CVISION TECHNOLOGIES

PdfCompressor 6.0

Developer's SDK

Copyright

PdfCompressor 6.0 Developer’s SDK Guide

Table of Contents

The PdfCompressor 6.0 Developer's SDK

Overview

CVISION's PdfCompressor **SDK** allows PdfCompressor users to incorporate PDF writing and OCR'ing functionality into their own custom applications. The **API** is based mainly upon two classes called **APICVistaPDFWriter** (see page 6) and **APICVistaPDFReader** (see page20). Nearly all API functions are implemented as public member functions of these two classes.

Interfaces for the PdfCompressor **API** are provided for both **C++** (the native language of the PdfCompressor engine) and .NET languages such as C# and VB.NET. Support for .NET languages is provided via an intermediary DLL wrapper.

64-Bit Support

Two variants of the **PdfCompressor SDK** are available - one for 32-bit (x86) application development and one for 64-bit (x64) application development. If you have installed the 32-bit version of PdfCompressor, you need to install the 32-bit version of the **PdfCompressor SDK** . If you have installed the 64-bit version of PdfCompressor, you need to install the 64-bit version of the **PdfCompressor SDK** . A 32-bit installation of the PdfCompressor cannot coexist with a 64-bit installation of the **PdfCompressor SDK** , and vice-versa. Also, you cannot install PdfCompressor for both x86 and x64 on the same machine.

If you need to develop for both 32-bit and 64-bit platforms, the easiest way to accomplish this is to install the 32-bit installations and 64-bit installations on two separate machines. After you have developed your application for one of those platforms, copy your project to the other machine and build your application for the other platform using the alternate binaries provided for that platform. However, if this presents a problem (such as if you only have a license for one machine), CVISION can send you just the binaries for the alternate platform. Please contact CVISION support at support@cvisiontech.com to assist you with this matter.

Compatibility Note

In order to ensure compatibility with your application, make sure you are running with **Microsoft Visual Studio 2008 SP1** or **2010 SP1**. You may encounter compilation or runtime errors with older versions. The PdfCompressor SDK may work with later versions of **Visual Studio** or with compilers from other vendors, but compatibility is not guaranteed.

Compiling and Linking in C++

The files needed for integrating **PdfCompressor API** functionality into your program are provided in the API subfolder of where you installed PdfCompressor.

Compiling your application with the APICVistaPDFWriter (or APICVistaPDFWriterMP) API

Five C++ header files are required for the **PdfCompressor API**. They are located in the VC\APICVistaPDFWriter\include folder:

```
APICommonDefines.h
APICVistaPDFWriterUser.h
APIerrors.h
APIWriterDefines.h
interfacedefs.h
```

A link library is provided as well, called `PdfEnc.lib` . Link with this file to resolve the symbols contained in `PdfEnc.dll` .

Compiling your application with the APICVistaPDFReader API

Five C++ header files are required for the **PdfCompressor API**. They are located in the VC\APICVistaPDFReader\include folder:

```
APICommonDefines.h
APICVistaPDFReaderUser.h
APIerrors.h
APIReaderDefines.h
interfacedefs.h
```

A link library is provided as well, called `PdfDec.lib` . Link with this file to resolve the symbols contained in `PdfDec.dll` .

Debugging Tip:

To correctly run applications written using the **PdfCompressor API** requires prior installation of PdfCompressor. Many of the binaries installed by PdfCompressor will be used by the API and therefore should be added to the same path as your project's output executable. See *Distributing Applications Based on the PdfCompressor API* on page 3 for details regarding which DLLs to copy into your project.

The .NET Wrapper for the PdfCompressor API

As mentioned above, in addition to native C++ API support, the **PdfCompressor API** also offers a .NET wrapper for the API methods provided. The wrapper provides the intermediary interfaces required to enable a program written in a .NET language (such as C# or VB.NET) to communicate with the API, in a file called `PdfEncNET.dll.`

Use of the .NET wrapper is fairly straightforward. To make use of the APICVistaPDFWriter class, simply add a reference to the `PdfEncNET.dll` in your .NET development project. The interfaces are essentially the same as those provided by the C++ file `APICVistaPdfWriterUser.h`. The namespace is called **CVision.PdfCompressor**.

Several versions of `PdfEncNET.dll` are provided in the API installation, corresponding to various versions of the .NET Framework (2.0 through 4.0). They are all located in the DOTNET folder of the API installation.

Distributing Applications Based on the PdfCompressor API

In order to distribute applications linked with the **PdfCompressor API**, you will need to package some of the files provided with the SDK installation and/or the main **PdfCompressor** installation. These files are listed below. Please note that wherever the word CVISTAPATH is shown, it means the location where **PdfCompressor** was installed on your development system (typically %PROGRAMFILES%\CVision\PdfCompressor 6.0).

Note: ☞

DO NOT DISTRIBUTE ANY SOURCE OR LIBRARY STUB FILES (*.h, *.cpp, *.lib, *.cs, etc.) PROVIDED BY THE PDFCOMPRESSOR SDK . THESE FILES ARE FOR COMPILATION AND LINKING PURPOSES ONLY. DISTRIBUTION OF THESE FILES CONSTITUTES A VIOLATION OF YOUR LICENSE AGREEMENT WITH CVISION TECHNOLOGIES.

***Files required for integrating the APICVistaPDFWriter (or APICVistaPDFWriterMP) API**

Path	Files Needed	Comment
CVISTAPATH	PdfEnc.dll JJpxWriter.dll cximagecrt.dll	Main compression engine files. Make sure that they will be installed somewhere on the user's system where the operating system will find them (preferably your main program folder).
CVISTAPATH	libeay32.dll	Needed for the 64-bit version only.
CVISTAPATH\OCR	(all)	Add these files if your application will create searchable PDFs. (Availability of this feature depends on your license agreement.) Also see below regarding our OCR Language Pack.*
CVISTAPATH\API\ DOTNET	PdfEncNET.dll	Add this file if your application is developed in .NET . It should reside in the same path as where you install PdfEnc.dll. Note that there are several subfolders here. Include the version of PdfEncNET.dll that corresponds to your target .NET framework.

- If you are distributing an application that requires OCR support for languages other than English, you need to download and install the CVISION OCR Language Pack. You can obtain it from the downloads page on our website at http://www.cvisiontech.com/download_main.html.

- If you are packaging an application for distribution, also make sure your deployment package includes the appropriate versions of any Visual C++ or .NET redistributables that are necessary to support your application.

Files required for integrating the APICVistaPDFReader API

Path	Files Needed	Comment
CVISTAPATH	PdfDec.dll JJpxWriter.dll cximagecrt.dll dtlstats.dll	Main decompression engine files. Make sure that they will be installed somewhere on the user's system where the operating system will find them (preferably your main program folder).
CVISTAPATH\API\ DOTNET	PdfDecNET.dll	Add this file if your application is developed in .NET . It should reside in the same path as where you install PdfDec.dll. Note that there are several subfolders here. Include the version of PdfDecNET.dll that corresponds to your target .NET framework.

Integrating a license mechanism into your application

The files `PdfEnc.dll` and `PdfDec.dll` have security mechanisms built into it that check for the existence of an electronic license before the software will run. While you are developing your application initially, you may use the versions of `PdfEnc.dll` and `PdfDec.dll` included with the version of PdfCompressor upon which you installed the API .

