PDF-XChange Viewer Simple DLL SDK

User Manual

Head Quarters:
Tracker Software Products (Canada) Ltd.,

P.O. Box 79, 9622 Chemainus Road
Chemainus. V0R 1K0
British Columbia, Canada

Sales
Tel: Canada (+00) 1-250-324-1621
Fax: Canada (+00) 1-250-324-1623

In Europe:

Unit 17, Raleigh Court. Priestly Way.
Crawley. Sussex. RH10 9PD.
England.

Sales
Tel: +44 (0) 20 8555 1122
Fax: (+00) 1-250-324-1623

http://www.tracker-software.com

sales@tracker-software.com

Created: Wednesday, April 09, 2014

PDF-XChange Viewer Simple DLL SDK

Table of Contents

Chapter 1: Welcome 1

Chapter 2: Getting Started 2
System Requirements 3
Redistribution 3

Chapter 3: Functions 4
PXCV_CheckPassword 5
PXCV_Delete 6
PXCV_DrawPageToDC 7
PXCV_DrawPageToDIBSection 9
PXCV_FinishReadDocument 11
PXCV_GetDocumentInfoW 12
PXCV_GetPageDimensions 14
PXCV_GetPageRotation 15
PXCV_GetPagesCount 16
PXCV_GetPermissions 17
PXCV_Init 18
PXCV_ReadDocumentFromIStream 19
PXCV_ReadDocumentFromMemory 20
PXCV_ReadDocumentW 21
PXCV_ReleaseCachedData 23
PXCV_ReleasePageCachedData 24
PXCV_SetCallBack 26
PXV_CommonRenderParameters 26
test 29

Chapter 4: Error handling 31
Error Codes 31
PXCV_Err_FormatFacility 35
PXCV_Err_FormatSeverity 36
PXCV_Err_FormatErrorCode 37

Chapter 5: Index 38

Welcome

PDF-XChange Viewer Simple DLL SDK

Help **Manual**

Welcome to the PDF-XChange Viewer Simple DLL SDK online help system. Browse through the help pages by clicking on the icons below or selecting pages in the table of contents to the left. To quickly find specific product information, enter search criteria in the search box above and click the search button.

Getting Started

Functions

Error handling

Ask Us

If you're unable to find what you're looking for in this help system, try these alternative resources:

- Our Website
- Knowledgebase
- FAQ

or email our support team: support@tracker-software.com

Getting Started

PDF-XChange Viewer Simple DLL SDK

Introduction

The PDF-XChange Viewer Simple DLL SDK is available as a stand alone Developer kit and also included with our PDF-XChange and PDF-Tools SDK's. The PDF-XChange Viewer offers both a Direct DLL method to create simple PDF viewing and Printing within your application or a full ActiveX which allows the use of the full PDF-XChange Viewer within your product. The PDF-XChange Viewer is not a Royalty Free tool kit and a limited number of Licenses are included for end user distribution with your application with the SDK product you purchase, with additional bulk license packs available for a modest cost - see our web site/price lists for more detailed information.

This toolkit as with all our developer kits may not be used to develop Toolkits or Components of any type for use by other non-licensed developers or for use to assist in the creation of Printer driver under the usual license conditions provided. For more information on licensing please read the license agreement and if any doubt as to whether your intended use would be in breach of the license terms please contact us to discuss your needs in more depth as we do offer alternate licensing and will tailor our agreement to meet your needs in most circumstances under different terms of supply.

Support

Support is available direct from our user forums http://www.tracker-software.com/forum.

We recommend that developers use the evaluation download as extensively as possible prior to purchase. The evaluation versions are fully functional with no time out or other crippling mechanism – save that a watermark is stamped on any PDF page generated by the Driver/Library tools – only on purchase will you be provided with the serial number and unlock string required by each component to be passed within your application code (see the demo applications provided) to enable PDF generation without this demo watermark stamp.

By virtually creating your application to the point where you are ready for distribution before you purchase, you are guaranteed satisfaction and we do not disappoint developers asking for refunds once purchased – we do not offer any money back options – so please ensure you are 100% satisfied before you purchase.

Developer's may also find it useful when developing applications for the purpose of creating and manipulating Adobe PDF Formats - to download the documentation relevant to this format from the Adobe web site - at the time of writing this is currently free and may prove useful in explaining in more detail the functionality available. http://www.adobe.com/.

Tracker Software Products Ltd also provide End User and Developer Tool Kits for the creation and manipulation of PDF and Raster Image files and Virtual Printer Drivers. For more information please visit http://www.tracker-software.com

System Requirements

PDF-XChange Standard 2012 supports all Windows (32/64 bit) operating systems from Windows XP**** and later.

Version 5 (2012): Microsoft/Citrix Terminal Server compatible*.
Version 4: Microsoft/Citrix Terminal Server compatible*.
Version 3: Still available for Windows 95/98

*** Note: Though many users have virtualized some of our component products such as the PDF-XChange Viewer and PDF-Tools application using XenApp, we do not support this at this time.Particularly the printer drivers are not designed to work in a virtualized environment.**

**** Limitations to product support for Windows XP**

**To experience the best performance of our products on all Windows operating systems, the onus is on users to ensure that
they have all the latest available Microsoft Windows Service Packs & Updates installed.**

Redistribution

Redistribution of the PDF-XChange Viewer

Redistribution of PDF-XChange Viewer SDK components

The PDF-XChange Viewer SDK depends only on the `pxcview.dll`, and is not reliant on any other PDF-XChange/Tools Image-XChange SDK components. However, please note, The PDF-XChange Viewer takes advantage of the Microsoft© GDI+ for vector printing and it is required to have installed it on the OS where it is not installed by default (all Windows prior to Windows XP). The PDF-XChange Viewer is available for Windows 2000 and later only - earlier versions of Windows are not supported.

