# Table of Contents

## 1. Welcome

1.1 Introduction .................................................................................................................... 6

1.2 Installation .......................................................................................................................... 8

1.2.1 System Requirements .................................................................................................. 9

## 2. Functions

2.1 PXCV_CheckPassword ........................................................................................................ 11

2.2 PXCV_Delete .................................................................................................................... 12

2.3 PXCV_DrawPageToDC ...................................................................................................... 13

2.4 PXCV_DrawPageToDIBSection ......................................................................................... 15

2.5 PXCV_DrawPageToIStream .............................................................................................. 17

2.6 PXCV_FinishReadDocument .......................................................................................... 19

2.7 PXCV_GetDocumentInfoW ............................................................................................... 20

2.8 PXCV_GetPageDimensions ............................................................................................ 22

2.9 PXCV_GetPageRotation .................................................................................................. 24

2.10 PXCV_GetPagesCount .................................................................................................... 26

2.11 PXCV_GetPermissions .................................................................................................. 27

2.12 PXCV_Init ...................................................................................................................... 28

2.13 PXCV_ReadDocumentFromIStream ................................................................................ 30

2.14 PXCV_ReadDocumentFromMemory ................................................................................. 31

2.15 PXCV_ReadDocumentW .................................................................................................. 33

2.16 PXCV_ReleaseCachedData ............................................................................................ 35

2.17 PXCV_ReleasePageCachedData ....................................................................................... 37

2.18 PXCV_SetCallBack ........................................................................................................ 39

2.19 PXV_CommonRenderParameters ..................................................................................... 41

2.20 PXV_DrawToImageParams ............................................................................................. 47

## 3. Error Handling

3.1 Error Codes ..................................................................................................................... 50

3.2 PXCV_Err_FormatFacility ............................................................................................. 55

3.3 PXCV_Err_FormatSeverity ............................................................................................ 57

3.4 PXCV_Err_FormatErrorCode .......................................................................................... 59

Index ........................................................................................................................................ 63
Welcome to the PDF-XChange Editor Simple SDK V6 user manual. Use the Table of Contents on the left to browse the topics of this manual. Click topics to expand them.

We offer several further cutting-edge applications for the manipulation of PDF and image files. See the Tracker Software Products Page for more information.

If you have any queries then please Contact Us. We aim to respond to all communication within eight hours - and we are usually much faster.

This manual is broken down into the following sections:

- The Welcome section contains Installation and System Requirements information.
- The Functions section details all functions that the PDF-XChange Editor Simple SDK can be used to achieve.
- The Error Handling section details all error codes that may appear within PDF-XChange Editor Simple SDK.

A PDF version of this manual is available here.

Note that the icons in the upper left of the screen can be used to browse/search the manual:
Welcome

Click the Table of Contents to view/move to chapters of the manual.

Click the Keyword Index to view keywords, and click keywords to move to their location in the manual.

Click Search Topics to enter custom search terms.
1.1 Introduction

The PDF-XChange Editor Simple SDK V6 installs via DLL and enables the basic viewing/printing of PDF documents within developer applications. Developers who require the use of all PDF-XChange Editor features within their applications should purchase the full PDF-XChange Editor SDK, which is available at the link below.

Additional developer kits are available from Tracker Software Products to create, view, edit and manipulate PDF documents. They include:

- The PDF-XChange Core API SDK, which provides developers with libraries and API for the creation and manipulation of fully-native, industry standard PDF files.
- The PDF-XChange Editor SDK, which enables the incorporation of a fully-licensed version of PDF-XChange Editor into developers’ applications.
- The PDF-XChange PRO SDK bundle, which combines the features of the PDF-XChange Drivers API SDK and the PDF-XChange Core API SDK. This SDK also features a programmatic OCR module and access to its library DLL functions - including the creation of programmatic, fillable forms and digital signature capabilities.

See here for further information about our developer products and a comprehensive list of available applications.

Please note:

- The PDF-XChange Editor Simple SDK is not a royalty-free tool kit. A fixed amount of licenses are included in each SDK product. Additional bulk-license packs are available - see here for further information.
- This toolkit must not be used to develop toolkits/components for non-licensed developers.
- The license agreement contains all relevant terms and conditions of use. If there is any uncertainty about whether the intended use would be in breach of license then please Contact Us for clarification. Our terms of licensing are flexible and we can often tailor them to meet the specific needs of our clientele.
- We recommend experimenting with the evaluation version prior to purchase. Evaluation versions are fully functional, but watermarks will be included on output pages. This means that
applications can be fully developed before a purchase is made. When a licensed version is purchased all output becomes watermark-free. Our hope is that this can guarantee satisfaction as we cannot offer refunds once a purchase has taken place.

**Licensing**

Please use [PXCV_Init](28) to add your registration key to the product.