However, before you can package your software and begin distributing it, depending on your licensing terms, you may need to swap in a special version of these files as well as some other files, which are either protected by a special alternative license mechanism provided by CVISION or designed to interface with a license mechanism provided in your own application. When you are nearing the point at which you will be packaging your application, please contact CVISION to work out these details.

The APICVistaPDFWriter Class

Background Information

This section explains the conventions and terminology used later on in the documentation.

DIB - An acronym for **Device-Independent Bitmap**, this is the raw Windows bitmap format used to exchange image data within the **PdfCompressor API**. The format of a DIB file is equivalent to that of a BMP file without the BITMAPFILEHEADER section. Additional information on the DIB format can be found at the Microsoft Developer Network (MSDN) web site.

The documentation refers to a **byte** data type. **byte** is a C++ `typedef`, defined as:

```
typedef unsigned char byte;
```

All functions described below which have a **bool** return type representing a success code return **true** upon success and **false** upon failure.

Class and Member Definitions

This section documents the APICVistaPDFWriter class and its member functions.

Data Structure	
C++ Data Type	***Description***
class APICVistaPDFWriter	The structure encapsulating the API functions for PDF file creation

Function	
Function Signature(s)	***Description***
APICVistaPDFWriter(const APICVistaPDFWriter&);	A function declaration which prevents copy-construction

Function	
Function Signature(s)	***Description***
void operator=(const APICVistaPDFWriter&)	A function declaration that prevents assignment

Function	
Function Signature(s)	***Description***
APICVistaPDFWriter()	A default constructor

Function	
Function Signature(s)	***Description***
~APICVistaPDFWriter()	The destructor

Function		
Function Signature(s)		***Description***
bool HasValidLicense(__int64 flagsToCheck)		Check whether a valid PDF writer license exists.
Input		
Param #	**Param Name**	**Description**
1	flagsToCheck	Flags to check for specific rights. Pass 0 to this parameter unless otherwise instructed.
Returns		
C++ Data Type		**Description**
bool		Success status; a value of true means that the license is valid, while false means that the license is not valid. Note that if the license installed for the software does not include API rights, a value of false will be returned. If the license is found to be invalid, the GetLastError() function can be called to determine the reason for the failure (see below).

Function	
Function Signature(s)	***Description***
bool Open(const char* PDFFile, const char* options="") bool Open(const wchar_t* PDFFile, const wchar_t* options=L"")	Open a PDF filename for writing, optionally supplying a string of flags affecting the conversion options.

Input		
Parameter #	***Parameter Name***	***Description***
1	PDFFile	The filename the PDF file should be written to.
2	options	Conversion options, , same as the command-line flags that would be used as parameters to CVCompress.exe. See below for a full description of these flags.

Returns	
C++ Data Type	***Description***
bool	Success code; see APIErrors.h for list of possible values

Function	
Function Signature(s)	***Description***
bool OpenMemory(byte * pTrMemory, size_t sizeLimit, long * pFinalSize, const char* options="")	Open a PDF filename for writing, optionally supplying a string of flags affecting the conversion options.

Input		
Param #	***Param Name***	***Description***
1	pTrMemory	The buffer in memory for writing the PDF file
2	sizeLimit	Maximum number of bytes CVWriter can use in pTrMemory buffer
3	pFinalSize	The final length of the output PDF file is stored in this variable after Close() function is called.
4	options	Conversion options, , same as the command-line flags that would be used as parameters to CVCompress.exe. See below for a full description of these flags.

Returns	
C++ Data Type	***Description***
bool	Success code; see APIErrors.h for list of possible values

Function	
Function Signature(s)	***Description***
bool AddFile(const char* fileName, const char* cOptions) bool AddFile(const wchar_t* fileName, const wchar_t* cOptions)	Add a PDF or image file (such as TIFF, JPEG, or BMP) to the current open PDF file. Two versions of this function are available - one that uses ASCII strings and one that uses Unicode strings.

Input		
Param #	***Param Name***	***Description***
1	fileName	The name of the file that should be converted and added to the current open PDF file
2	cOptions	Optionally specifies alternate options (flags) for this input file, to amend or override the document-level conversion options specified in the Open() API function. At this time, only compression-related or OCR-related options can be specified here.

Returns	
C++ Data Type	***Description***
bool	Success code

Function	
Function Signature(s)	***Description***
bool AddPage(const byte* pDIB, const char* cOptions)	Add an image DIB to the currently open PDF file. Note that this method cannot add an image in stream mode. To do that, you should use the **AddFile** method above.

Input		
Param #	***Param Name***	***Description***
1	pDIB	A handle to the DIB page that should be added the current PDF file.
2	cOptions	Optionally specifies alternate options (flags) for this input page, to amend or override the document-level conversion options specified in the Open() API function. At this time, only compression-related or OCR-related options can be specified here.

Returns	
C++ Data Type	***Description***
bool	Success code

Function	
Function Signature(s)	***Description***
bool AddMemoryStream(const byte* pTrMemory, long memsize, const char* streamtype) bool AddMemoryStream(const byte* pTrMemory, long memsize, const char* streamtype, const char* cOptions)	Add a PDF, TIFF, JPEG, or BMP stored in a memory stream to the current open PDF file. Two version of this method are available, one of them containing an extra parameter for specifying the desired processing flags if they differ from what was specified in OpenMemory().

Input		
Param #	***Param Name***	***Description***
1	pTrMemory	A memory pointer pointing to a TIFF, JPEG, or BMP image stored in memory.
2	memsize	Size of the pTrMemory buffer.
3	streamtype	String representing the file extension normally attributed to the image type stored in pTrMemory, e.g. **pdf**, **tif**, **jpg**, or **bmp**.
4	cOptions	Optionally specifies alternate options (flags) for this memory stream, to amend or override the document-level conversion options specified in the Open() function. At this time, only compression-related or OCR-related options can be specified here.

Returns	
C++ Data Type	***Description***
bool	Success code

Function	
Function Signature(s)	***Description***
int NumPages()	Number of pages written so far (non-stream mode only)

Returns	
C++ Data Type	***Description***
int	Number of pages

Function	
Function Signature(s)	***Description***
bool Close()	Close an open instance of an APICVistaPDFWriter object.

Returns	
C++ Data Type	***Description***
bool	Success code; see APIErrors.h for list of possible values

Function	
Function Signature(s)	***Description***
bool IsOpen()	Checks whether the APICVistaPDFWriter object is open for writing, i.e., whether the Open() method has already been called.
Returns	
C++ Data Type	***Description***
bool	Returns **true** if file is open; **false** otherwise.

Function	
Function Signature(s)	***Description***
int GetLastError()	Get the last error set by the PDF writer
Returns	
C++ Data Type	***Description***
int	Integer error code; see APIErrors.h for list of possible values

Function		
Function Signature(s)		***Description***
const char* GetErrorString(int errorCode) const wchar_t* GetErrorStringW(int errorCode)		Get description of an error for a give error code. Both ASCII and Unicode versions of this function are available.
Input		
Param #	***Param Name***	***Description***
1	errorCode	An integer error code, obtained by calling GetLastError()
Returns		
C++ Data Type		***Description***
const char* / const wchar_t*		A pointer to the string describing the error code

Function	
Function Signature(s)	***Description***
static const char* GetSupportedExtensions() static const wchar_t* GetSupportedExtensionsW()	Get list of supported input file extensions as a Unicode string. Both ASCII and Unicode versions of this function are available.
Returns	
C++ Data Type	***Description***
static const char* / static const wchar_t*	List of supported extensions in the following form (all lowercase): ".ext1|.ext2|.ext3|...|.extn|"

Function

Function Signature(s)	*Description*
bool GetOutputFileName(char* fileName, int arrayLength) bool GetOutputFileName(wchar_t* fileName, int arrayLength)	Get the final filename of the output PDF file. Both ASCII and Unicode versions of this function are available.