Functions

Viewer Simple DLL SDK Functions and Structures

» PXCV_CheckPassword
» PXCV_Delete
» PXCV_DrawPageToDC
» PXCV_DrawPageToDIBSection
» PXCV_FinishReadDocument
» PXCV_GetDocumentInfoW
» PXCV_GetPageDimensions
» PXCV_GetPageRotation
» PXCV_GetPagesCount
» PXCV_GetPermissions
» PXCV_Init
» PXCV_ReadDocumentFromIStream
» PXCV_ReadDocumentFromMemory
» PXCV_ReadDocumentW
» PXCV_ReleaseCachedData
» PXCV_ReleasePageCachedData
» PXCV_SetCallBack
» PXV_CommonRenderParameters

PXCV_CheckPassword

PXCV_CheckPassword validates the supplied password against the current document.

This function should only be called after **PXCV_ReadDocumentW** returns PS_ERR_DocEncrypted.

```
HRESULT PXCV_CheckPassword(
      PXVDocument Doc,
      BYTE* pPassword,
      DWORD PassLen
);
```

Parameters

Doc
: [in] *Doc* specifies the PDF object previously created by the function **PXCV_Init**.

pPassword
: [in] *pPassword* specifies a pointer to a buffer which contains password data (buffer may contain zero `'\0'` symbol(s)).

PassLen
: [in] *PassLen* specifies the length of the buffer.

Return Values

If the function succeeds, the return value is one of the following:

Value	Meaning
1	User password
2	Owner password

If the function fails, the return value is error code.

See Also

PXCV_Init, **PXCV_Delete**, **PXCV_ReadDocumentW**, **PXCV_FinishReadDocument**

PXCV_Delete

PXCV_Delete releases the PDF object, created previously using the **PXCV_Init** function.

You must call this function once the PDF object is no longer required or all updates are complete.

```
HRESULT PXCV_Delete(
      PXVDocument Doc
);
```

Parameters

Doc

[in] *Doc* specifies the PDF object previously created by the function **PXCV_Init**.

Return Values

If the function succeeds, the return value is `DS_OK`.

If the function fails, the return value is error code.

PXCV_DrawPageToDC

PXCV_DrawPageToDC draws the specified page (or part thereof) of the document to a specified device context (DC).

```
HRESULT  PXCV_DrawPageToDC(
      PXVDocument Doc,
      DWORD page_num,
      HDC hDC,
      LPPXV_CommonRenderParameters pParams
);
```

Parameters

Doc
    [in] *Doc* specifies the document previously created by the **PXCV_Init** function.
page_num
    [in] *page_num* specifies the zero-based page number to be drawn.
hDC
    [in] *hDC* specifies handle of the device context onto which the page information should be drawn.
pParams
    [in] Pointer to the **PXV_CommonRenderParameters** structure, which defines drawing parameters.

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or other value which isn't an error code.

Example (C++)

```cpp
HRESULT DrawPageThumbnail(PXVDocument pDoc, DWORD page_num, LPCRECT
thumb_bound_rect, HDC dc)
{
    HRESULT hr;
    double pw, ph;
    hr = PXCV_GetPageDimensions(pDoc, page_num, &pw, &ph);
    if (IS_DS_FAILED(hr))
        return hr;
    // calculation rect of thumbnail in pixels
    // (fitting page proportional into thumb_bound_rect)
    LONG tbw = thumb_bound_rect->right - thumb_bound_rect->left;
    LONG tbh = thumb_bound_rect->bottom - thumb_bound_rect->top;
    LONG tw = tbw;
    LONG th = tbh;
    double z1 = (double)tw / pw;
    double z2 = (double)th / ph;
    if (z1 >= z2)
    {
        tw = (LONG)(z2 * pw + 0.5);
    }
    else
    {
        th = (LONG)(z1 * ph + 0.5);
    }
    RECT thumb_rect;
    thumb_rect.left = thumb_bound_rect->left + (tbw - tw) / 2;
    thumb_rect.top = thumb_bound_rect->top + (tbh - th) / 2;
    thumb_rect.right = thumb_rect.left + tw;
    thumb_rect.bottom = thumb_rect.top + th;
    // now filling PXV_CommonRenderParameters structure
    PXV_CommonRenderParameters crp;
    crp.WholePageRect = &thumb_rect;
    crp.DrawRect = NULL; // because we will draw whole page. It is equal to:
crp.DrawRect = &thumb_rect;
    crp.Flags = 0;       // should be zero as specified
    crp.RenderTarget = pxvrm_Viewing;
    hr = PXCV_DrawPageToDC(pDoc, page_num, dc, &crp);
    return hr;
}


```

PXCV_DrawPageToDIBSection

PXCV_DrawPageToDIBSection creates a Microsoft® Windows® Graphics Device Interface (GDI) DIB section from the specified page (or its rectangular area).

```
HRESULT  PXCV_DrawPageToDIBSection(
      PXVDocument Doc,
      DWORD page_num,
      LPPXV_CommonRederParameters pParams,
      HDC hBaseDC,
      COLORREF backcolor,
      HBITMAP* pResDIBSection,
      HANDLE hSection,
      DWORD dwOffset
);
```

Parameters

Doc
[in] *Doc* specifies a document previously created by the **PXCV_Init** function.

page_num
[in] *page_num* specifies the zero-based page number to be drawn.

pParams
[in] Pointer to the **PXV_CommonRenderParameters** structure, which defines drawing parameters.
Note: Please note that the flag `pxvrpf_UseVectorRenderer` within the `Flags` field of the **PXV_CommonRenderParameters** structure is ignored by this function.

hBaseDC
[in] Handle of a device context which may be used for creation of the DIB section. This parameter may be `NULL`.

backcolor
[in] Specifies created bitmap's background color.
Note: Please note that the most significant byte is used as transparency value. **0** means fully transparent; **255** means no transparency.

pResDIBSection
[out] Pointer to the `HBITMAP` variable that receives the handle to the GDI DIB section.

hSection
[in] Handle of a file-mapping object that the function will use to create the DIB. This parameter may be `NULL`.
For more information regarding this parameter, please see the documentation detailing the Windows® GDI function **CreateDIBSection**.

dwOffset
[in] Specifies the offset from the beginning of the file-mapping object referenced by *hSection* where storage for the bitmap bit values begin. This value is ignored if *hSection* is `NULL`.