**Support**

- Our [Developer Forums](32) are an excellent resource for troubleshooting.
- The [Adobe Website](33) is a very useful resource for developers working on PDF-related applications.
- [Tracker Software Products](35) also provides SDKs for the creation/manipulation of PDF and raster image files, as well as an SDK for virtual printer driver functionality. See [here](37) for further information.
1.2 Installation

Click [here](#) to download the **PDF-XChange Editor Simple SDK V6**. The installation folder contains the following items:

- A PDF version of the **PDF-XChange Editor Simple SDK** manual.
- A PDF version of the **PDF-XChange Editor Simple SDK** user license.
- An examples folder of the **PDF-XChange Editor Simple SDK** utilized in the following programming languages:
  - C#
  - C/C++
  - Delphi
  - Visual Basic
  - Visual Basic.NET

**Redistribution**

The **PDF-XChange Editor Simple SDK** depends on only the **pxcview.dll**. Additional **SDK** components from Tracker Software applications are not required. However, the **Microsoft© GDI+** must be installed on the operating system of the local machine in order to enable vector printing. The **Microsoft© GDI+** is installed by default on all version of windows from **Windows XP**. Therefore, if an earlier version is being used, then **Microsoft© GDI+** must be installed. Additionally, **PDF-XChange Editor** is compatible with only **Windows 2000** and later - earlier versions are not supported.
1.2.1 System Requirements

The **PDF-XChange Editor Simple SDK** supports all **Windows** (32/64 bit) operating systems from **Windows XP** and later:

![Supported Operating Systems](image)

**Figure 1. Supported Operating Systems**

Please note the following:

- We recommended that users install the latest **Microsoft Windows** service packs and updates before using our products, as doing so will ensure the greatest possible performance of the software.
- Our products - in particular the printer drivers that **PDF-XChange Standard** and **PDF-XChange Lite** utilize - are not designed to work in virtualized environments such as the **XenApp** software.
- *There are some limitations to product support for **Windows XP** as **Microsoft** have stopped supporting it. Further information about this issue is available [here](#).
The following functions are available in the PDF-XChange Editor Simple SDK:

- **PXCV_CheckPassword** validates the supplied password against the current document.
- **PXCV_Delete** releases the PDF object that **PXCV_Init** created.
- **PXCV_DrawPageToDC** draws specified pages to a device context.
- **PXCV_DrawPageToDIBSection** creates a Windows graphics device interface DIB section from document pages.
- **PXCV_DrawPageToIStream** renders the specified page and saves it to the stream object.
- **PXCV_FinishReadDocument** completes the reading of encrypted documents when **PXCV_ReadDocumentW** returns **PS_ERR_DocEncrypted** and **PXCV_CheckPassword** supplied the correct password.
- **PXCV_GetDocumentInfoW** retrieves information from the info dictionary of documents.
- **PXCV_GetPageDimensions** retrieves page dimensions.
- **PXCV_GetPageRotation** retrieves the rotation angle of pages.
- **PXCV_GetPagesCount** retrieves the page count of documents.
- **PXCV_GetPermissions** extracts the encryption level and permission settings of documents.
- **PXCV_Init** creates a PDF object.
- **PXCV_ReadDocumentFromIStream** uses an Istream interface to read documents.
- **PXCV_ReadDocumentFromMemory** reads documents from memory buffers.
- **PXCV_ReadDocumentW** reads documents from specified PDF files.
- **PXCV_ReleaseCachedData** releases cached document data.
- **PXCV_ReleasePageCachedData** releases cached data for individual document pages.
- **PXCV_SetCallBack** sets the callback function used during the process of PDF rasterization.
- **PXV_CommonRenderParameters** defines drawing parameters for the functions **PXCV_DrawPageToDC** and **PXCV_DrawPageToDIBSection**.

Please note that all functions and their parameters are case-sensitive.
2.1 PXCV_CheckPassword

**PXCV_CheckPassword** validates the supplied password against the current document, and should be called when **PXCV_ReadDocumentW** returns **PS_ERR_DocEncrypted**.

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

**Parameters**

**Doc**

[in] Specifies a document that **PXCV_Init** created.

**pPassword**

[in] Specifies a pointer to a buffer that contains password data. (Buffers may contain zero '"0' symbols if desired).

**PassLen**

[in] Specifies the buffer length.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is 1 in the case of user passwords and 2 in the case of owner passwords.

If the function fails then the return value is an **error code**.
2.2 PXCV_Delete

**PXCV_Delete** releases the PDF object that **PXCV_Init** created, and must be called once the object is no longer required/all updates are complete.

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

**Parameters**

- **Doc**
  
  [in] Specifies a document that **PXCV_Init** created.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.
2.3 PXCV_DrawPageToDC

**PXCV_DrawPageToDC** draws specified pages to a device context.

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

**Parameters**

*Doc*

[in] Specifies a document that **PXCV_Init** created.

*page_num*

[in] Specifies the zero-based page number to be drawn.

*hDC*

[in] Specifies the handle of the device context onto which page information is drawn.

*pParams*

[in] Pointer to the **PXV_CommonRenderParameters** structure that defines drawing parameters.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function fails then the return value is an *error code*. If the function succeeds then the return value is DS_OK or a different value that isn't an error code.
Functions

Example (C++)

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;
}
2.4 PXCV_DrawPageToDIBSection

**PXCV_DrawPageToDIBSection** creates a Windows graphics device interface DIB section from document pages.

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

**Parameters**

**Doc**

[in] Specifies a document that **PXCV_Init** created.