Input

Param #	*Param Name*	*Description*
1	fileName	Buffer that will contain the final output filename.
2	arrayLength	Length of the supplied **fileName** buffer.

Returns

C++ Data Type	*Description*
bool	Success code

Function

Function Signature(s)	*Description*
static int NumInputPages(char* fileName, const char* ownerPW, const char* userPW) static int NumInputPages (wchar_t* fileName, const char* ownerPW, const char* userPW)	Get the number of pages in the specified input file. Both ASCII and Unicode versions of this function are available.

Input

Param #	*Param Name*	*Description*
1	fileName	Name of the input file
2	ownerPW	Owner password for the file. Pass NULL if none is used.
3	userPW	User password for the file. Pass NULL if none is used.

Returns

C++ Data Type	*Description*
int	Number of pages

Function		
Function Signature(s)		***Description***
static bool GetPageDib(const char* fileName, unsigned char*& pDIB, int pageNum) static bool GetPageDib(const wchar_t* fileName, unsigned char*& pDIB, int pageNum)		Get the DIB image for a given page in the specified file. Both ASCII and Unicode versions of this function are available.
Input		
Param #	***Param Name***	***Description***
1	fileName	Name of the input file
2	pDIB	Buffer that gets the DIB
3	pageNum	Page number to get
Returns		
C++ Data Type		***Description***
bool		Success code

Function	
Function Signature(s)	***Description***
int NumProcessedPages()	Get the number of pages written to the output file. Note that this function will not return the correct value until the Close() function is called.
Returns	
C++ Data Type	***Description***
int	Number of pages processed

Function	
Function Signature(s)	***Description***
int GetFinalSize()	Gets the final size of the output PDF file. Note that this function will not return the correct value until the Close() function is called.
Returns	
C++ Data Type	***Description***
int	Final size of output file

Function		
Function Signature(s)		***Description***
bool ExtractBookmarks(const wchar_t* fileName)		Extracts the bookmarks from the current file.
Input		
Param #	***Param Name***	***Description***
1	fileName	Name of the file to which the bookmark data should be saved
Returns		
C++ Data Type		***Description***
bool		Success code

Function		
Function Signature(s)		***Description***
static bool DeleteMem(unsigned char*& ptr)		Deletes the memory allocated for a DIB by one of the other API functions. **NOTE**: The parameter's ***data type has changed*** since version 5.0.
Input		
Param #	***Param Name***	***Description***
1	ptr	Pointer to an allocated block of memory
Returns		
C++ Data Type		***Description***
bool		Success code

.NET Implementation

The function signatures provided by the .NET version of the `APICVistaPDFWriter` class for languages such as C# and VB.NET are quite similar to those provided for C++. The specifications are below.

Namespace:	**CVision.PdfCompressor**
Class:	**APICVistaPDFWriterNET**

Member Functions:

Function Signature	Notes
APICVistaPDFWriterNET()	
~**APICVistaPDFWriterNET**()	
bool **AddFile**(string FileName, string Options)	
bool **AddMemoryStream**(byte[] pTrMemory, int Memsize, string Streamtype, string Options)	
bool **AddPage**(System.Drawing.Bitmap oBitmap, string Options)	Takes a .NET Bitmap instead of a DIB
bool **Close**()	
override void **Dispose**()	
bool **ExtractBookmarks**(string fileName)	
string **GetErrorString**(int errorCode)	
uint **GetFinalSize**()	
int **GetLastError**()	
string **GetOutputFileName**()	
static System.Drawing.Bitmap **GetPageDib**(string FileName, int pageNum)	Returns a .NET Image instead of a DIB
static string **GetSupportedExtensions**()	Function is static
bool **HasValidLicense**(long flagsToCheck)	
bool **IsOpen**()	
static int **NumInputPages**(string FileName)	
int **NumPages**()	
int **NumProcessedPages**()	
bool **Open**(string FileName, string Options)	
bool **OpenMemory**(byte[] pTrMemory, int SizeLimit, string Options)	

The APICVistaPDFWriterMP Class

Overview

The **PdfCompressor API** now provides a special version of the APICVistaWriter class that enables fast merging. Under the original APICVistaWriter class, merging files was always forced to be a sequential operation, and this prevented users from taking advantage of their processing power on machines with multiple cores. A new version of this class, called **APICVistaWriterMP**, processes files asynchronously and merges the files in the correct order into the final output filename specified.

To take advantage of multiprocessing when you are **not** merging input files, you do not need the APICVistaWriterMP class. Rather, simply spawn a separate thread for each file you need to process, and create a separate APICVistaWriter object in each thread.

Class and Member Definitions

This section documents the `APICVistaPDFWriterMP` class and its member functions.

At this time, the APICVistaWriterMP class provides only a basic subset of the functions provided by the original APICVistaWriter class. Some of the functions provided in the original APICVistaWriter class only can be invoked by creating a separate APICVistaWriter object.

Data Structure	
C++ Data Type	***Description***
class APICVistaPDFWriterMP	The structure encapsulating the API functions for PDF file creation

Function	
Function Signature(s)	***Description***
APICVistaPDFWriterMP()	The default constructor

Function	
Function Signature(s)	***Description***
~APICVistaPDFWriterMP()	The destructor

<table>
<tr><th colspan="3">Function</th></tr>
<tr><td colspan="2"><b><i>Function Signature(s)</i></b></td><td><b><i>Description</i></b></td></tr>
<tr><td colspan="2">bool Open(const wchar_t* PDFFile, const wchar_t* options="")</td><td>Open a PDF filename for writing, optionally supplying a string of flags affecting the conversion settings.</td></tr>
<tr><td colspan="3"><b>Input</b></td></tr>
<tr><td><b><i>Parameter #</i></b></td><td><b><i>Parameter Name</i></b></td><td><b><i>Description</i></b></td></tr>
<tr><td>1</td><td>PDFFile</td><td>The filename the PDF file should be written to.</td></tr>
<tr><td>2</td><td>options</td><td>Conversion options, same as the command-line. These options only affect the final merging of files, and therefore only annotation-related flags can be used here. Compression and OCR flags should be set in the call to the <b>AddFile()</b> function.</td></tr>
<tr><td colspan="3"><b>Returns</b></td></tr>
<tr><td colspan="2"><b><i>C++ Data Type</i></b></td><td><b><i>Description</i></b></td></tr>
<tr><td colspan="2">bool</td><td>Success code; see APIErrors.h for list of possible values</td></tr>
</table>

