:

```
--prefix newname --digits 4 mydoc.pdf
```

The '--digits' parameter specifies the number of digits used in the page counter portion of the output filename. By default, new digits are added as needed, but the 'prefix' parameter could be used to format the page counter field to a uniform width (e.g. myfile0001.svg, myfile0010.svg, instead of myfile_1.svg, myfile_10.svg, etc).

To avoid any ambiguities in file naming, the prefix option should be used only for conversion of individual documents.

4.3.3 How do I create compressed SVG (SVGZ)?

To create compressed SVG (SVGZ), use '--svgz' as one of the command-line options. This option will instruct PDF2SVG for compress SVG using GZIP compression and to generate output files with the 'svgz' extension. For example,

```
--svgz in.pdf
```

4.3.4 How do I produce stand-alone SVG?

Some PDF documents use many small bitmaps to represent text or patterns. In this case, the converted SVG document will reference hundreds of external images. You may choose to embed these images within the SVG document using the '--embedimages' or ('-i') option.

By embedding images it is possible to create self contained SVG files (i.e. files without any references to external resources). Although it is sometimes desirable to create self contained files, this option can result in files that are slower to render in some viewers. The files with embedded

images may also be slower to download over the Net, and because images can't be shared among different pages the total file size for the entire document may increase.

4.3.5 How do I open a password protected PDF?

PDF2SVG will, without user intervention, decrypt and convert documents secured with a master/owner password. If the document is secured using a user (or file open) password, PDF2SVG will prompt you to enter the password.

For unattended conversion, the password can also be specified directly on the command-line using the '-p' (or --password) option. For example:

```
-p secret secured.pdf
```

The above command line will convert PDF to SVG and will use the provided password ('secret') to open the secured document (i.e. 'secured.pdf').

Note: PDF2SVG supports all standard security options available in PDF, including 40 and 128 bit RC4 encryption, Crypt filters, and 128 AES (Advanced Encryption Standard) encryption.

4.3.6 Why is PDF2SVG generating page thumbnails and the XML summary document?

By default, PDF2SVG generates a bitmap thumbnail for each converted SVG and one XML summary document for the entire document. Image thumbnails can be used for quick preview of SVG documents, whereas XML summary document could be used to create HTML files that wrap SVG files in a web ready eBook. XML summary document can also be used for content repurposing, navigation, and indexing.

PDF2SVG lets you control the dimensions of thumbnail images using '--thumbsize' parameter. The following command-line will generate 150x150 pixel image thumbnails for every page in the document:

```
--thumbsize 150 in.pdf
```

To disable generation of thumbnail images, use the '--nothumbs' option. Similarly, to disable generation of wrapper XML, use the '--noxmldoc' switch.

4.3.7 How do I specify which pages to convert?

By default, PDF2SVG will convert all pages. You can specify a subset of pages to convert using the '-a' or '--pages' options. For example:

```
-a 1,3,10 in.pdf
```

will convert only pages 1, 3, and 10. Please note that PDF2SVG assumes that all pages are numbered sequentially starting from page 1.

To specify a range of pages, use dash character between numbers. For example:

```
-a 1,10-20,50- in.pdf
```

will convert the first page, pages in the range from 10 to 20 and all pages starting with page 50 to the last page in the document.

All even pages can be selected using the ‘e’ (or ‘even’) string. For example, the following line converts all even pages:

```
--pages even in.pdf
```

Similarly odd pages can be selected using the ‘o’ (or ‘odd’) string. The following line converts all odd pages in the document and every page in the range from 100 to the last page:

```
--pages odd,100- in.pdf
```

4.3.8 How do I batch-convert files?

PDF2SVG supports batch conversion of many PDF files in a single pass. To convert all PDF files in a given folder(s) you can use the following syntax:

```
myfolder1
```

The ‘--subfolders’ option can be used to recursively process all subfolders. For example, the following line will convert all documents in ‘myfolder1’ and ‘myfolder2’ as well as all subfolders:

```
--subfolders myfolder1 myfolder2
```

By default, PDF2SVG will convert all files with the extension ‘.pdf’. To select different files based on the extension use the ‘--extension’ parameter. For example, to convert all PDF documents with a custom extension ‘.blob’, you could use the following line:

```
--extension .blob --subfolders myfolder1
```

4.3.9 How can I show/hide crop marks or the trim region?

A PDF page can define as many as five separate boundaries to control various aspects of the imaging process:

- The media box defines the boundaries of the physical medium on which the page is to be printed. It may include any extended area surrounding the finished page for bleed, printing marks, or other such purposes. It may also include areas close to the edges of the medium that cannot be marked because of physical limitations of the output device. Content falling outside this boundary can safely be discarded without affecting the meaning of the PDF file.
- The crop box defines the region to which the contents of the page are to be clipped (cropped) when displayed or printed. Unlike the other boxes, the crop box has no defined meaning in terms of physical page geometry or intended use; it merely imposes clipping on the page contents. The default value is the page’s media box.
- The bleed box defines the region to which the contents of the page should be clipped when output in a production environment. This may include any extra bleed area needed to accommodate the physical limitations of cutting, folding, and trimming equipment. The default value is the page’s crop box.
- The trim box defines the intended dimensions of the finished page after trimming. It may be smaller than the media box to allow for production related content, such as printing instructions, cut marks, or color bars. The default value is the page’s crop box.
- The art box defines the extent of the page’s meaningful content (including potential white space) as intended by the page’s creator. The default value is the page’s crop box.

By default, PDF2SVG uses the page crop box as a default clip region. Different page regions can be selected as the default clip region using the -b (or --box) parameter. For example, the following line will instruct PDF2SVG to use the media box for rasterization:

```
--box media in.pdf
```

4.4 Command String Examples

Example 1. The simplest command line: Convert PDF to SVG.

Notes:

- The '-o' (or --output) parameter is used to specify the output folder. If this option is not specified, all converted SVG-s will be stored in the current working folder.

```
-o outfolder in.pdf
```

Example 2. Convert PDF to compressed SVG and without thumbnails and XML summary.

Notes:

- The '--noxmldo' option disables generation of thumbnails.
- The '--nothumbs' option disables generation of thumbnails.
- The '--svgz' option instructs PDF2SVG to compress SVG using GZIP compression.
- The '--verb' option instructs PDF2SVG to output more feedback in the console window.

```
--output test_out/ex2 --svgz --nothumbs --noxmldoc --verb 3 in.pdf
```

Example 3. Convert a password protected file to SVG.

Notes:

- The '-p' (or --pass) parameter is used to specify the password (i.e. 'secret') required to open the encrypted document.
- The '--pages' option instructs PDF2SVG to convert only the first page.

```
-p secret -o ex3 --nothumbs --noxmldoc --pages 1 secret.pdf
```

Example 4. Convert all PDF document in a given folders to stand alone SVG.

Notes:

- The '--bbox' parameter instructs PDF2SVG to use media box for clipping instead of crop box, which is the default.
- The '--embedimages' option (or –i in the short form) instructs PDF2SVG to embed all images as inline resources. This option produces stand-alone SVG files (i.e. SVG files without external references).

```
-o OUT --embedimages --box media "My Folder1" "My Folder2"
```

5. XML Summary Document

This section describes the XML Summary Document that can be generated using PDF2SVG and its potential use in various applications.

By default, PDF2SVG generates an XML Summary Document for every PDF document. The XML Summary Document contains document-level information that is not part of SVG files describing individual pages. The information includes general information about the document (such as author, subject, title, keywords), as well as a listing of document parts and relationships such as pages, thumbnails, annotations, and bookmarks.

The following is a sample XML snippet generated by converting this user manual to SVG:

```
<?xml version="1.0"?>
<!-- Generator: PDFTron PDF2SVG Converter -->
<doc name="1" ext="svg">
 <info>
  <title>PDFTron PDF2SVG User Manual</title>
  <author>PDFTron Systems</author>
  <subject>PDFTron PDF2SVG User Manual</subject>
  <keywords></keywords>
  <creator>Acrobat PDFMaker 7.0.7 for Word</creator>
  <producer>Acrobat Distiller 7.0.5 (Windows)</producer>
 </info>
 <pages>
  <page id="1" href="1_1.svg">
   <thumb href="1_1_thumb.jpg"/>
  </page>
  <page id="2" href="1_2.svg">
   <thumb href="1_2_thumb.jpg"/>
  </page>
  ...
 </pages>
 <bookmarks>
   <bookmark title=" 2. Installing and Uninstalling PDF2SVG" open="true"
goto="7" href="1_7.svg">
    <bookmark title="2.1 PDF2SVG Installation" open="false" goto="7"
href="1_7.svg"/>
    <bookmark title="2.2 Demo Version Installation" open="false" goto="7"
href="1_7.svg"/>
    <bookmark title="2.3 Uninstalling PDF2SVG" open="false" goto="7"
href="1_7.svg"/>
   </bookmark>
   ...
 </bookmarks>
</doc>
```

Most of the elements and attributes are self-explanatory. The ‘info’ element lists document information properties, the ‘pages’ element lists all ‘page’ elements that are part of the high level ‘document’, and the ‘bookmarks’ element specifies the outline tree that can be used for quick navigation between pages.