**page_num**

[in] Specifies the zero-based page number to be drawn.

**pParams**

[in] Pointer to the **PXV_CommonRenderParameters** structure, which defines drawing parameters. Please note that this function ignores the **pxvrpf_UseVectorRenderer** flag contained in the **Flags** field of **PXV_CommonRenderParameters**.

**hBaseDC**

[in] Handle of the device context used to create DIB sections. This parameter can be set to NULL if desired.

**backcolor**

[in] Specifies the background color. The most significant byte is used as the transparency value. 0 is full transparency and 255 is no transparency.
**pResDIBSection**

[out] Pointer to the **HBITMAP** variable that receives the DIB section handle.

**hSection**

[in] Handle of the file-mapping object used to create the DIB section. This parameter can be set to NULL if desired. See **CreateDIBSection** for further information about this parameter.

**dwOffset**

[in] Specifies the offset from the beginning of the object that **hSection** references to where storage of the bitmap bit values begin. This value is ignored if **hSection** is NULL. Please note that the bitmap bit values are aligned on doubleword boundaries, therefore the offset must be a multiple of the size of a DWORD.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function fails then the return value is an **error code**.

If the function succeeds then the return value is DS_OK, or a different value that isn't an error code.
2.5 PXCV_DrawPageToIStream

**PXCV_DrawPageToIStream** renders the specified page and saves it to the stream object.

```c
HRESULT PXCV_DrawPageToIStream(
    PXVDocument Doc,
    DWORD page_num,
    LPPXV_CommonRenderParameters pParams,
    COLORREF backcolor,
    LPPXV_DrawToImageParams pImageParams,
    IStream* pDest
);
```

**Parameters**

*Doc*

[in] Specifies a document that **PXCV_Init** created.

*page_num*

[in] Specifies the zero-based page number to be rendered.

*pParams*

[in] Pointer to the **PXV_CommonRenderParameters** structure, which defines drawing parameters.

*backcolor*

[in] Specifies the color used to fill the background before pages are rendered. The most significant byte is used as the transparency value. 0 is full transparency and 255 is no transparency.

*pImageParams*

[in] Pointer to the **PXV_DrawToImageParams** structure, which defines the parameters of generated image files.

*pDest*
Functions

[in] Pointer to the IStream object where images that this function creates are stored.

Please note that all functions and parameters are case-sensitive.

Return Values

If the function fails then the return value is an error code.

If the function succeeds then the return value is DS_OK, or a different value that isn't an error code.

Example (C++)

```cpp
HRESULT DrawPageToPNG(PXVDocument pDoc, DWORD page_num, LPCWSTR sFileName)
{
    HRESULT hr = S_OK;

    RECT rect;
    PXV_CommonRenderParameters crp = {0};  // common render parameters - used to specify which part of the page and into with which size should rasterised (in our example whole page into 600x800 rect) // also defines different render options
    PXV_DrawToImageParams dip = {0};       // specify resulting image parameters, like format, bits per pixel

    IStream* pImage = NULL;                // in real app it will be the stream where image should be stored

    // lets create a stream on file where our rendered image will be stored
    hr = SHCreateStreamOnFile(sFileName, STGM_CREATE | STGM_READWRITE, &pImage);  // lets specify parameters for rendering
    rect.left = 0; rect.top = 0;
    rect.right = 600; rect.bottom = 800;
    crp.WholePageRect = &rect;
    crp.DrawRect = &rect;
    crp.Flags = pxvrpf_EmbeddedFontAsCurves | pxvrpf_NoTransparentBkgnd;
    crp.RenderTarget = pxvrm_Exporting;
    dip.ImageFormat = IMGF_PNG;
    dip.Flags = 0; // reserved
    dip.Bpp = 24;

    // rendering
    hr = PXCV_DrawPageToIStream(pDoc, 0, &crp, RGB(255, 255, 255) | 0xFF000000, &dip, pImage);

    pImage->Release();
    return hr;
}
```
2.6 PXCV_FinishReadDocument


HRESULT PXCV_FinishReadDocument(
    PXVDocument Doc,
    DWORD Flags
);

Parameters

Doc
    [in] Specifies a document that PXCV_Init[26] created.

Flags
    [in] This parameter is reserved for future use and should be set to 0.

Please note that all functions and parameters are case-sensitive.

Return Values

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.[50]

Comments

This function should be called only when PXCV_ReadDocumentW[33] returns PS_ERR_DocEncrypted[55] and PXCV_CheckPassword[15] supplied the correct password. If calls to PXCV_ReadDocumentW[33] are successful then there is no need to call this function.
### 2.7 PXCV_GetDocumentInfoW

**PXCV_GetDocumentInfoW** retrieves information from the info dictionary of documents. (The information retrieved is the same as that displayed when files are right-clicked and the **Properties** option is selected).

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

**Parameters**

- **Doc**
  - [in] Specifies a document that **PXCV_Init** created.