<table>
<tr><th colspan="3">Function</th></tr>
<tr><td colspan="2"><b>Function Signature(s)</b></td><td><b>Description</b></td></tr>
<tr><td colspan="2">bool AddFile(const wchar_t* fileName, const wchar_t* cOptions)</td><td>Add a PDF or image file (such as TIFF, JPEG, or BMP) to the current open PDF file.</td></tr>
<tr><td colspan="3"><b>Input</b></td></tr>
<tr><td><b><i>Parameter #</i></b></td><td><b><i>Parameter Name</i></b></td><td><b><i>Description</i></b></td></tr>
<tr><td>1</td><td>fileName</td><td>The name of the file that should be converted and added to the current open PDF file</td></tr>
<tr><td>2</td><td>cOptions</td><td>Specify the flags you would like to use to process the input file. In MP mode, this will usually be the place to specify compression flags and not in the Open method. These options will affect only the specified input file. For a description of the flags, please check below in the appropriate section.</td></tr>
<tr><td colspan="3"><b>Returns</b></td></tr>
<tr><td colspan="2"><b><i>C++ Data Type</i></b></td><td><b><i>Description</i></b></td></tr>
<tr><td colspan="2">bool</td><td>Success code</td></tr>
</table>

<table>
<tr><th colspan="2">Function</th></tr>
<tr><td><b>Function Signature(s)</b></td><td><b>Description</b></td></tr>
<tr><td>bool Close()</td><td>Close an open instance of an <b>APICVistaPDFWriterMP</b> object.</td></tr>
<tr><td colspan="2"><b>Returns</b></td></tr>
<tr><td><b><i>C++ Data Type</i></b></td><td><b><i>Description</i></b></td></tr>
<tr><td>bool</td><td>Success code; see APIErrors.h for list of possible values</td></tr>
</table>

Function	
Function Signature(s)	**Description**
int GetLastError()	Get the last error set by the PDF writer
Returns	
C++ Data Type	***Description***
int	Integer error code; see APIErrors.h for list of possible values

Function		
Function Signature(s)		**Description**
void SetLogFilePath(const wchar_t* path)		Set the path for the log file.
Input		
Parameter #	Parameter Name	Description
1	path	The fully-qualified path to the log file
Returns		
C++ Data Type		***Description***
void		

Function		
Function Signature(s)		**Description**
void LogError(const wchar_t* cFormat, ...)		Write a custom message to the log file.
Input		
Parameter #	**Parameter Name**	**Description**
1	cFormat	The format string to output, conforming to formatting specifications used by C++ library functions such as printf().
2, ...	(n/a)	Any parameters used by the format string
Returns		
C++ Data Type		***Description***
void		

.NET Implementation

The function signatures provided by the .NET version of the `APICVistaPDFWriterMP` class for languages such as C# and VB.NET are quite similar to those provided for C++. The specifications are below.

Namespace:

CVision.PdfCompressor

Class:

APICVistaPDFWriterMP_NET

Member Functions:

Function Signature	Notes
APICVistaPDFWriterMP_NET()	
~APICVistaPDFWriterMP_NET()	
bool AddFile(string FileName, string Options)	
bool Close()	
override void Dispose()	
int GetLastError()	
bool Open(string FileName, string Options)	
void SetLogFilePath(string pathString)	In the .NET version of the API, you can set the log file path, but you cannot output a custom message to the log file.

The APICVistaPDFReader Class

Background Information

This section explains the conventions and terminology used later on in the documentation.

- DIB - An acronym for **Device-Independent Bitmap**, this is the raw Windows bitmap format used to exchange image data within the CVista API. The format of a DIB file is equivalent to that of a BMP file without the BITMAPFILEHEADER section. Additional information on the DIB format can be found at the Microsoft Developer Network (MSDN) website.
- The documentation refers to a **byte** data type. **byte** is defined as:
  ```
      typedef  unsigned char byte;
  ```
- All memory dynamically allocated for an **APICVistaPDFReader** object must be deleted with the **CVPDFReaderDelete** function when the client is done with it.
- All functions described below which have a bool return type representing a success code return true upon success and false upon failure.

Helper Data Structures and Functions

This section documents supporting data structures and functions used in conjunction with **APICVistaPDFReader** objects.

Data Structure		
C++ Data Type		***Description***
struct InfoData_t		Used to communicate document property fields a PDF file.
Data Structure Members		
C++ Data Type	***Name***	***Description***
char*	Title	The PDF file’s title.
char*	Author	The name of the person who created the pdf file.
char*	Subject	The subject of the pdf file
char*	KeyWords	Keywords associated with the PDF file
char*	Creator	The name of the original application that created this pdf file
char*	Producer	The name of the application that modified or converted this pdf file
char*	CreationDate	The date and time the pdf file was created. //YYYYMMDDHHmmSSOHH'mm' format
char*	ModifiedDate	The date and time the document was most recently modified. //YYYYMMDDHHmmSSOHH'mm' format

Note: Any document fields that are not present in the current document are set to NULL. For more information on the document fields see Entries in the document information dictionary section of a PDF Reference manual.

Data Structure	
C++ Data Type	***Description***
struct ImageInfo_t	Structure to store information related to an image embedded in a pdf file.

Data Structure Members		
C++ Data Type	***Name***	***Description***
int	iWidth	Width of the image
int	iHeight	Height of the image
int	iNumComponents	Number of color components used by the image
int	iBitsPerComponent	Number of bits used by each component
int	iBitsPerPixel	Number of bits used by each pixel of the image
int	iXResolution	Horizontal resolution of the image
int	iYResolution	Vertical resolution of the image
int	iFilter	Compression filter used by the image
int	iColorSpace	Colorspace used by the image
char*	cFilterParams	Any parameters related to compression filter used by the image

Function	
Function Signature	***Description***
void CVPDFReaderDelete(void* ptr)	Delete any memory dynamically allocated by **APICVistaPDFReader** API

Input		
Param #	***Param Name***	***Description***
1	void*	A pointer to the memory dynamically allocated by **APICVistaPDFReader** API

Class and Member Definitions

This section documents the APICVistaPDFReader class and its functions.

Data Structure	
C++ Data Type	***Description***
class APICVistaPDFReader	The structure encapsulating the API functions for PDF manipulation

Initialization and access functions

Function	
Function Signature	***Description***
bool HasValidLicense()	Check whether a valid PDF reader license exists
Returns	
C++ Data Type	***Description***
bool	Success code; see APIErrors.h for list of possible values

Function		
Function Signature		***Description***
bool Open(const char* fname,const char* options="") bool Open(const wchar_t* fname, const wchar_t* options=L"")		Open a PDF filename for writing, optionally supplying a string of flags affecting the conversion settings.
Input		
Param #	***Param Name***	***Description***
1	fname	A pointer to a C-style string representing the name of the file to be opened
2	options	Optional decompression flags, same as the command-line flags that would be used as parameters to CVDecompress.exe. See below for a full description of these flags.
Returns		
C++ Data Type		***Description***
bool		Success code

Function	
Function Signature	***Description***
bool Close()	Close the current PDF file
Returns	
C++ Data Type	***Description***
bool	Success code

Note: This function resets all internal data structures. After this call the object instance is ready to have its **Open( )** functions called again with a different file.

Function	
Function Signature	***Description***
bool IsOpen()	Get the **APICVistaPDFReader** object status
Returns	
C++ Data Type	***Description***
bool	Returns true if a pdf files is open.