The summary document can be used as a map of the abstract document that contains many SVG files representing document pages, as well as outline tree and annotations describing how different document parts are related.

In most cases, the summary document is further consumed by an XML consumer/processor (e.g. XML DOM/SAX Library or XSLT). For example, an application may read XML summary to create database records for archiving purposes. Another application may implement interactive navigation through SVG pages using the document outline.

Yet another example of the XML wrapper consumer is an eBook generator that converts the XML Summary Document to HTML. The generated HTML would wrap converted SVG files and would provide web-based eBook interface for navigation between different pages, including bookmark tree, thumbnail index, etc. The end result would look as follows:

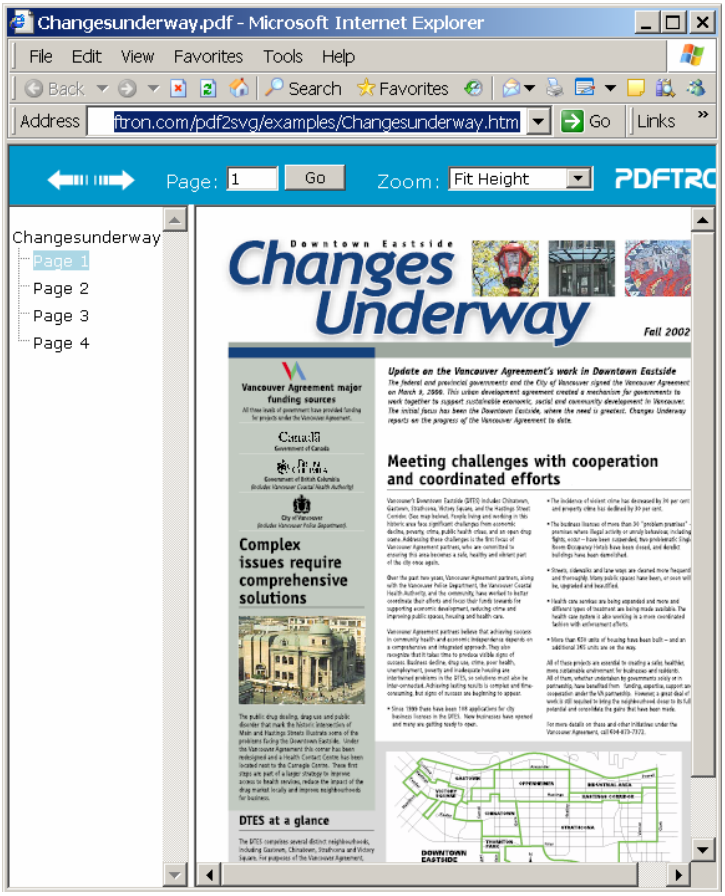
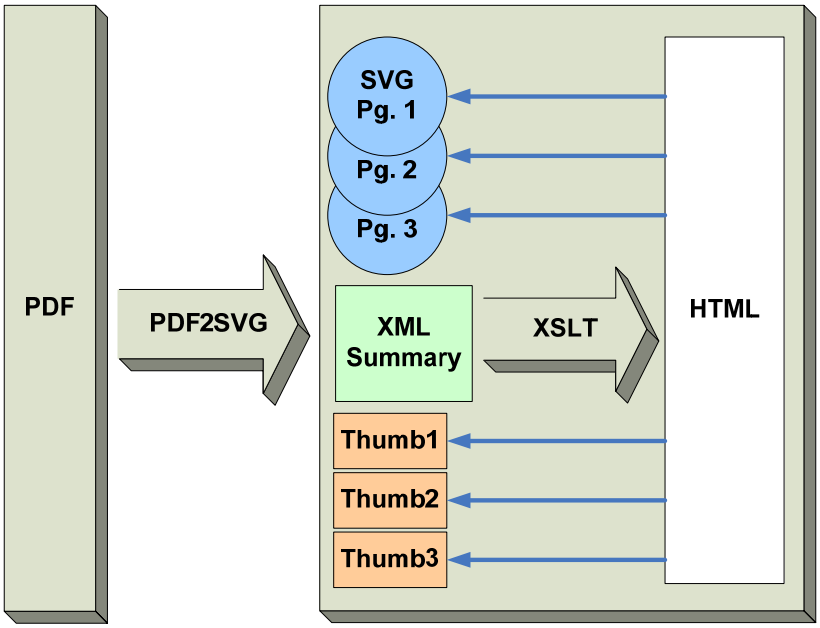


Figure: SVG wrapped in an HTML web-browser eBook.