- **name**
  - [in] Pointer to an ASCII string that defines the information key (Title, Author, Subject etc) for the value to be retrieved. Please note that this parameter is case-sensitive and incompatible with UNICODE.

- **value**
  - [in/out] Specifies a pointer to a buffer where retrieved information is placed. If this parameter is set to NULL then the required buffer size will be placed in **valuebuflen**. Buffer sizes are given in characters.

- **valuebuflen**
  - [in/out] Specifies an available buffer size. Buffer sizes are given in characters, and a null-terminating character is included. If **value** is NULL then **valuebuflen** will write the required buffer size in characters. If **value** is not NULL then **valuebuflen** will write the character count, including a null-terminating character, and place it into a buffer.

Please note that all functions and parameters are case-sensitive.
**Return Values**

If the function fails then the return value is an error code.

If the function succeeds then the return value is DS_OK, or a different value that is not an error code.

**Example (C++)**

```cpp
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;
}
```
2.8 PXCV_GetPageDimensions

PXCV_GetPageDimensions retrieves page dimensions in points.

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

Parameters

Doc
    [in] Specifies a document that PXCV_Init created.

page_num
    [in] Specifies the zero-based page number for which to retrieve dimensions.

width
    [out] Pointer to a double variable that receives the width of the page. The measurement returned is the width (in points) of the crop box.

height
    [out] Pointer to a double variable that receives the height of the page. The measurement returned is the height (in points) of the crop box.

Please note that all functions and parameters are case-sensitive.

Return Values

If the function fails then the return value is an error code.

If the function succeeds then the return value is DS_OK, or a different value that 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;
}
2.9 PXCV_GetPageRotation

PXCV_GetPageRotation retrieves the rotation angle of pages. The angle is always a multiple of ninety degrees.

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

**Parameters**

- **Doc**
  - [in] Specifies a document that PXCV_Init created.

- **page_num**
  - [in] Specifies the zero-based page number from which to retrieve rotation information.

- **angle**
  - [out] Pointer to the LONG variable that receives the rotation angle. Possible results are:
    - **0** - no rotation.
    - **90** - page is rotated ninety degrees clockwise.
    - **180** - page is rotated one hundred and eighty degrees.
    - **270** - page is rotated ninety degrees counter-clockwise.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function fails then the return value is an error code.

If the function succeeds then the return value is DS_OK, or a different value that isn't an error code.
2.10 PXCV_GetPagesCount

PXCV_GetPagesCount retrieves the page count of documents.

HRESULT PXCV_GetPagesCount(
    PXVDocument Doc,
    DWORD* count
);

Parameters

Doc
    [in] Specifies a document that PXCV_Init created.

count
    [out] Pointer to a DWORD variable into which the page count is returned.

Please note that all functions and parameters are case-sensitive.

Return Values

If the function fails then the return value is an error code.

If the function succeeds then the return value is DS_OK, or a different value that isn't an error code.
2.11 PXCV_GetPermissions

PXCV_GetPermissions extracts the encryption level and permission settings of documents.

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

**Parameters**

*Doc*

[in] Specifies a document that PXCV_Init created.

*enclevel*

[out] Specifies a pointer to a DWORD variable that receives the encryption level information of the document. The value will be either 40 or 128.

*permFlags*

[out] Specifies a pointer to a DWORD variable that receives the permission flag information of the document.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function fails then the return value is an **error code**.

If the function succeeds then the return value is DS_OK, or a different value that is not an error code.
2.12 PXCV_Init

PXCV_Init creates new PDF objects that the majority of functions in the PDF_XChange Editor Simple SDK require.

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

**Parameters**

*pDoc*
- Pointer to a `PXVDocument` variable that will receive the PDF object.

*Key*
- [in] Pointer to a null-terminated string that contains the license key. This parameter can be set to NULL, in which case the library will operate in evaluation mode. (Permanent watermarks will be printed on all output).

*DevCode*
- [in] Pointer to a null-terminated string that contains the developer code. If this parameter is absent/invalid, then demo labels will be added to all generated pages. Please note that pDevCode is only needed when the older style license keys are used. If a new style of key is being used then the pDevCode parameter should be `NULL` or just an empty string. This is because the new style keys include the information necessary for the pDevCode parameter, but we must retain the property as there are still users who have the older style license keys.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is `DS_OK`, and a variable pointer to `pDoc` will contain the valid PDF object.
If the function fails then the return value is an error code.

Example (C++)

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

PXCV_ReadDocumentFromIStream uses an IStream interface to read documents.

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

**Parameters**

- **Doc**
  - `[in]` Specifies a document that PXCV_Init created.

- **stream**
  - `[in]` Specifies a pointer for the stream that loads the PDF document.

- **Flags**
  - `[in]` This parameter is reserved for future use and should be set to 0.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.

If the function return value is equal to PS_ERR_DocEncrypted then PXCV_CheckPassword and PXCV_FinishReadDocument must be used to provide a password.
2.14 PXCV_ReadDocumentFromMemory

**PXCV_ReadDocumentFromMemory** reads documents from memory buffers.

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

**Parameters**

*Doc*
- [in] Specifies a document that **PXCV_Init** created.

