Version 1.0.0
Sun May 1 2016

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Class Documentation

PdfAnnotAppearance Struct Reference

PdfAnnotAppearance.

Public Attributes

- PdfRGB fill_color
- PdfFillType fill_type
- PdfRGB border_color
- double border_width
- PdfBorderStyle border
- double opacity
- double font_size

Detailed Description

PdfAnnotAppearance.
Annot appearance.

Member Data Documentation

PdfBorderStyle PdfAnnotAppearance::border

The border style.

PdfRGB PdfAnnotAppearance::border_color

The border color.

double PdfAnnotAppearance::border_width

The border width in points. If this value is 0, no border is drawn. Default value: 1.

PdfRGB PdfAnnotAppearance::fill_color

The fill color.

PdfFillType PdfAnnotAppearance::fill_type

The fill type.

double PdfAnnotAppearance::font_size

The default appearance font size to be used in formatting the text.

double PdfAnnotAppearance::opacity

The constant opacity value to be used in painting the annotation.

PdfBookmarkAppearance Struct Reference

PdfBookmarkAppearance.

Public Attributes

- PdfRGB color
- int italic
- int bold

Detailed Description

PdfBookmarkAppearance.
Bookmark appearance.

Member Data Documentation

int PdfBookmarkAppearance::bold

1 - true, 0 - false.

PdfRGB PdfBookmarkAppearance::color

The fill color.

int PdfBookmarkAppearance::italic

1 - true, 0 - false.

PdfColorState Struct Reference

PdfColorState.

Public Attributes

- PdfFillType fill_type
- PdfFillType stroke_type
- PdfRGB fill_color
- PdfRGB stroke_color
- int fill_opacity
- int stroke_opacity

Detailed Description

PdfColorState.
Color state.

Member Data Documentation

PdfRGB PdfColorState::fill_color

Fill color.

int PdfColorState::fill_opacity

Fill opacity from 0 to 255.

PdfFillType PdfColorState::fill_type

Fill type.

PdfRGB PdfColorState::stroke_color

Stroke color.

int PdfColorState::stroke_opacity

Strok opacity from 0 to 255.

PdfFillType PdfColorState::stroke_type

Stroke type.

PdfDevPoint Struct Reference

PdfDevPoint.

Public Attributes

- int x
- int y

Detailed Description

PdfDevPoint.
A data structure representing a point in the page view's device space.

Member Data Documentation

int PdfDevPoint::x

x coordinate in device space.

int PdfDevPoint::y

y coordinate in device space.

PdfDevQuad Struct Reference

PdfDevQuad.

Public Attributes

- PdfDevPoint tl
- PdfDevPoint tr
- PdfDevPoint bl
- PdfDevPoint br

Detailed Description

PdfDevQuad.
A quadrilateral represented by four points (one at each corner) in device space coordinates. A quadrilateral differs from a rectangle in that a rectangle must always have horizontal and vertical sides, and opposite sides must be parallel.

Member Data Documentation

PdfDevPoint _PdfDevQuad::bl

Bottom left point.

PdfDevPoint _PdfDevQuad::br

Bottom right point.

PdfDevPoint _PdfDevQuad::tl

Top left point.

PdfDevPoint _PdfDevQuad::tr

Top right point.

PdfDevRect Struct Reference

PdfDevRect.

Detailed Description

PdfDevRect.
A data structure representing a rectangle in a device space.

PdfEventParams Struct Reference

PdfEventParams.

Public Attributes

- PdfEventType type
- PdfDoc * doc
- PdfPage * page
- PdfAnnot * annot

Detailed Description

PdfEventParams.
Callback structure.

Member Data Documentation

PdfAnnot* PdfEventParams::annot

Event annot or null.

PdfDoc* PdfEventParams::doc

Event document or null.

PdfPage* PdfEventParams::page

Event page or null.

PdfEventType PdfEventParams::type

Event type.

PdfFontState Struct Reference

PdfFontState.

Public Attributes

- PdfFontType type
- PdfFontFlags flags
- PdfRect bbox
- int ascent
- int descent
- int italic
- int bold
- int fixed_width
- int vertical
- int TT_font
- int height

Detailed Description

PdfFontState.
Font state.

Member Data Documentation

int PdfFontState::ascent

Ascent.

PdfRect PdfFontState::bbox

Font bounding box.

int PdfFontState::bold

1 - true, 0 - false.

int PdfFontState::descent

Descent.

int PdfFontState::fixed_width

1 - true, 0 - false.