Note: The bitmap bit values are aligned on doubleword boundaries, so the offset must be a multiple of the size of a `DWORD`.

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or other value which isn't an error code.

PXCV_FinishReadDocument

PXCV_FinishReadDocument Completes the reading of an encrypted document after **PXCV_ReadDocumentW** returns PS_ERR_DocEncrypted and the correct password was supplied using the **PXCV_CheckPassword** function.

```
HRESULT  PXCV_FinishReadDocument(
      PXVDocument Doc,
      DWORD Flags
);
```

Parameters

Doc
  [in] *Doc* Specifies the PDF object previously created by the function **PXCV_Init**.
Flags
  [in] *Flags* This argument is reserved for future use and should be set to `0`.

Return Values

If the function succeeds, the return value is `DS_OK`.

If the function fails, the return value is error code.

Remarks

This function should be called only after **PXCV_ReadDocumentW** has returned PS_ERR_DocEncrypted and a correct password was supplied using **PXCV_CheckPassword**.
In the case of a successful call to the function **PXCp_ReadDocumentW** there is no need to call the function.

PXCV_GetDocumentInfoW

Function **PXCV_GetDocumentInfoW** retrieves information from the `Info` dictionary for the current PDF document (e.g. this same information would be displayed when using a mouse in Windows Explorer and selecting a file - this information becomes viewable when you 'right click' and select the 'Properties' option for the selected file).

```
HRESULT PXCV_GetDocumentInfoW(
      PXVDocument Doc,
      LPCSTR name,
      LPWSTR value,
      DWORD* valuebuflen
);
```

Parameters

Doc
[in] *Doc* specifies the document that was previously created by the **PXCV_Init** function.

name
[in] Pointer to an `ASCII` string which defines the information key (e.g. `Title`, or `Author`) for the value to be retrieved.

Note: Please note that *name* is case-sensitive and it is an ASCII string, not UNICODE.

value
[in/out] *value* specifies a pointer to a buffer where the information will be placed. If the value of *value* is `NULL`, the required buffer size (in chars) will be placed in *valuebuflen*.

valuebuflen
[in/out] *valuebuflen* specifies an available buffer size in characters (including a null-terminating character). If *value* is `NULL`, the function will insert a `DWORD` variable as pointed to by *valuebuflen* the required buffer size (in chars). If *value* is not `NULL`, the function will write the actual character count and place into a buffer (including a null-terminating character).

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or another value that is not an error code.

See Also

PXCV_Init

Example (C++)

```
LPCWSTR GetAuthor(PXVDocument Doc)
  {
        LPWSTR res = NULL;
        DWORD sz = 0;
        HRESULT hr = PXCV_GetDocumentInfoW(Doc, "Author", res, &sz);
        if (IS_DS_FAILED(hr) || (sz == 0))
            return res;
        res = new WCHAR[sz + 1];
        PXCV_GetDocumentInfoW(Doc, "Author", res, &sz);
        return res;
  }
```

PXCV_GetPageDimensions

PXCV_GetPageDimensions retrieves the dimensions (width and height) of the specified page of the document. This function utilises the page's rotation and crop box when calculating its dimensions. Returned values are in points (1 point is 1/72 inch).

```
HRESULT PXCV_GetPageDimensions(
      PXVDocument Doc,
      DWORD page_num,
      double* width,
      double* height
);
```

Parameters

Doc

[in] *Doc* specifies the document previously created by the **PXCV_Init** function.

page_num

[in] *page_num* specifies the zero-based page number for which dimensions should be retrieved.

width

[out] Pointer to a `double` variable, which receives the width of the page (width of crop box of the page), in points.

height

[out] Pointer to a `double` variable, which receives the height of the page (height of crop box of the page), in points.

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or other value which isn't an error code.

Example (C++)

```
HRESULT GetPageDimInPixels(PXVDocument pDoc, DWORD page_num, DWORD dpi, SIZE* dims)
{
    double pw, ph;
    HRESULT hr = PXCV_GetPageDimensions(pDoc, page_num, &pw, &ph);
    if (IS_DS_SUCCESSFUL(hr))
    {
        dims.cx = (LONG)(pw * dpi / 72.0 + 0.5);
        dims.cy = (LONG)(ph * dpi / 72.0 + 0.5);
    }
    return hr;
}
```

PXCV_GetPageRotation

PXCV_GetPageRotation retrieves the specified rotation angle for the specified page. The angle is always a multiple of 90 degrees.

```
HRESULT  PXCV_GetPageRotation(
      PXVDocument Doc,
      DWORD page_num,
      LONG* angle
);
```

Parameters

Doc

[in] *Doc* specifies the document as previously created by the **PXCV_Init** function.

page_num

[in] *page_num* specifies the zero-based page number whose rotation will be retrieved.

angle

[out] Pointer to the `LONG` variable which receives the rotation angle of the page. Value **0** means that page isn't rotated; **90** - page is rotated clockwise a quarter turn; **180** - page is rotated by 180 degrees (upside-down); and **270** - page is rotated counter-clockwise a quarter turn.

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or other value which isn't an error code.

PXCV_GetPagesCount

PXCV_GetPagesCount retrieves the page count of the specified document.

```
HRESULT  PXCV_GetPagesCount(
      PXVDocument Doc,
      DWORD* count
);
```

Parameters

Doc
[in] *Doc* specifies the document which was previously created by the **PXCV_Init** function.

count
[out] Pointer to a `DWORD` variable into which to return the page count.

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or other value which isn't an error code.