*mem*
- [in] Pointer to a memory buffer that contains the document to be opened.

*size*
- [in] Specifies the size (in bytes) of the buffer to which *mem* points.

*Flags*
- [in] This parameter is reserved for future use and should be set to 0.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an **error code**.

If the function return value is equal to **PS_ERR_DocEncrypted** then **PXCV_CheckPassword** and **PXCV_FinishReadDocument** must be used to complete the reading and parsing of the document.
Comments

Memory blocks are passed to the function and should not be released until the function \texttt{PXCV\_Delete} has been called.
Functions

2.15 PXCV_ReadDocumentW

PXCV_ReadDocumentW reads documents from specified PDF files.

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

Parameters

Doc
[in] Specifies a document that PXCV_Init created.

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

Flags
[in] This parameter is reserved for future use and should be set to 0.

Please note that all functions and parameters are case-sensitive.

Return Values

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.

If the function return value is equal to PS_ERR_DocEncrypted then PXCV_CheckPassword and PXCV_FinishReadDocument must be used to complete the reading and parsing of the document.

Example (C++)
// 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.
2.16 PXCV_ReleaseCachedData

**PXCV_ReleaseCachedData** releases cached document data.

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

**Parameters**

- **Doc**
  
  [in] Specifies a document that [PXCV_Init](#) created.

- **dwFlags**
  
  [in] Specifies the cached content to be freed. The flag `pxvrcd_ReleaseDocumentFonts` is the only flag available for this parameter, and it features the value `0x0002`. See [PXCV_ReleasePageCachedData](#) for further information on this parameter.

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.

**Comments**

- This function clears all cached data for the document, which means the next rendering operations will require re-reading and conversion of some data. However, clearing cached data may free a significant quantity of used memory. Therefore a call to this function is recommended after several pages have been rendered, especially if they will not be reused.
The PDF rasterizer requires significant memory usage for many operations, including: sequences of rendering operators, sharing fonts between pages for text rendering, sharing non-embedded fonts between documents and sharing images between pages. All of these objects must be converted into internal, rasterized representations before being used, which is likely to be a time-consuming operation. The PDF rasterizer keeps all objects as internal representations in order to accelerate page rendering. This is most significant when several parts of the same page are rendered sequentially. This is because it means that some objects will not require repeated conversion during subsequent rendering operations. However, some objects require a lot of memory - for example a "simple" page of text may contain several thousand rendering operators - therefore it may become necessary to free cached objects in order to free used memory. Two functions are provided to achieve this: `PXCV_ReleaseCachedData` and `PXCV_ReleasePageCachedData`. 

---

Last Updated: 21/11/2019

© 2019 Tracker Software Products (Canada) Ltd
2.17 PXCV_ReleasePageCachedData

This function releases page-specific, cached data for one document page. (It also contains optional functionality to release global/shared resources used subsequently to the initial release of data. See Comments for further information).

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

**Parameters**

**Doc**

[in] Specifies a document that PXCV_Init created.

**page_num**

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

**dwFlags**

[in] Specifies the cached content to be freed. Any combination of the following flags is possible:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxvrcd_ReleaseDocumentImages</td>
<td>0x0001</td>
<td>Release used images.</td>
</tr>
<tr>
<td>pxvrcd_ReleaseDocumentFonts</td>
<td>0x0002</td>
<td>Release used embedded fonts.</td>
</tr>
</tbody>
</table>
### Functions

| pxvrcd_ReleaseGlobalFonts | 0x0004 | Release used global (unembedded) fonts. |

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.

**Comments**

- This function releases data used to render a specified page (as opposed to all cached document data). If `dwFlags` is zero (0) then only the content-rendering operators for the page specified will be released. This function is recommended when a page has been rendered that is unlikely to be rendered again soon. Additionally, it is recommended that the `pxvrcd_ReleaseDocumentImages` flag is used to call `PXCV_ReleaseCachedData`, as images are not usually shared between adjacent pages.

- The PDF rasterizer requires significant memory usage for many operations, including: sequences of rendering operators, sharing fonts between pages for text rendering, sharing non-embedded fonts between documents and sharing images between pages. All of these objects must be converted into internal, rasterized representations before being used, which is likely to be a time-consuming operation. The PDF rasterizer keeps all objects as internal representations in order to accelerate page rendering. This is most significant when several parts of the same page are rendered sequentially. This is because it means that some objects will not require repeated conversion during subsequent rendering operations. However, some objects require a lot of memory - for example a "simple" page of text may contain several thousand rendering operators - therefore it may become necessary to free cached objects in order to free used memory. Two functions are provided to achieve this: `PXCV_ReleaseCachedData` and `PXCV_ReleasePageCachedData`.  

---
2.18 PXCV_SetCallBack

**PXCV_SetCallBack** sets the callback function used during the PDF rasterization process.

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

**Parameters**

- **Doc**
  
  [in] Specifies a document that PXCV_Init created.