PDF page control functions

Function	
Function Signature	***Description***
int NumPages()	Get the total number of pages in the current PDF file
Returns	
C++ Data Type	***Description***
int	the total number of pages

Function	
Function Signature	***Description***
int CurPage()	Get the current page in the PDF file
Returns	
C++ Data Type	***Description***
int	The page in the PDF file that the **APICVistaPDFReader** object is pointing to

Function	
Function Signature	***Description***
bool Next()	Advance to the next page in the PDF file
Returns	
C++ Data Type	***Description***
bool	Success code

PDF file functions: saving/conversion

Function

Function Signature	*Description*
bool SaveToFile(const char* outFile, const char* opt) bool SaveToFile(const wchar_t* outFile, const wchar_t* opt)	Save the PDF to a disk file. You can save to any of the formats indicated by the GetSupportedExtensions() function (see below), such as PDF, BMP, JPG or TIFF format. The file extension of the specified filename determines the type of output file. This function is available in both ASCII and Unicode variations.

Input

Param #	*Param Name*	*Description*
1	outFile	Filename that the file should be saved to
2	opt	Reserved optional parameter. Set it to NULL.

Returns

C++ Data Type	*Description*
bool	Success code

Function

Function Signature	*Description*
bool SavePageToFile(const char* outFile, const char* opt) bool SavePageToFile(const wchar_t* outFile, const wchar_t* opt)	Save the current PDF page to a disk file. You can save to any of the formats indicated by the GetSupportedExtensions() function (see below), such as PDF, BMP, JPG, or TIFF format. The file extension of the specified filename determines the type of output file. This function is available in both ASCII and Unicode variations.

Input

Param #	*Param Name*	*Description*
1	outFile	A pointer to the filename the file should be saved to
2	opt	Reserved optional parameter. Set it to NULL.

Returns

C++ Data Type	*Description*
bool	Success code

Function	
Function Signature	***Description***
static const char* GetSupportedExtensions() static const wchar_t* GetSupportedExtensionsW()	Returns supported "SavetTo" file format extensions in the form (all lowercase) ".ext1|.ext2|.ext3|...|.extn|". GetSupportedExtensionsW() is the Unicode version of this function.
Returns	
C++ Data Type	***Description***
static const char*	A pointer to format extension list string.

Error functions

Function	
Function Signature	***Description***
int GetLastError()	Get the value of the last error
Returns	
C++ Data Type	***Description***
int	The integer value of the error

Function		
Function Signature		***Description***
const char* GetErrorString(int errCode) const wchar_t* GetErrorString(int errCode)		Get the string value of the error code. This function is available in ASCII and Unicode variations.
Input		
Param #	***Param Name***	***Description***
2	options	Optional decompression flags. See the appendix in the main PdfCompressor documentation for more information.
Returns		
C++ Data Type		***Description***
const char*		A pointer to the error's description

.NET Implementation

The function signatures provided by the .NET version of the `APICVistaPDFReader` class for languages such as C# and VB.NET are quite similar to those provided for C++. The specifications are below.

Namespace CVision.PdfCompressor

Class APICVistaPDFReaderNET

Member Functions

Function Signature	*Notes*
APICVistaPDFReaderNET()	
~**APICVistaPDFReaderNET**()	
bool **Close**()	
int **CurPage**()	
bool **GetDocInfoFields**(InfoDataNET oDocInfo)	InfoDataNET object has same fields as struct InfoData_t from C++
System.Drawing.Image **GetImageBitmap**(int iImageNumber)	Equivalent of **GetImageDib**() function, but returns a .NET image instead of a DIB
bool **GetImageInfo**(int iImageNumber, ImageInfoNET oImageIn)	ImageInfoNET object has same fields as struct ImageInfo_t from C++
MemoryStream **GetImageStream**(int iImageNumber)	
int **GetLastError**()	
System.Drawing.Image **GetPageBitmap**(string Options)	Equivalent of **GetPageDib**() function, but returns a .NET image instead of a DIB
bool **GoToPage**()	
bool **HasValidLicense**()	
bool **IsOpen**()	
bool **IsPageImageOnly**()	Check if current page of the open document is an **Image-Only** page. Note that if you check a pdf file that has more than one stream, it will automatically return false.
bool **Next**()	
int **NumImages**()	
int **NumPages**()	
bool **Open**(string FileName, string Options)	
bool **Prev**()	
bool **SavePageToFile**(string OutFileName)	
bool **SaveToFile**(string OutFileName)	

Compression Flags

The following is the full list of flags used to specify the various desired options for use with functions in the **APICVistaPDFWriter API**, such as **Open()** and **AddFile()**. These are the same flags as the ones that would be used in conjunction with CVCompress.exe. The groupings correspond roughly to the options pages that appear in the **PdfCompressor Wizard GUI**.

Compression-Related Options

Option/Flag	Description
-acroVersion <val>	Specifies the minimum version of Adobe Acrobat and Adobe Reader with which the output PDF file will be compatible. Setting this version to a higher value enables you to take advantage of more features, but this comes at the cost of being incompatible with earlier version of Adobe Reader. Setting acroVersion to 5 or even 6 should generally be safe unless you know that you have some users with very old versions of Adobe Reader that can't be upgraded readily. The main features added to Acrobat 5 were JBIG2 bitonal compression and enhanced security options. The main feature added to Acrobat 6 was JPEG2000 compression. Note that a special version value of 128 specifies PDF/A compatibility mode.
-minCompRatio <val> [**pdfonly**]	Specifies a minimum compression ratio. If the output file size is more than val times the input file size, the output file is discarded and the input file is copied in its place, if applicable. If the **pdfonly** keyword is added after the value, the minimum ratio criterion will be applied only when the input file is in PDF format.
-m <mode>	When running in perceptually lossless mode (the default), this flag determines the type of symbolic matching. Mode '0' is faster but generates larger files. Mode '1' provides the best compression rates. Mode '2' is slower and is included for those who prefer to use our 4.0 matcher. Mode '1' is recommended, and is the default if no mode is specified.
-lossless[<mode>]	Directive to compress images in lossless mode. This mode provides a lower level of compression and is therefore not recommended unless you absolutely need a pixel-for-pixel replica of the original document (e.g., for legal reasons). Mode '0' provides the best compression, but runs slowly. Mode '1' uses a balance of speed and compression. Mode '2' is faster but generates much larger files. Mode '1' is recommended, and is the default if no mode is specified. This flag should *not* be used together with the **-m** flag.

Option/Flag	Description
-halftone	Specifies that the halftone algorithm will be used when compressing bitonal images in perceptually lossless mode. Halftoning is the screening effect found in newspaper images and the like, where a bitonal photograph approximates greyscale tones by varying the size and placement of a fine series of dots.
-colorComptype <val>	Specifies the compression method to be used for compressing color and greyscale images. Currently three compression methods are supported: DCT, JPEG2000, and Mixed Raster Content (MRC). The value of '0' specifies DCT (JPEG) compression, the value of '1' specifies JPEG2000 compression, and the value of '2' specifies MRC compression, which is also known as Auto-Segmentation.
-mrcColorComptype <val>	Specifies the compression filter to be used when compressing the background and foreground color layers using MRC compression. The supported MRC color compression filters are DCT (JPEG) and JPEG2000. The DCT compression filter can be specified with the value of '0' and the JPEG2000 compression filter can be specified with the value of '1'.
-mrcResample <val>	If <val> equals 1, this specifies that MRC (auto-segmented) image streams should be auto-resampled. If <val> is 0 or this entire flag is not present, the default is that MRC streams will not be auto-resampled.
-mrcQuality <val>	Controls the quality setting for the PDFs generated using MRC compression. The MRC quality value ranges from 1 to 10. The quality setting of 10 yields highest quality MRC PDFs and the quality setting of 1 yields lowest quality MRC PDFs. The lower quality settings generates highly compressed PDFs and vice versa. The default value for **-mrcQuality** is 7.
-qualityc <val> **-qualityg** <val>	Sets the target quality of color and greyscale images, respectively, using the DCT (JPEG) compression filter. <val> should be an integer from 1 to 99. Note that if auto-segmentation (MRC) is used, these flags are ignored, and the image quality depends on **-mrcQuality**.