PXCV_GetPermissions

The **PXCV_GetPermissions** function extracts the existing encryption level and user's permissions set for the document.

```
HRESULT  PXCV_GetPermissions(
      PXVDocument Doc,
      DWORD* enclevel,
      DWORD* permFlags
);
```

Parameters

Doc
: [in] *Doc* specifies the document which was previously created by the **PXCV_Init** function.

enclevel
: [out] Specifies a pointer to a variable of the `DWORD` type, which receives encryption level information for the document. Possible values are `40` and `128`.

permFlags
: [out] Specifies a pointer to the variable of the `DWORD` type which receives permission flag information for the document.
For more information about the value of 'bits' used for permissions, please read the `Adobe PDF Specification` (section "Standard Encryption Dictionary", table "User access permissions").

Return Values

If the function fails, the return value is error code.

If the function succeeds, the return value is `DS_OK`, or any other value which is not an error code.

PXCV_Init

PXCV_Init Creates a PDF object, usually required by the majority of functions in the PXCV Library.

```
HRESULT  PXCV_Init(
     PXVDocument* pDoc,
     LPCSTR Key,
     LPCSTR DevCode
);
```

Parameters

pDoc

Pointer to a variable of the type **PXVDocument** that will receive the created PDF object.

Key

[in] Pointer to a null-terminated string which contains your license key for use with . This parameter may be `NULL`; if so, the library will operate in 'evaluation' mode and a demo stamp/watermark will be printed on all output and cannot be removed subsquently.

DevCode

[in] Pointer to a null-terminated string which contains your individual developer code for use with . This parameter may be `NULL`; if so, the library will operate in 'evaluation' mode and a demo stamp/watermark will be printed on all output and cannot be removed subsquently.

Return Values

If the function succeeds, the return value is `DS_OK`, and a variable pointer to *pDoc* will contain the valid PDF object.

If the function fails, the return value is error code.

Example (C++)

```
PXVDocument   hDocument = NULL;
// Please note - RegCode and DevCode are case sensitive
LPCSTR regcode = "<Your serial/keycode code here>";
LPCSTR devcode = "<Your developers' code here>";
HRESULT res = PXCV_Init(&hDocument, regcode, devcode);
if (IS_DS_FAILED(res))
    return res;
...
PXCV_Delete(hDocument);
```

PXCV_ReadDocumentFromIStream

PXCV_ReadDocumentFromIStream reads the PDF document using an `IStream` interface.

```
HRESULT  PXCV_ReadDocumentFromIStream(
      PXVDocument Doc,
      IStream* stream,
      DWORD Flags
);
```

Parameters

Doc

[in] *Doc* Specifies the PDF object previously created by the function **PXCV_Init**.

stream

[in] *stream* specifies a pointer to the `IStream` interface for the stream from which the PDF document is to be loaded.

Flags

[in] *Flags* This argument is reserved for further usage and should be set to `0`.

Return Values

If the function succeeds, the return value is `DS_OK`.

If the function return value is equal to PS_ERR_DocEncrypted, then a password must be provided using **PXCV_CheckPassword** and **PXCV_FinishReadDocument**

If the function fails, the return value is **error code.**

PXCV_ReadDocumentFromMemory

PXCV_ReadDocumentFromMemory reads the document from the specified memory buffer.

```
HRESULT  PXCV_ReadDocumentFromMemory(
      PXVDocument Doc,
      const BYTE* mem,
      UINT size,
      DWORD Flags
);
```

Parameters

Doc
[in] *Doc* specifies the PDF object previously created by the function **PXCV_Init**.

mem
[in] *mem* specifies a pointer to a memory buffer containing PDF document to be opened.

size
[in] Specifies the size in bytes of the buffer pointed to by *mem*.

Flags
[in] This argument is reserved for further usage and should be set to `0`.

Return Values

If the function succeeds, the return value is `DS_OK`.

If the function return value is equal to **PS_ERR_DocEncrypted**, then a password must be provided using **PXCV_CheckPassword** and **PXCV_FinishReadDocument** must be called to complete reading and parsing the document.

If the function fails, the return value is **error code**.

Comments

Memory block passed to the function and should not be released until the function **PXCV_Delete** has been called

PXCV_ReadDocumentW

PXCV_ReadDocumentW reads the document from the specified PDF file.

```
HRESULT  PXCV_ReadDocumentW(
      PXVDocument Doc,
      LPCWSTR pwFileName,
      DWORD Flags
);
```

Parameters

Doc

[in] *Doc* Specifies the PDF object previously created by the function **PXCV_Init**.

pwFileName

[in] *pwFileName* Specifies a pointer to a null-terminated UNICODE string that contains the fully qualified path to the file.

Flags

[in] *Flags* This argument is reserved for further usage and should be set to `0`.

Return Values

If the function succeeds, the return value is `DS_OK`.

If the function return value is equal to **PS_ERR_DocEncrypted**, then a password must be provided using **PXCV_CheckPassword** and **PXCV_FinishReadDocument** must be called to complete reading and parsing the document.

If the function fails, the return value is **error code**.

Example (C++)

```cpp
// Generic example on how to read the document
PXVDocument   hDocument = NULL;
// Please note - RegCode and DevCode are case sensitive
LPCSTR regcode = "<Your serial/keycode code here>";
LPCSTR devcode = "<Your developers' code here>";
HRESULT res = PXCV_Init(&hDocument, regcode, devcode);
if (IS_DS_FAILED(res))
    return res;
hr = PXCV_ReadDocumentW(hDocument, FileName, 0);
  if (IS_DS_FAILED(hr))
  {
      if (hr == PS_ERR_DocEncrypted)
      {
          while (IS_DS_FAILED(hr))
          {
              BYTE*   Password;
              DWORD   PassLen;
              // Obtain password (i.e. showing some dialog)

              // ...

              // Check password
              hr = PXCV_CheckPassword(hDocument, Password, PassLen);
          }
          // Finish read document
          hr = PXCV_FinishReadDocument(hDocument, 0);
          if (IS_DS_FAILED(hr))
          {
          PXCV_Delete(hDocument);
              // In this case document seems to be corrupted
              // ...
          }
      }
      else
      {
          PXCV_Delete(hDocument);
          // In this case document seems to be corrupted
          // ...
      }
  }
  // In this place the document is completely read.
```