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

  ```c
  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 below), and the third will always have the same value as that passed in **UserData**.

**Callback Function’s State Constants Table**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Meaning of Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXCVClb_Start</td>
<td>1</td>
<td><strong>MaxVal</strong> - maximum value of the level which will be passed.</td>
</tr>
<tr>
<td>PXCVClb_Processing</td>
<td>2</td>
<td>Current progress level - any value from 0 to <strong>MaxVal</strong>.</td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXCVClb_Finish</td>
<td>3</td>
<td>Any value from 0 to MaxVal (if all levels are passed then the value is MaxVal). This constant can be ignored if desired.</td>
</tr>
</tbody>
</table>

Please note that the callback function should return TRUE (any non-zero value) to continue processing or FALSE (zero) to abort the operation.

**UserData**

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

Please note that all functions and parameters are case-sensitive.

**Return Values**

If the function succeeds then the return value is DS_OK.

If the function fails then the return value is an error code.
2.19 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*

Spefies the rectangular area in which the PDF page rectangle will be drawn. See *Comments* for further information.

*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 that defines rendering options such as rotation and vector rendering. Any combination of the following values is possible:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pxvrpf_Rotate_NoRotate</code></td>
<td>0x0000</td>
<td>No rotation is carried out before pages are drawn.</td>
</tr>
<tr>
<td><code>PXCVClb_Processing</code></td>
<td>0x0001</td>
<td></td>
</tr>
</tbody>
</table>
# Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pxvrpf_Rotate_Rotate180</code></td>
<td>0x0002</td>
<td>Pages are rotated ninety degrees clockwise and then drawn. This is the standard Landscape layout.</td>
</tr>
<tr>
<td><code>pxvrpf_Rotate_Rotate90CCW</code></td>
<td>0x0003</td>
<td>Pages are rotated ninety degrees counterclockwise and then drawn.</td>
</tr>
<tr>
<td><code>pxvrpf_UseVectorRenderer</code></td>
<td>0x0004</td>
<td>Specifies that vector rendering is used (as opposed to raster rendering). This feature is recommended as the print job and the resources it uses are considerably smaller as a result. The Microsoft GDI is used for rendering.</td>
</tr>
<tr>
<td><code>pxvrpf_RenderAsGray</code></td>
<td>0x0008</td>
<td>Specifies that rendering is performed in grayscale mode.</td>
</tr>
</tbody>
</table>
| `pxvrpf_EmbeddedFontAsCurves` | 0x0010 | Specifies that all embedded fonts are rendered as curves - text functions will not be used. If this flag is not used then embedded, true-type fonts are installed temporarily for rendering.  

N.b. this flag has meaning only when `pxvrpf_UseVectorRenderer` is used - otherwise it is ignored. |
| `pxvrpf_AllFontsAsCurves` | 0x0030 | Specifies that all fonts are rendered as curves and without using text output API functions.  

N.b. this flag has meaning only when `pxvrpf_UseVectorRenderer` is used - otherwise it is ignored. |
| `pxvrpf_NoTransparentBkgnd` | 0x0040 | Specifies that raster images are filled with non-transparent white color before they are drawn. When this |
flag is not specified drawings are provided on a transparent background.

N.b. this flag has meaning only when \texttt{pxvrpf\_UseVectorRenderer} is not used.

\textbf{RenderTarget}

Specifies the rendering mode to be used. This is meaningful when the optional content exists within a document that is visible only in some rendering modes, such as push-buttons within an Adobe Acroform. Any combination of the following values is possible:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{pxvrm_Viewing}</td>
<td>0</td>
<td>Sets the rendering target as \texttt{View}. For example, this mode can be used for displaying a document on the screen.</td>
</tr>
<tr>
<td>\texttt{pxvrm_Printing}</td>
<td>1</td>
<td>Sets the rendering target as \texttt{Print}. For example, this mode is used for printing a document (or for a print preview).</td>
</tr>
<tr>
<td>\texttt{pxvrm_Exporting}</td>
<td>2</td>
<td>Sets the rendering target as \texttt{Export}. For example, this mode is used for exporting document content to a different format, such as supported raster image formats.</td>
</tr>
</tbody>
</table>

Please note that all functions and parameters/members are case-sensitive.

\textbf{Comments}
When specific sections of PDF pages are drawn, it is necessary to specify two areas. Firstly, the rectangular area of the target's DC must be given as "WholePageRect," which the entire PDF page will occupy. Secondly, the area of the PDF page drawn within WholePageRect must be given as "DrawRect." If DrawRect is set to NULL then the entire PDF page will be drawn within the target device's WholePageRect area. This simplifies the scaling of the PDF page (zoom level) and helps prevent rounding errors during the conversion from points to pixels.

Example 1

The objective is to draw a PDF page within the application window, given the following dimensions:

1. The PDF page has the dimensions 576 x 792 points (8 x 11 inches).
2. The desired "zoom level" is 400% - a scaling factor of four.
3. The application window's DC has the dimensions 600 x 800 pixels, with a DPI of 96.
4. The application window has scroll bars to control the page display. 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.

The first step is to calculate the PDF page's dimensions in pixels:

\[
\begin{align*}
\text{page\_width\_in\_pixels} &= (576 / 72) * 96 * 4 = 3072 \text{ pixels} \\
\text{page\_height\_in\_pixels} &= (792 / 72) * 96 * 4 = 4224 \text{ pixels}
\end{align*}
\]

If the dimensions of the PDF page and the zoom level remain constant then these values will also remain constant.

Therefore the WholePageRect and DrawRect values for the 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; // the window width
DrawRect.top = 800; // the window height

If the zoom level is constant then WholePageRect depends on only the position of the scroll bars. This simplifies the calculations involved and reduces rounding errors.

Example 2

The objective is to draw a portion of a PDF page, given the following dimensions:

1. The PDF page has a width of 576 points (8 inches) and a height of 792 points (11 inches).
2. The desired portion of the page, starting from point (10, 10), is defined as:
   left = 144pt;
   top = 288pt;
   right = 360pt;
   bottom = 648pt
3. The page portion will be drawn on the target DC with a zoom factor of 200%, or a scaling factor of two.
4. The DC has a DPI of 96.

The first step is to calculate the PDF page's dimensions in pixels:

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

Therefore the width of the required page portion is:

portion_width = ((360 - 144) / 72) * 96 * (200 / 100) = 576 pixels;
portion_height = 960 pixels.

The top-left point of the drawn area must be located at the coordinates (10, 10) on the DC. Therefore the top-left point of the complete page will have the following 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: \textbf{WholePageRect} = (-374, -758, -354 + 1536, -758 + 2112); and \textbf{DrawRect} = (10, 10, 10 + 576, 10 + 960).
2.20 PXV_DrawToImageParams

The PXV_DrawToImage structure formats image files that PXCV_DrawPageToIStream creates.