Option/Flag	Description
-dctsc <val> **-dctsg** <val>	Sets the smoothing level for color and greyscale images, respectively, using the DCT (JPEG) compression filter. <val> should be an integer from 0 to 100, where 0 means no smoothing. (These flags can be omitted entirely if no smoothing is desired.) Note that if auto-segmentation (MRC) is used, these flags are ignored.
-jpxratioc <val> **-jpxratiog** <val>	Sets the target JPEG2000 compression ratio for color and greyscale images, respectively. If either of these flags is not included, DCT-based compression will be used instead for that colorspace. The <val> parameter should be expressed as a floating-point number between 0.0 and 1.0, using a period (".") as a decimal separator. The smaller the value, the greater the compression, but greater compression comes at the cost of image quality reduction. Note that if auto-segmentation (MRC) is used, these flags are ignored, and the image quality depends on **-mrcQuality**. These two flags should NOT be used simultaneously with the DCT quality and smoothing flags.
-cconc **-ccong**	These two flags turn on compression for color and greyscale ICC-based images, respectively. By default, ICC-based images are not compressed, due to the risk of a slight color shift.
-inline	Compress all inline images (images stored in the content stream of a PDF file). By default, only large inline images are compressed. Compressing small inline images often results in little compression, thereby slowing down performance needlessly.
-huffman <val>	Indicates that MMR (Huffman) encoding should be used. Using MMR encoding can speed up time to render pages to both the screen and the printer, but this can result in a larger file size. The <val> parameter should be either 1 for full MMR encoding or 2 for auto MMR encoding.

Option/Flag	Description
-jpxresample <val>	If <val> equals 1, this specifies that JPEG2000 image streams should be auto-resampled. If <val> is 0 or this entire flag is not present, the default is that JPEG2000 streams will not be auto-resampled.

Output Options

Option/Flag	Description
-pdfPageDim <width> <height>	Manually specifies the output page dimensions. The width and height (in inches) should be listed after the flag. If -pdfPageDim is not specified (and -pdfPageSize is not specified either), then PdfCompressor will automatically determine the output page dimensions.
-pdfPageSize <type>	Manually specifies a named page size for the output file. The <type> parameter can be one of the following: **letter** **legal** **A4** **A5** **A6** If -pdfPageSize is not specified (and -pdfPageDim is not specified either), then PdfCompressor will automatically determine the output page dimensions.
-pdfPagePrintMargin <val>	Specifies an optional whitespace margin for the output file, in inches. The output image will be scaled to fit within the bounds of the margin.

<table>
<tr><th>Option/Flag</th><th>Description</th></tr>
<tr><td><b>-pdfARGBProfile</b> <val></td><td>If PDF/A mode is specified (by using <b>-acroVersion 128</b>), this flag will determine the RGB profile to use. The val parameter can be one of the following:
<table>
<tr><th><val></th><th>Meaning</th></tr>
<tr><td><b>0</b></td><td>sRGB IEC61966-2.1</td></tr>
<tr><td><b>1</b></td><td>Adobe RGB (1998)</td></tr>
<tr><td><b>2</b></td><td>Apple RGB</td></tr>
<tr><td><b>3</b></td><td>ColorMatch RGB</td></tr>
</table>
</td></tr>
<tr><td>-generateThumbnails <mode></td><td>Indicates that JPEG thumbnail images should be created for the document in the same folder as the output file. The <mode> parameter can have one of three values:
<table>
<tr><th><mode></th><th>Meaning</th></tr>
<tr><td><b>all</b></td><td>Creates a thumbnail file for each page of the input file.</td></tr>
<tr><td><b>first</b></td><td>Creates a thumbnail file for the first page only.</td></tr>
<tr><td><b>firstnonblank</b></td><td>Creates a thumbnail file for the first non-blank page only.</td></tr>
</table>
</td></tr>
<tr><td>-thumbnailsize <val></td><td>Specifies the size, in pixels, that the thumbnail files should be. The larger dimension of the image will be scaled to this size.</td></tr>
</table>

Document Structure Options

Option/Flag	Description
-linearize	Causes the output files to be "web-optimized", by creating them with an internal structure that facilitates efficient transmission and viewing of PDF documents through a web browser. Linearized PDF files allow you to jump directly to a given page and display it in your web-based PDF viewer, even before the entire file is downloaded. Note that not all PDF viewers support this functionality. The Adobe Acrobat Reader™ web browser plugin is one viewer that does.
-thumb <val>	Determines whether to keep thumbnails from the original document. <val> can be one of **on**, **off**, or **auto**. If it is set to auto, the software automatically determines whether to keep thumbnails. New thumbnails are not created. It should be noted, however, that Adobe Reader™ often creates thumbnails on the fly anyway, making it appear as if the file still contains thumbnails despite their removal.

PDF-to-PDF Processing Options

If the input file format is PDF, the file may contain multiple regions on each page. They may be visual elements such as images and text streams, or they may be special elements such as bookmarks and form objects. The compressed file can be created in one of two ways:

- ***Stream-based mode:*** In this mode, all of the individual page elements will be preserved as-is, attempting only to compress the existing image content on the page.
- ***Rasterized mode:*** In this mode, the entire page is "flattened" before being compressed, treating the entire page as a single scanned image. All special information such as bookmarks and forms will be lost, and any font-based text regions will be rasterized and treated as part of the page image.

The options below describe how to toggle between the two modes, as well as some additional flags that apply only to rasterized mode.

Option/Flag	Description
-stream	Turns stream-based mode on, compressing all images within the PDF file on a stream-by-stream basis, leaving the original structure of the PDF intact. This option can also be used with input files that are simple image PDFs without individual streams, and can in fact be beneficial in that case as well. Thus, for PDF input files, this flag is recommended. Compression of other file formats should not use this flag, however, as it would cause a syntax error. If this flag is not specified, the PDFs will be compressed in rasterized mode.
-pdfres <xres> [<yres>]	If compressing PDF files in rasterization mode (i.e. the -stream flag is NOT present), this flag manually sets the desired output resolution. If this flag is not specified, the compression engine will attempt to automatically determine the resolution from the input file (which is not always a reliable procedure). <xres> is a value specifying the horizontal resolution and <yres> is a value specifying the vertical resolution. If <yres> is not specified explicity, it will be assumed the same as <xres>

Option/Flag	Description
-pdfbw **-pdfgray** **-pdfcolor**	In rasterized mode, using one these three flags forces the input PDF files to be converted to a given colorspace (bitonal, greyscale, or color, respectively) before compressing. At most, only one of these flags should be specified. If none of these flags is specified, the software will guess the colorspace of each page.
-kbg	Used in conjunction with the **-pdfbw** flag in rasterized mode. Keeps the background layer and dithers it separately from the foreground.