PXCV_ReleaseCachedData

PXCV_ReleaseCachedData releases all cached data from specified document. See **Remarks** below.

```
HRESULT  PXCV_ReleaseCachedData(
      PXVDocument Doc,
      DWORD dwFlags
);
```

Parameters

Doc

[in] *Doc* specifies the PDF object previously created by the function **PXCV_Init**.

dwFlags

[in] *dwFlags* specifies which cached content should be freed:

Name	Value	Meaning
pxvrcd_ReleaseDocumentFonts	0x0002	Release embedded fonts - this is the only flag that may be used.

Note: See **PXCV_ReleasePageCachedData** for more information on this parameter.

Return Values

If the function succeeds, the return value is DS_OK.

If the function fails, the return value is error code.

Comments

This function clears all currently cached data for the document, so the next rendering operations will require re-reading and converting of some data. However it may free a significant quantity of used memory, so calling this function is recomended after rendering several pages, and especially in the case where already rendered pages are not expected to be reused..

Remarks

The PDF rasterizer requires significant amounts of memory for such things as: sequences of content rendering operators; fonts used for text rendering that are almost always shared between pages; non-embedded fonts may that be shared between documents; and images which may be shared between pages et al. All of these objects must be converted into internal rasterized representations before being used and this may well be a time-consuming operation.
To accelerate page rendering, especially in the case when several parts of same page are rendered sequentially, the rasterizer keeps all objects as internal representations, so that in subsequent rendering operations some objects will not require repeated conversion. But some objects require a lot of memory; for example, a "simple" text page may contain several thousand rendering operators, so it may become necessary to free some or all cached objects to free used memory. To do this the provides two functions: **PXCV_ReleaseCachedData** and **PXCV_ReleasePageCachedData**.

PXCV_ReleasePageCachedData

The Function **PXCV_ReleasePageCachedData** releases page-specific cached data for one page in a document and global/shared resources used there-on (optional).
See **Remarks** below for details.

```
HRESULT  PXCV_ReleasePageCachedData(
        PXVDocument Doc,
        DWORD page_num,
        DWORD dwFlags
);
```

Parameters

Doc
[in] *Doc* specifies the PDF object previously created by the function **PXCV_Init**.

page_num
[in] *page_num* specifies the zero-based page number for which cached data should be released.

dwFlags
[in] *dwFlags* specifies which cached content should be freed. May be any combination of following flags.

Name	Value	Meaning
`pxvrcd_ReleaseDocumentImages`	`0x0001`	Release used images
`pxvrcd_ReleaseDocumentFonts`	`0x0002`	Release used embedded fonts
`pxvrcd_ReleaseGlobalFonts`	`0x0004`	Release used global (not embedded) fonts

Return Values

If the function succeeds, the return value is `DS_OK`.

If the function fails, the return value is error code.

Comments

This function does not release all cached data in a document, but only data used to render a specified page. If *dwFlags* is zero (`0`) then only the content-rendering operators for this page will be released. Using this function is recomended after rendering a page is complete and if it is unlikely that it would be necessary render this page again soon. Further we also recomended calling **PXCV_ReleaseCachedData** using the `pxvrcd_ReleaseDocumentImages` flag, as in most cases images are not shared between adjacent pages.

Remarks

The PDF rasterizer requires significant amounts of memory for such things as: sequences of content rendering operators; fonts used for text rendering that are almost always shared between pages; non-embedded fonts may that be shared between documents; and images which may be shared between pages et al. All of these objects must be converted into internal rasterized representations before being used and this may well be a time-consuming operation.

To accelerate page rendering, especially in the case when several parts of same page are rendered sequentially, the rasterizer keeps all objects as internal representations, so that in subsequent rendering operations some objects will not require repeated conversion. But some objects require a lot of memory; for example, a "simple" text page may contain several thousand rendering operators, so it may become necessary to free some or all cached objects to free used memory. To do this the provides two functions: **PXCV_ReleaseCachedData** and **PXCV_ReleasePageCachedData**.

PXCV_SetCallBack

PXCV_SetCallBack sets the callback function to be used during the PDF rasterization process.

```
HRESULT  PXCV_SetCallBack(
     PXVDocument Doc,
     PXV36_CALLBACK_FUNC pProc,
     LPARAM UserData
);
```

Parameters

Doc

[in] *Doc* specifies the PDF object previously created by the function **PXCV_Init**.

pProc

[in] *pProc* specifies the callback function, which must be defined as:

```
typedef BOOL (__stdcall *PXV36_CALLBACK_FUNC) (DWORD dwStage, DWORD
dwLevel, LPARAM param);
```

The first parameter of this function indicates the callback state; the second indicates the progress level (see table below), and the third will always have the same value as passed in *UserData*.

Callback function's state constants table

Constant	Value	Meaning of level
`PXCVClb_Start`	**1**	`MaxVal` - maximum value of the level which will be passed
`PXCVClb_Processing`	**2**	Current progress level - any value from `0` to `MaxVal`
`PXCVClb_Finish`	**3**	May be any value from `0` to `MaxVal` (`MaxVal` if all passed), may be ignored

Note: The Callback function should return `TRUE` (any non-zero value) to continue processing or `FALSE` (zero) to abort the operation.

UserData

[in] *UserData* specifies a user-defined callback parameter to be passed as a third parameter to the function specified by *pProc*.

Return Values

If the function succeeds, the return value is `DS_OK`.
If the function fails, the return value is error code.