```c
typedef struct _PXV_DrawToImageParams {
    DWORD ImageFormat;
    DWORD Bpp;
} PXV_DrawToImageParams;
```

**Members**

*ImageFormat*

Specifies into which file format images are stored. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x504e4720</td>
<td>PNG</td>
<td>Specifies PNG format. This format supports transparency. Supported Bpp values: 1, 8, 24, 32.</td>
</tr>
<tr>
<td>0x4a504547</td>
<td>Jpeg</td>
<td>Specifies JPEG format. This format does not support transparency. Supported Bpp values are 8 and 24.</td>
</tr>
<tr>
<td>0x54494646</td>
<td>TIFF</td>
<td>Specifies TIFF format. Supported Bpp values are: 1, 8, 24, 32.</td>
</tr>
</tbody>
</table>

*Flags*

This member is reserved for future usage and should be set to 0.

*Bpp*
Functions

Specifies the bits per pixel value, which depends on the format used.

Please note that all functions and parameters/members are case-sensitive.
3 Error Handling

See the following pages for information on error handling:

- Error Codes [58]
- PXCV_Err_FormatFacility [58]
- PXCV_Err_FormatSeverity [57]
- PXCV_Err_FormatErrorCode [59]
3.1 Error Codes

Functions return an HRESULT value in most cases. This provides a simple means to determine the success/failure of a function call.

If the most significant bit or result is set to 1 then the specified error occurred. Any other result means the function was successful. The following macros for **C/C++** apply these checks:

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

**Note:** it is strongly recommended to use the same macros consistently in order to establish the success of function calls. A simple comparison with zero will often result in unreliable data, as detailed in the example below.

Please note that these macros are case-sensitive.

Functions may return warning codes that are neither equal to zero nor negative. Usually this means that the function was successful and is providing additional information about the call, for example that a default value was returned. See **Functions** for further information.

The **IS_DS_WARNING** macros can be used to determine if the return value generates a warning. The following code can be used to check for the error status of the **PXCV_CheckPassword** function:

```c
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 a value that indicates whether the password supplied was owner or
    ...
}
The following code is an example of how error-checking should **not** be performed:

```csharp
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 return value has not been adequately identified and this is unreliable!)
    ...
}
```

The most common error codes are listed in the table below, but it should be noted that functions may return other error codes. There are three further functions available for dealing with errors that may provide additional information: `PXCV_Err_FormatSeverity`, `PXCV_Err_FormatFacility` and `PXCV_Err_FormatErrorCode`. A code example is provided below the table. Please note that this function will provide information about all possible error codes.