OCR-Related Options

Option/Flag	Description
-o [**-oocr**] [**-oicr**] [**-obarcode**]	Master switch for enabling PdfCompressor's OCR/recognition features. Must be used in conjunction with at least one of the optional flags: **-oocr** Turns on OCR (scans mechanically or electronically printed pages) **-oicr** Turns on ICR (scans handwritten text) **-obarcode** Turns on barcode recognition The other OCR/recognition-related options listed below are applicable only when the **-o** flag is set.
-ocrmode [<mode>]	This setting determines the balance between speed and accuracy used by the OCR engine. <mode> can be **realtime**, fast, **accurate**, or superaccurate. The default setting is **fast**. As a rule the slower the OCR, the more accurate it will be. In order of speed realtime is the fastest, followed by **fast**,**accurate**, and **superaccurate**.
-oraster	Forces the OCR engine to analyze the entire page as a single image, even if the input page is a PDF with multiple image streams.
-ocrwordconf <val>	Sets the minimum confidence level for OCR text that is recognized. <val> should be an integer from 0 to 100, representing a confidence percentage. Any text that has a confidence level below <val> will be discarded.
-ocrtwod	Enable two-dimensional (multidirectional) OCR. This allows the OCR engine to recognize text in multiple orientations within the same image. Note that two-dimensional OCR does not work with **-ocrzone** .

Option/Flag	Description
-ocrdict <file>	Use the custom OCR dictionary specified by <file> .
-ocrzone <file>	Use the OCR Zone file specified by <file> .
-dsoff	**-dsoff** disables checking for document skew (alternate document orientations), which is on by default. It is recommended to include this flag if you know that none of your pages are rotated.
-lang <language>	Specifies the language dictionary to use with the OCR engine. Using a dictionary that is native to the language of the document can greatly improve OCR results. If this flag is not included, the English language dictionary is used. More information on obtaining and installing additional language dictionaries for CVista PdfCompressor, as well as a list of <language> parameters supported, can be found online at: http://www.cvisiontech.com/langpack_instructions.html
-lsize <var>	Specifies that line-based OCR should be used instead of word-based OCR. Word-based OCR can sometimes result in more accurate bounding boxes displayed around search hits, but this can also slow down the OCR engine. In most cases, line-based OCR is recommended. The <var> parameter specifies the maximum number of words that should be grouped as a single line. The GUI uses **-lsize 25** as the default.
-ot <val>	**-ot** <val> sets a threshold for how long you want to allow the OCR process to take on each page. <val> is specified in seconds. 120 seconds is often a good value to use. If the threshold is exceeded for a given page, that page will not contain OCR information.

<table>
<tr><th>Option/Flag</th><th>Description</th></tr>
<tr><td><b>-ocroutput</b>
<format&gt;</td><td>Outputs a file in the specified format along with every document compressed, containing all of the OCR text from the document. Acceptable <format> options are as follows:
<table>
<tr><th>&lt;format&gt;</th><th>Format</th><th>Extension</th></tr>
<tr><td>0</td><td>Text (ASCII)</td><td>.txt</td></tr>
<tr><td>1</td><td>Excel 97/2000</td><td>.xls</td></tr>
<tr><td>2</td><td>HTML</td><td>.htm</td></tr>
<tr><td>3</td><td>Open EBook 1.0</td><td>.opf</td></tr>
<tr><td>4</td><td>PowerPoint 97 (RTF)</td><td>.rtf</td></tr>
<tr><td>5</td><td>Basic RTF</td><td>.rtf</td></tr>
<tr><td>6</td><td>Word 2000 (RTF)</td><td>.rtf</td></tr>
<tr><td>7</td><td>WordPad (RTF)</td><td>.rtf</td></tr>
<tr><td>8</td><td>WordPerfect 8</td><td>.wpd</td></tr>
<tr><td>9</td><td>XML</td><td>.xml</td></tr>
<tr><td>13</td><td>Publisher 98 (RTF)</td><td>.rtf</td></tr>
<tr><td>15</td><td>Word 2000/XP</td><td>.doc</td></tr>
<tr><td>16</td><td>Text (Unicode)</td><td>.txt</td></tr>
<tr><td>17</td><td>XPS</td><td>.xps</td></tr>
<tr><td>18</td><td>Searchable XPS</td><td>.xps</td></tr>
</table>
</td></tr>
</table>

Option/Flag	Description
-omergeoutput <val>	Specifies how to handle merging of auxiliary output files: **<val> Meaning** 0 Save the auxiliary OCR output from each page to a separate file. 1 Merge the auxiliary output from all pages of a multipage file into a single file. 2 If the folder is being processed in merge mode, whereby all input files are merg into a single PDF output file, this will similarly merge the auxiliary output from a files into a single file. If the folder is not being processed in merge mode, this w behave as if a value of 1 was passed, and auxiliary output will be merged on a p file basis only. The default behavior without any flags is to merge auxiliary output on a per-file basis only.

General Image Processing Options

Option/Flag	Description
-c {**ON**|**OFF**}	Turns bitonal cleaning **ON** or **OFF**. Cleaning removes very small stray dots that were clearly artifacts of the scanning process. By default, cleaning is **ON** for perceptually lossless mode and **OFF** for lossless mode. If a file is compressed in lossless mode, the image first undergoes the cleaning process, and then the cleaned image is compressed in a lossless maner.
-v	Turns bitonal despeckling on. More ambitious than regular cleaning, this removes larger stray specks that are judged to be artifacts from scanning or photocopying. This computerized judgment call is not infallible, however, so the resultant document should be checked to ensure that no portion of the actual image was lost in this process.
-hb <mode> **-hg** <mode> **-hc** <mode>	Attempts to smooth out jagged edges in a bitonal, greyscale, or color image, respectively. <mode> is a value from 1 to 4. '1' is Smart Smoothing, which varies the level of smoothing in different parts of the image. '2' will automatically pick a level of aggressiveness based on the DPI of the image. '3' will use the less aggressive smoothing, and '4' will use more aggressive smoothing.
-c2b <val> **-g2b** <val>	Remaps color and greyscale images, respectively, to bitonal. <val> should be a value from 0 to 255 indicating the threshold level. The higher the number, the darker the image will tend to be.
-c2g	Remaps color images to greyscale.
-rscdpi <val> **-rsgdpi** <val> **-rsbdpi** <val>	Resamples color, greyscale, and bitonal images, respectively, to a given DPI value specified by <val> , regardless of the original image resolution.

Option/Flag	Description
-rscinterp <val> **-rsginterp** <val> **-rsbinterp** <val>	Specifies the interpolation method to use for color, greyscale, and bitonal images. The <val> parameter can be one of the following: **nearestneighbor** **bilinear** **smartbicubic** **bicubic** If no flags are specified, the default interpolation method is **nearestneighbor** for bitonal images and **smartbicubic** for greyscale and color images.
-rscdwndpi <val> **-rsgdwndpi** <val> **-rsbdwndpi** <val>	Downsamples color, greyscale, and bitonal images, respectively, to a given DPI value specified by <val> . Only images that originally had a higher DPI than <val> will be resampled.
-rscupdpi <val> **-rsgupdpi** <val> **-rsbupdpi** <val>	Upsamples color, greyscale, and bitonal images, respectively, to a given DPI value specified by <val> . Only images that originally had a lower DPI than <val> will be resampled.
-rsc <val> **-rsg** <val> **-rsb** <val>	Resamples color, greyscale, and bitonal images, respectively, by a given percentage specified by <val> . This should be a floating-point number, using a period (".") as a decimal separator.