PXV_CommonRenderParameters

The **PXV_CommonRenderParameters** structure defines drawing parameters for the functions **PXCV_DrawPageToDC** and **PXCV_DrawPageToDIBSection**.

```
typedef struct _PXV_CommonRenderParameters {
    LPRECT WholePageRect;
    LPRECT DrawRect;
    DWORD Flags;
    DWORD RenderTarget;
} PXV_CommonRenderParameters;
```

Members

WholePageRect

Specifies the rectangular area within the Windows® target device's Device Context (DC) coordinate system where the entire PDF page's rectangle would be drawn. See **Comments** for more details.

DrawRect

Specifies the rectangular portion of the PDF page to be drawn. If this field is `NULL`, then the entire PDF page will be drawn.

Flags

This `DWORD`value is a combination of flags, which defines rendering options, such as rotation and vector rendering. May be combination of the following values.

Flags	Value	Meaning
`pxvrpf_Rotate_NoRotate`	0x0000	Draws the page without any additional rotation.
`pxvrpf_Rotate_Rotate90CW`	0x0001	Draws the page with a 90 degree clockwise rotation (i.e. landscape).
`pxvrpf_Rotate_Rotate180`	0x0002	Draws the page with a 180 degree rotation.
`pxvrpf_Rotate_Rotate90CCW`	0x0003	Draws the page with a 90 degree counter clockwise rotation.
`pxvrpf_UseVectorRenderer`	0x0004	When this flag is specified, vector rendering, using Microsoft© GDI+®, is used (in place of raster rendering). This feature is reccomended when possible for printing as the print job and therefore resources used are considerably smaller.
`pxvrpf_RenderAsGray`	0x0008	When this flag is specified, rendering will be performed in grayscale mode, in place of color.
pxvrpf_EmbeddedFontAsCurves	0x0010	This flag has meaning only when `pxvrpf_UseVectorRenderer` used, otherwise it is ignored. When this flag is specified, all embedded fonts are rendered as curves, not using text functions. When the flag isn't specified, embedded true-type fonts are installed temporary into system for rendering.
pxvrpf_AllFontsAsCuves	0x0030	This flag has meaning only when `pxvrpf_UseVectorRenderer` used, otherwise it is ignored. When it is used, all fonts are rendered as curves without using text output API functions.

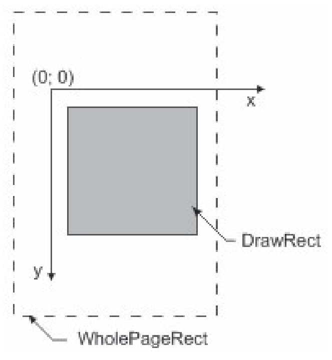
pxvrpf_NoTransparentBkgnd	0x0040	This flag has meaning only when `pxvrpf_UseVectorRenderer` is not used. When this flag is specified, raster image is filled by white color without transparency before drawing. When flag isn't specified, drawings are provided on transparent background.

RenderTarget

Specifies the rendering mode to be used. This is meaningful if the optional content exists within the document which is visible only in some rendering modes, such as Push buttons within an Adobe Acroform. May be any one of the following values:

Constant	Value	Meaning
`pxvrm_Viewing`	0	Rendering target is **View**. For example, this mode is used for displaying a document on screen.
`pxvrm_Printing`	1	Rendering target is **Print**. This mode is used for printing a document (or for a print preview).
`pxvrm_Exporting`	2	Rendering target is **Export**. This mode is used for exporting document content to another format (e.g any one of the supported raster image formats).

Comments



To specify only a part of the PDF page is to be drawn, it is necessary to specify the rectangular area of the target's DC as `WholePageRect` which the entire PDF page would occupy, and `DrawRect`, which defines the part of the PDF page which should be drawn within `WholePageRect`. If `DrawRect` is set to `NULL`, then the entire PDF page will be drawn within the target device's `WholePageRect` area. This simplifies scaling of the PDF page (zoom level), and helps prevent rounding errors during the "points to pixels" conversion that must take place in order to render the PDF information onto the target device.

Example 1

We want to draw a PDF page within our application window. For example:
1. The PDF page has the dimensions 576 x 792 points (8 x 11 inches).
2. The desired "zoom level" is 400%, or a scaling factor of 4.
3. Our application window's DC has of dimensions 600 x 800 pixels, with a DPI of 96.
4. Our application window has scroll bars to control the page display, and their positions are: 120 for the vertical and 180 for the horizontal, assuming that the maximum position for the horizontal scroll bar is the page's width in pixels, less the window width.

Therefore if is first necessary to calculate the PDF page's dimensions in pixels. This is given by:

```
page_width_in_pixels = (576 / 72) * 96 * 4 = 3072 pixels;

page_height_in_pixels = (792 / 72) * 96 * 4 = 4224 pixels;
```

As long as the PDF page's dimensions and zoom level remain constant, these values will remain constant.

Now `WholePageRect` and `DrawRect` values for `PXV_CommonRenderParameters` structure are:

```
WholePageRect.left = -180; // horizontal scroll position
WholePageRect.top = -120; // vertical scroll position
WholePageRect.right = WholePageRect.left + page_width_in_pixels;
WholePageRect.bottom = WholePageRect.top + page_height_in_pixels;

DrawRect.left = 0;
DrawRect.top = 0;
DrawRect.right = 600; // our window width
DrawRect.top = 800; // our window height
```

For any constant zoom level, `WholePageRect` depends only on the scroll bars' positions. This minimizes calculations (and rounding errors).

Example 2
We have a PDF page with width 576 points (8 inches) and height 792 points (11 inches). And we want to draw part of the page as defined by: left = 144pt; top = 288pt; right = 360pt; bottom = 648pt starting from point (10, 10) onto our target DC with a zoom factor of 200%, or a scaling factor of 2. Our DC has a DPI of 96, as do most screens in Windows.