The process used to create HTML eBook wrapping converted SVG-s is illustrated in the following figure:

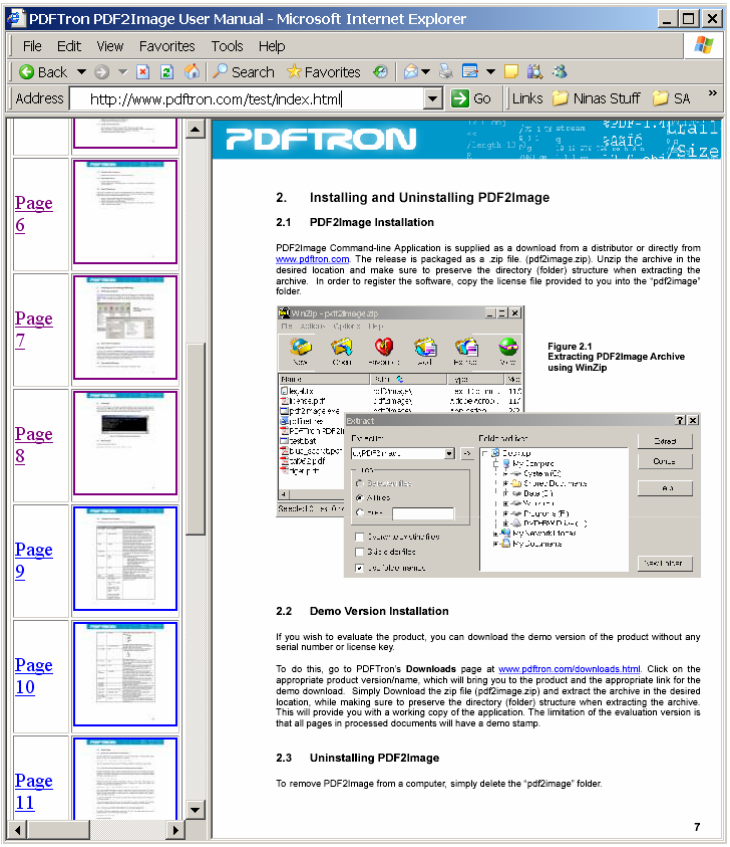


Using PDF2SVG, a PDF document is converted to a set of SVG images and their thumbnails, as well as the XML Summary Document. The fastest way to create HTML wrappers around SVG is using XSLT. XSLT is a very simple language for transforming XML documents. A simple XSLT transform may look as follows:

```
<?xml version='1.0'?>
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:output method='html' indent='yes' doctype-public='-//W3C//DTD HTML
3.2 Final//EN'/>
  <xsl:template match='/'>
    <HTML>
      <HEAD>
        <TITLE>HTML SVG Wrapper</TITLE>
      </HEAD>
      <BODY>
          <xsl:apply-templates select='doc/info'/>
          <HR/>
        <xsl:apply-templates select='doc/pages'/>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match='info'>
   <table border="0" cellspacing="0" cellpadding="4">
     <tr><td>Title:</td><td><xsl:value-of select='title'/></td></tr>
     <tr><td>Author:</td><td><xsl:value-of select='author'/></td></tr>
     <tr><td>Subject:</td><td><xsl:value-of select='subject'/></td></tr>
     <tr><td>Keywords:</td><td><xsl:value-of select='keywords'/></td></tr>
     <tr><td>Creator:</td><td><xsl:value-of select='creator'/></td></tr>
     <tr><td>Producer:</td><td><xsl:value-of select='producer'/></td></tr>
```

```
  </table>
  </xsl:template>

  <xsl:template match='pages'>
      <TABLE BORDER="1">
    <xsl:apply-templates/>
    </TABLE>
  </xsl:template>

  <xsl:template match='page'>
    <TR>
       <TD><A TARGET="view" HREF="{@href}">Page <xsl:value-of
select='@id'/></A></TD>
       <TD><A TARGET="view" HREF="{@href}"><IMG
SRC="{thumb/@href}"/></A></TD>
    </TR>
  </xsl:template>
</xsl:stylesheet>
```