Possible error values of PDF parser/structure:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ERR_NOTIMPLEMENTED</td>
<td>0x820f04b0</td>
<td>The function is not implemented.</td>
</tr>
<tr>
<td>PS_ERR_INVALID_ARG</td>
<td>0x820f0001</td>
<td>The argument is invalid.</td>
</tr>
<tr>
<td>PS_ERR_MEMALLOC</td>
<td>0x820f03e8</td>
<td>There is insufficient memory to perform the function.</td>
</tr>
<tr>
<td>PS_ERR_USER_BREAK</td>
<td>0x820f01f4</td>
<td>The user aborted the operation.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INTERNAL</td>
<td>There are an internal error.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INVALID_FILE_FORMAT</td>
<td>The file format is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_REQUIRED_PROP_NOT_SET</td>
<td>A required property is not set.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INVALID_PROP_TYPE</td>
<td>The property type is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INVALID_PROP_VALUE</td>
<td>The property value is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INVALID_OBJECT_NUMBER</td>
<td>The object number is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INVALID_PS_OPERATOR</td>
<td>The PS operator is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_UNKNOWN_OPERATOR</td>
<td>The operator is unknown.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_INVALID_CONTENT_STATE</td>
<td>The content state is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_NoPassword</td>
<td>There is no password.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_UnknownCryptFlt</td>
<td>There is an unknown crypt filter.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_WrongPassword 0x820f27aa</td>
<td>The password provided is incorrect.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_InvlaidObjStruct 0x820f27ab</td>
<td>The object structure is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_WrongEncryptDict 0x820f27ac</td>
<td>The encryption dictionary is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_DocEncrypted 0x820f27ad</td>
<td>The document is encrypted.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_DocNOTEncrypted 0x820f27ae</td>
<td>The document not encrypted.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_WrongObjStream 0x820f27af</td>
<td>The object stream is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_WrongTrailer 0x820f27b0</td>
<td>The document trailer is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_WrongXRef 0x820f27b1</td>
<td>The xref table is invalid.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_WrongDecodeParms 0x820f27b2</td>
<td>There is at least one invalid decode parameter.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_XRefNotFounded 0x820f27b3</td>
<td>The xref table was not found.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_DocAlreadyRead 0x820f27b4</td>
<td>The document is already read.</td>
<td></td>
</tr>
<tr>
<td>PS_ERR_DocNotRead 0x820f27b5</td>
<td>The document was not read.</td>
<td></td>
</tr>
</tbody>
</table>

Comments
Error Handling

The utility **DSErrorLookUp.exe** can provide additional error code data. This is a very useful application development tool and can be found in the installation folders. It is strongly recommended that developers utilize **DSErrorLookUp.exe** to debug their applications during the development process.

Example (C++)

```c++
// Using of PXCV_Err_FormatSeverity, PXCV_Err_FormatFacility, PXCV_Err_FormatErrorCode functions
char* err_message = NULL;
char* buf = NULL;
PXCEPage* 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 formatted text is the sum of the length for each description
    // plus some additional characters for formatting
    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;
}
```
3.2 PXCV_Err_FormatFacility

PXCV_Err_FormatFacility returns information on where errors occurred for the error code specified.

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

**Parameters**

**errorcode**

[in] Specifies the HRESULT that a function returned.

**buf**

[out] Specifies a pointer to a buffer where the error facility information is returned. Pass NULL for `buf` in order to determine the required buffer size.

**maxlen**

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

Please note that all functions and parameters are case-sensitive.

**Return Values**

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

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

If the function successfully retrieves information and the parameter `buf` is NULL then 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 then the return value is the number of characters written to the buffer (including a null-terminating character).
3.3  PXCV_Err_FormatSeverity

PXCV_Err_FormatSeverity returns information regarding the severity of the error. See Error Codes for further information.

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

**Parameters**

- `errorcode`  
  [in] Specifies an HRESULT that a library function returned.

- `buf`  
  [out] Specifies a pointer to a buffer where the error severity value is returned. Pass NULL for `buf` in order to determine the required buffer size.

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

Please note that all functions and parameters are case-sensitive.

**Return Values**

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

- If the function fails to retrieve information on an error code then the return value is zero.

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

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

PXCV_Err_FormatErrorCode provides error code information.

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

**Parameters**

errorcode  
[in] Specifies the HRESULT that a library function returned.

buf  
[out] Specifies a pointer to a buffer where the error description is returned. Pass NULL for `buf` in order to determine the required buffer size.

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

Please note that all functions and parameters are case-sensitive.

**Return Values**

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

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

If the function successfully retrieves information and the parameter `buf` is NULL, then 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, then the return value is the number of characters written to a buffer (including a null-terminating character).
Index

- E -
  Error Codes  50
  Error Handling  49

- F -
  Functions  10

- I -
  Installation  8
  Introduction  6

- P -
  PXCV_CheckPassword  11
  PXCV_Delete  12
  PXCV_DrawPageToDC  13
  PXCV_DrawPageToDIBSection  15
  PXCV_DrawPageToIStream  17
  PXCV_Err_FormatErrorCode  59
  PXCV_Err_FormatFacility  55
  PXCV_Err_FormatSeverity  57
  PXCV_FinishReadDocument  19
  PXCV_GetDocumentInfoW  20
  PXCV_GetPageDimensions  22
  PXCV_GetPageRotation  24
  PXCV_GetPagesCount  26
  PXCV_GetPermissions  27
  PXCV_Init  28
  PXCV_ReadDocumentFromIStream  30
  PXCV_ReadDocumentFromMemory  31
  PXCV_ReadDocumentW  33
  PXCV_ReleaseCachedData  35
  PXCV_ReleasePageCachedData  37
  PXCV_SetCallBack  39
  PXV_CommonRenderParameters  41
  PXV_DrawToImageParams  47

- S -
  System Requirements  9

- W -
  Welcome  4