In pixels, the width and height of the page will be:

```
width = (576 / 72) * 96 * (200 / 100) = 1536 pixels;
height = 2112 pixels.
```

Therefore the width of the required portion of the page is:

```
part_width = ((360 - 144) / 72) * 96 * (200 / 100) = 576 pixels;
part_height = 960 pixels.
```

Left-top point of the drawn area we need to locate as at the coordinates (10, 10) onto our DC. So, the left upper point of the complete page will have coordinates:

```
Page_Origin_X = 10 - (144 / 72) * 96 * 200 / 100 = -374;
Page_Origin_Y = 10 - (288 / 72) * 96 * 200 / 100 = -758;
```

The required values are therefore: WholePageRect = {-374, -758, -354 + 1536, -758 + 2112}; and DrawRect = {10, 10, 10 + 576, 10 + 960}.

test

Apply all cached settings that were passed with the PXCVA_Flags::PXCVA_NoApply flag.

Syntax

```
HRESULT ApplyAllCachedChanges(VOID);
```

Return Value

Returns **S_OK** if successful, or an error value otherwise. To obtain the text description of a received error code, use GetTextFromResult.

Related Topics

PXCVA_Flags::PXCVA_NoApply,

PDFXCview::DiscardAllCachedChanges,

Operations::ApplyAllCachedChanges

Error handling

Error handling

» **Error codes**
» **PXCV_Err_FormatFacility**
» **PXCV_Err_FormatSeverity**
» **PXCV_Err_FormatErrorCode**

Error Codes

Most functions return an `HRESULT` value which provides a simple means to determine the success or otherwise of a function call.

If the most significant bit or result is set to 1 then the specified error occurred, otherwise the function was successful. Here are two simple macro's for C/C++ which apply these checks:

```
#define IS_DS_SUCCESSFUL(x)      (((x) & 0x80000000) == 0)
#define IS_DS_FAILED(x)          (((x) & 0x80000000) != 0)
```

Note:

It is strongly recommended to always use the specified (or equivalent macro's) to establish if the function call was successful or otherwise. A simple comparison with `0` (`zero`) will usually provide eroneous and unreliable results described in the following example scenario's.

A Function may return a warning with a code that is not equal to `zero` (and also not negative!). This usually means that the function has succeeded and is providing additional information about the call. i.e. The function returns a default value etc. For more information see the description provided for each particular function.
To determine if the return value is generating a warning we provide the `IS_DS_WARNING` macro's. This would be the correct syntax to check for the error status of the **PXCV_CheckPassword** function:

```
HRESULT hr = PXCV_CheckPassword(doc, password, len);

    if (IS_DS_FAILED(hr))
    {
        // An error occurred!
        // Manage the error accordingly to provide an orderly exit from the
function call
        ...
    }
    else
    {
        // 'hr' contains value which indicate which password was supplied - owner
or user
        ...
    }
```

The example code below demonstrates how NOT to provide error checking in your code:

```
HRESULT hr = PXCV_CheckPassword(doc, password, len);

    if (hr == 0)
    {
        // treat as success
        ...

        (this is not true as a positive return value was received!)
        ...
    }
    else
    {
        // treat as error
        (Incorrect as the retrun value has not been adequately identified and this
is unreliable!)
        ...
    }
```

Most frequently returned error codes are listed in the table below, however functions may return additional codes which are not listed here. There are 3 further functions available for dealing with error's and may give additional information relating to a specific error: **PXCV_Err_FormatSeverity**, **PXCV_Err_FormatFacility**, **PXCV_Err_FormatErrorCode**. A code example of how to use this table is provided below the table itself. Please note that this function will provide information about **all** error codes which may be returned.

Possible values of errors of `PDF parser/structure`:

Constant	Value	Description
PS_ERR_NOTIMPLEMENTED	0x820f04b0	Not implemented
PS_ERR_INVALID_ARG	0x820f0001	Invalid argument
PS_ERR_MEMALLOC	0x820f03e8	Insufficient memory
PS_ERR_USER_BREAK	0x820f01f4	Operation aborted by user
PS_ERR_INTERNAL	0x820f0011	Internal error
PS_ERR_INVALID_FILE_FORMAT	0x820f0002	Invalid file format
PS_ERR_REQUIRED_PROP_NOT_SET	0x820f2716	Required property is not set
PS_ERR_INVALID_PROP_TYPE	0x820f2717	Invalid property type
PS_ERR_INVALID_PROP_VALUE	0x820f2718	Invalid property value
PS_ERR_INVALID_OBJECT_NUM	0x820f2719	Invalid object number
PS_ERR_INVALID_PS_OPERATOR	0x820f271c	Invalid PS operator
PS_ERR_UNKNOWN_OPERATOR	0x820f2787	Unknown operator
PS_ERR_INVALID_CONTENT_STATE	0x820f2788	Invalid content state
PS_ERR_NoPassword	0x820f27a8	No password
PS_ERR_UnknowCryptFlt	0x820f27a9	Unknown crypt filter
PS_ERR_WrongPassword	0x820f27aa	Wrong password
PS_ERR_InvlaidObjStruct	0x820f27ab	Invalid object structure

PS_ERR_WrongEncryptDict	0x820f27ac	Invalid encryption dictionary
PS_ERR_DocEncrypted	0x820f27ad	Document encrypted
PS_ERR_DocNOTEncrypted	0x820f27ae	Document not encrypted
PS_ERR_WrongObjStream	0x820f27af	Invalid object stream
PS_ERR_WrongTrailer	0x820f27b0	Invalid document trailer
PS_ERR_WrongXRef	0x820f27b1	Invalid xref table
PS_ERR_WrongDecodeParms	0x820f27b2	Invalid decode parameter(s)
PS_ERR_XRefNotFounded	0x820f27b3	xref table is not foud
PS_ERR_DocAlreadyRead	0x820f27b4	Document is already read
PS_ERR_DocNotRead	0x820f27b5	Document is not read

Comments

There is an additional utility included with this library which provides valuable additional data regarding all known error codes - **DSErrorLookUp.exe**. This can be found in your PDF-XChange/Tools installation folders and is extremly useful during your application development process - we strongly reccomend ALL developers utilise **DSErrorLookUp.exe** during the debugging of their applications and prior to support requests relating to Error Code return values and their meaning.