The above XSLT template will create an HTML page containing general information about the documents such as it title, subject, keywords, etc. The HTML will also contain a thumbnail index of all pages in the document. Clicking on page labels or on thumbnails will open SVG graphics in the right pane of the browser window. The final result would look as follows:



To run XSLT transforms you can use your favorite XSLT processor. As a starting point, PDF2SVG distribution comes with a sample project illustrating how to run XSLT transform using Microsoft .NET Framework.

6. PDF2SVG General FAQ

6.1 What is SVG?

SVG (Scalable Vector Graphics) , developed by a working group of the World Wide Web Consortium (W3C), is an open-standard vector graphics format for describing two-dimensional graphics in XML (Extensible Markup Language).

SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects. SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting.

6.2 Why are some characters missing in SVG?

This is one of the limitations of the evaluation version of PDF2SVG. The full version of PDF2SVG will convert all text.

6.3 Why is SVG text using absolute positioning?

This is one of the limitations of the evaluation version of PDF2SVG. The full version of PDF2SVG does not use absolute positioning for each character unless font embedding is disabled.

6.4 How do I create an SVG eBook?

In PDF2SVG versions prior to version 3.0, HTML eBook wrapper generation was part of the conversion process.

With PDF2SVG V3.0 and higher, there is added flexibility because the conversion process generates an XML summary document that can be transformed using XSLT (or another XML processor) to a customized HTML, or to another file format. For a simple example of how to generate an HTML eBook wrapper, see the XSLT folder in your PDF2SVG directory.

6.5 Is it possible to customize the "look-and-feel" of the SVG eBook (HTML wrappers)?

Because PDF2SVG generates the XML document outline without any formatting elements, you have full control over the "look-and-feel" of the HTML wrapper. The transformation and formatting from XML to HTML can be specified using XSLT or another XML processor.

6.6 Why are some fonts not accurate?

In order to preserve fonts, you need to enable font embedding. Conversions with font embedding will produce slightly larger SVG documents, but the text should display correctly on different systems.

In some cases the original font may neither be available on the system nor in the PDF. In this case, PDF and SVG viewers need to perform font substitution. Because SVG viewers use different font

substitution procedures, the results may differ from one viewer to another. To avoid font substitution errors, make sure to create PDF documents with embedded fonts.

6.7 Why is a white space separating neighboring pictures?

In some cases, SVG viewers that support anti-aliased rendering produce line/space artifacts at neighboring picture elements (e.g. for image tiles or polygons sharing common edges). These artifacts are not a byproduct of PDF2SVG conversion, but are produced due to anti-aliased rendering in the SVG viewer. To eliminate anti-aliasing artifacts you can try to disable 'high-quality' rendering option in your SVG viewer.

6.8 Can I integrate PDF2SVG with my client/server application?

PDF2SVG SDK has a simple-to-use API that can be easily integrated into any third-party client and server-based applications. PDF2SVG SDK is available as a .NET component or as a cross-platform C library. For more information on licensing the PDF2SVG library, please contact a PDFTron representative at info@pdftron.com.

6.9 Does PDF2SVG SDK have any dependencies on third party components or software?

PDF2SVG SDK is a completely component and does not include any dependencies on any third-party components or software.

7. Support

7.1 Reporting Problems

If you encounter a problem or question regarding PDFTron PDF2SVG SDK that is not addressed on PDFTron's website, please submit a problem report to PDFTron's Support group at http://www.pdftron.com/reportproblem.html.

When submitting a problem you will be asked to provide the following information:

- Contact details
- Product and Version of the product
- Detailed description of problem
- Problem file(s)
- Whether you have an AMC (Annual Maintenance Contract) subscription
- Any other information that may be related

7.2 Contact Information

To contact PDFTron directly, you can use the contact information below:

PDFTron Systems, Inc.
#303 - 2575 West 4th Avenue
Vancouver, BC, V6K 1P5
Canada

Tel: 1-604-730-8989
Fax: 1-604-676-2477

Web site: www.pdftron.com.

Email Contacts:

General Business Inquiries: info@pdftron.com
Licensing, Sales Inquiries: sales@pdftron.com
Product Support: support@pdftron.com
Website related questions: webmaster@pdftron.com