Annotations, Document Tags, and Viewer Preferences

Some features of PdfCompressor, particularly those pertaining to annotations and document properties, require the specification of several parameters. Due to this fact, a configuration file is used for specifying many of these new options instead of passing parameters directly on the command line. The configuration file itself is passed to PDFCompress.exe as one of the command-line flags in the compression options. The syntax is as follows.

Option/Flag	Description
-config <filename>	Specifies the name of a config file that contains settings for some of the features of PdfCompressor with more extensive sets of parameters. This file can include information about stamps, watermarks, text annotations, document properties, viewer preferences, and batch auditing. The <filename> parameter should be the full path to the file. The config file is described in greater detail in Appendix B.

Security Options

Option/Flag	Description
-ownerpw <passwd> **-userpw** <passwd>	If an input file is password-protected, one or both of these flags are required to gain access to the file in order to compress it. **-ownerpw** specifies the owner password, and **-userpw** specifies the user password. Note, however, that the output file will NOT be password-protected unless you re-encrypt using the **-encrypt** flag, described below.

<table>
<tr><th>Option/Flag</th><th>Description</th></tr>
<tr><td><b>-encrypt</b> &lt;rev&gt; <npw> <opasswd> [<upasswd>] <flags&gt;</td><td>
<p>Encrypts the output file.</p>
<p>&lt;rev&gt; is the revision number of the encryption model. If this is set to <b>2</b>, 40-bit encryption is used. If it is set to <b>3</b>, 128-bit encryption is used.</p>
<p>&lt;npw&gt; indicates the number of passwords to use. This should be set to <b>1</b> or <b>2</b>. This parameter should then be followed by either one or two passwords. <opasswd> is the owner password, which is always required and allows for the changing of permissions. <upasswd>, if specified, indicated the password needed by a user to open the document.</p>
<p>&lt;flags> is a bitwise OR of various permissions flags. Note that the bit positions of these flags are NOT exactly the same as in the Adobe PDF Specification. The mapping of these bits to those in the Adobe PDF Specification is as follows:</p>
<table>
<tr><th>CVISION bit pos (val)</th><th>Adobe bit pos</th><th>Meaning</th></tr>
<tr><td>1 (1)</td><td>3</td><td>ALLOW_PRINT</td></tr>
<tr><td>2 (2)</td><td>4</td><td>MODIFY_CONTENTS</td></tr>
<tr><td>3 (4)</td><td>5</td><td>COPY_EXTRACT</td></tr>
<tr><td>4 (8)</td><td>6</td><td>ADD_MODIFY</td></tr>
<tr><td>5 (16)</td><td>7</td><td>FILL_FORMS</td></tr>
<tr><td>6 (32)</td><td>10</td><td>EXTRACT</td></tr>
<tr><td>7 (64)</td><td>11</td><td>ASSEMBLE</td></tr>
<tr><td>8 (128)</td><td>12</td><td>PRINT_HIGHRES</td></tr>
</table>
<p>So, for example, to enable COPY_EXTRACT and FILL_FORMS, you would specify a value of (4 | 16) = 20.</p>
<p>See the Adobe PDF Specification for further description of the meaning of each of these flags.</p>
</td></tr>
</table>

Logging Options

Some of the logging options are described below. Also see “Configuration File Overview” in the *PdfCompressor User Guide* for a description of how to set up an audit log, which is done through the config file.

Option/Flag	Description
-log <filename>	Generates an error log for batch compression. <filename> should be the fully-qualified path to the error log file.

Decompression Flags

The following is the list of flags used to specify the various desired options for use with Open() functions in the APICVistaPDFReader API. They are described below.

<table>
<tr><th>Option/Flag</th><th>Description</th></tr>
<tr><td><b>-q</b></td><td>Run in quiet mode (suppress verbose messages).</td></tr>
<tr><td><b>-tifcompcg</b> &lt;val&gt;</td><td>Sets the encoding options for output TIFF images.
val can be one of the following:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>raw</td><td>Use raw RGB encoding.</td></tr>
<tr><td>rawcmyk</td><td>Use raw CMYK encoding.</td></tr>
<tr><td>lzw</td><td>Use LZW compression.</td></tr>
<tr><td>lzwcmyk</td><td>Use LZW CMYK compression.</td></tr>
<tr><td>packbits</td><td>Use packbits compression.</td></tr>
<tr><td>packbitscmyk</td><td>Use packbits CMYK compression.</td></tr>
<tr><td>jpegrgb</td><td>Use JPEG RGB compression.</td></tr>
<tr><td>jpegcmyk</td><td>Use JPEG CMYK compression.</td></tr>
<tr><td>jpegycbcr</td><td>Use JPEG YCbCr compression.</td></tr>
</table></td></tr>
<tr><td><b>-tifcompbw</b> &lt;val&gt;</td><td>Sets the TIFF compression options for generating bitonal TIFF images.
val can be one of the following:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>raw</td><td>Use raw encoding.</td></tr>
<tr><td>lzw</td><td>Use LZW compression.</td></tr>
<tr><td>ccitt3</td><td>Use CCITT3 compression.</td></tr>
<tr><td>ccitt4</td><td>Use CCITT4 compression.</td></tr>
<tr><td>ccittrle</td><td>Use CCITT-RLE compression.</td></tr>
</table></td></tr>
</table>

Option/Flag	Description
-pdfres <xres> [<yres>]	If compressing PDF files in rasterization mode (i.e. the -stream flag is NOT present), this flag manually sets the desired output resolution. If this flag is not specified, the compression engine will attempt to automatically determine the resolution from the input file (which is not always a reliable procedure). <xres> is a value specifying the horizontal resolution and <yres> is a value specifying the vertical resolution. If <yres> is not specified explicity, it will be assumed the same as <xres>.
-pdfbw **-pdfgray** **-pdfcolor**	In rasterized mode, using one these three flags forces the input PDF files to be converted to a given colorspace (bitonal, greyscale, or color, respectively) before compressing. At most, only one of these flags should be specified. If none of these flags is specified, the software will guess the colorspace of each page.
-kbg	Used in conjunction with the -pdfbw flag in rasterized mode. Keeps the background layer and dithers it separately from the foreground.
-quality <val>	Sets the target quality of color and greyscale images when outputting to either JPEG format or a JPEG-encoded TIFF format. <val> should be an integer from 1 to 99. Note that this flag has no effect if using a non-JPEG encoding.
-split	When the output formats supports multipage files (as in TIFF or PDF), this flag indicates that the output pages of a multipage file should be split into individual files. Each file will contain the original base output filename but with a numeric suffix indicating the page number. Other formats that do not support multipage files will always follow this convention. **NOTE:** This flag is not part of the main decompressions flags and must be included **after the output file path** in the command line call. This flag is not available via the API .