Example (C++)

```cpp
// Using of PXCV_Err_FormatSeverity, PXCV_Err_FormatFacility, PXCV_Err_FormatErrorCode functions
char* err_message = NULL;
char* buf = NULL;
_PXCPage* p = NULL;
    // Code below should always return an error and never work
HRESULT dummyError = PXCV_ReadDocumentW(NULL, NULL, 0);
LONG sevLen = PXCV_Err_FormatSeverity(dummyError, NULL, 0);
LONG facLen = PXCV_Err_FormatFacility(dummyError, NULL, 0);
LONG descLen = PXCV_Err_FormatErrorCode(dummyError, NULL, 0);
if ((sevLen > 0) && (facLen > 0) && (descLen > 0))
{
    // Total length of the formated text is the sum of the length for each description
    // plus some additional characters for formating
    LONG total = sevLen + facLen + descLen + 128;
    // allocate buffer for message
    err_message = new char[total];
    err_message[0] = '\0';
    // allocate temporary buffer
    buf = new char[total];
    // get error severity and append to message
    if (PXCV_Err_FormatSeverity(dummyError, buf, total) > 0)
        lstrcat(err_message, buf);
    lstrcat(err_message, " [");
    // get error facility and append to message
    if (PXCV_Err_FormatFacility(dummyError, buf, total) > 0)
        lstrcat(err_message, buf);
    lstrcat(err_message, "]: ");
    // and error code description and append to message
    if (PXCV_Err_FormatErrorCode(dummyError, buf, total) > 0)
        lstrcat(err_message, buf);
    ::MessageBox(NULL, err_message, "Test error", MB_OK);
    delete[] buf;
    delete[] err_message;
}
```

PXCV_Err_FormatFacility

PXCV_Err_FormatFacility returns information of where an error occurred for the specified error code.

```
LONG  PXCV_Err_FormatFacility(
      HRESULT errorcode,
      LPSTR buf,
      LONG maxlen
);
```

Parameters

errorcode
[in] *errorcode* specifies the `HRESULT` returned by a function.

buf
[out] *buf* specifies a pointer to a buffer where the error facility information will be returned.

Note: To determine the required buffer size you should pass `NULL` for *buf*.

maxlen
[in] *maxlen* specifies the available buffer size in characters (including null-terminating character).

Return Values

If the function fails to recognize an error code the return value is negative.

If the function fails to retrieve information about an error code the return value is zero.

If the function successfully retrieves information and the parameter *buf* is `NULL` the return value is the number of characters required to store the description (including null-terminating character).

If the function successfully retrieves information and the parameter *buf* is not `NULL` the return value is the number of characters written to the buffer (including null-terminating character).

PXCV_Err_FormatSeverity

PXCV_Err_FormatSeverity returns information regarding the error severity. See Error codes for additional information.

```
LONG   PXCV_Err_FormatSeverity(
        HRESULT errorcode,
        LPSTR buf,
        LONG maxlen
);
```

Parameters

errorcode
[in] *errorcode* specifies an `HRESULT` returned by a library function.

buf
[out] *buf* specifies a pointer to a buffer where the error severity will be returned.

Note: To determine the required buffer size you should pass `NULL` as *buf*.

maxlen
[in] *maxlen* specifies the available buffer size in characters (including a null-terminating character).

Return Values

If the function fails to recognize an error code the return value is negative.

If the function fails to retrieve information regarding an error code the return value is zero.

If the function successfully retrieves information and the parameter *buf* is `NULL` the return value is the number of characters required to store the description (including a null-terminating character).

If the function successfully retrieves information and the parameter *buf* is not `NULL` the return value is the number of characters written to the buffer (including a null-terminating character).

PXCV_Err_FormatErrorCode

PXCV_Err_FormatErrorCode provides information relating to an error code.

```
LONG  PXCV_Err_FormatErrorCode(
      HRESULT errorcode,
      LPSTR buf,
      LONG maxlen
);
```

Parameters

errorcode

[in] *errorcode* specifies the `HRESULT` returned by a library function.

buf

[out] *buf* specifies a pointer to a buffer where the error description will be returned.

Note: To determine the required buffer size you should pass `NULL` as *buf*.

maxlen

[in] *maxlen* specifies an available buffer size in characters (including a null-terminating character).

Return Values

If the function fails to recognize an error code the return value is negative.

If the function fails to find information regarding the error code the return value is zero.

If the function successfully retrieves information and the parameter *buf* is `NULL` the return value is the number of characters required to store the description (including a null-terminating character).

If the function successfully retrieves information and the parameter *buf* is not `NULL` the return value is the number of characters written to a buffer (including a null-terminating character).

Index

E

Error Codes 31
Error Handling 31

F

Functions 4

G

Getting Started 2

P

PXCV_CheckPassword 5
PXCV_Delete 6
PXCV_DrawPageToDC 7
PXCV_DrawPageToDIBSection 9
PXCV_Err_FormatErrorCode 37
PXCV_Err_FormatFacility 35
PXCV_Err_FormatSeverity 36
PXCV_FinishReadDocument 11
PXCV_GetDocumentInfoW 12
PXCV_GetPageDimensions 14
PXCV_GetPageRotation 15
PXCV_GetPagesCount 16
PXCV_GetPermissions 17
PXCV_Init 18
PXCV_ReadDocumentFromIStream 19
PXCV_ReadDocumentFromMemory 20
PXCV_ReadDocumentW 21
PXCV_ReleaseCachedData 23
PXCV_ReleasePageCachedData 24
PXCV_SetCallBack 26
PXV_CommonRenderParameters 26

R

Redistribution 3

S

System Requirements 3

W

Welcome Page 1