PdfFontFlags PdfFontState::flags

Font flags.

int PdfFontState::height

Font height.

int PdfFontState::italic

Italic angle, 0 if horizontal.

int PdfFontState::TT_font

1 - true, 0 - false.

PdfFontType PdfFontState::type

Font type.

int PdfFontState::vertical

1 - true, 0 - false.

PdfGraphicState Struct Reference

PdfGraphicState.

Public Attributes

- PdfColorState color_state
- double line_width
- double miter_limit
- PdfLineCap line_cap
- PdfLineJoin line_join

Detailed Description

PdfGraphicState.
Graphics state.

Member Data Documentation

PdfColorState PdfGraphicState::color_state

Fill and stroke color properties in PdfColorState.

PdfLineCap PdfGraphicState::line_cap

The line cap style.

PdfLineJoin PdfGraphicState::line_join

The line join style.

double PdfGraphicState::line_width

Line width in user space coordinates (PDF).

double PdfGraphicState::miter_limit

The miter limit.

PdfMatrix Struct Reference

PdfMatrix.

Detailed Description

PdfMatrix.
Matrix containing six double numbers.

PdfPageMapParams Struct Reference

PdfPageMapParams.

Public Attributes

- PdfRect clip_rect
- PdfPageMapFlags flags

Detailed Description

PdfPageMapParams.
PageMap parameters. Currently not used.

Member Data Documentation

PdfRect PdfPageMapParams::clip_rect

Clipping rectangle in user space coordinates.

PdfPageMapFlags PdfPageMapParams::flags

Default kPageMapNone.

PdfPageRangeParams Struct Reference

PdfPageRangeParams.

Detailed Description

PdfPageRangeParams.
Specifies a range of pages in a document. Page numbers begin with 0.

PdfPageRenderParams Struct Reference

PdfPageRenderParams.

Public Attributes

- PdfRect clip_rect
- PdfRenderFlags render_flags

Detailed Description

PdfPageRenderParams.
Handles page rendering.

Member Data Documentation

PdfRect PdfPageRenderParams::clip_rect

Clipping rectangle in device space coordinates.

PdfRenderFlags PdfPageRenderParams::render_flags

PdfRenderFlags

PdfPoint Struct Reference

PdfPoint.

Public Attributes

- double x
- double y

Detailed Description

PdfPoint.
A data structure representing a point in the user space. To avoid the device-dependent effects of specifying objects in device space, PDF defines a device-independent coordinate system that always bears the same relationship to the current page, regardless of the output device on which printing or displaying occurs. This device-independent coordinate system is called user space. The origin of the user space(0, 0) represents the bottom-left corner of the PDF page. PDF files specify 72 points to 1 physical inch.

Member Data Documentation

double PdfPoint::x

x coordinate in user space.

double PdfPoint::y

y coordinate in user space.

PdfQuad Struct Reference

PdfQuad.

Public Attributes

- PdfPoint tl
- PdfPoint tr
- PdfPoint bl
- PdfPoint br

Detailed Description

PdfQuad.
A quadrilateral represented by four points (one at each corner) in user space coordinates. A quadrilateral differs from a rectangle in that a rectangle must always have horizontal and vertical sides, and opposite sides must be parallel.

Member Data Documentation

PdfPoint PdfQuad::bl

Bottom left point.

PdfPoint PdfQuad::br

Bottom right point.

PdfPoint PdfQuad::tl

Top left point.

PdfPoint PdfQuad::tr

Top right point.

PdfRect Struct Reference

PdfRect.

Detailed Description

PdfRect.
A data structure representing a rectangle in a user space (a quadrilateral having only horizontal and vertical sides) The coordinate system is defined so that (0,0) is at the top, x increases to the right, and y increases down. A PdfRect is defined so that its top is above its bottom, but this means that 0 < top < bottom.

PdfRGB Struct Reference

PdfRGB.

Public Attributes

- int r
- int g
- int b

Detailed Description

PdfRGB.
RGB color representation.

Member Data Documentation

int PdfRGB::b

Blue component from 0 to 255.

int PdfRGB::g

Green component from 0 to 255.

int PdfRGB::r

Red component from 0 to 255.

PdfTextState Struct Reference

PdfTextState.

Public Attributes

- PdfColorState color_state
- PdfFont * font
- double font_size
- double char_spacing
- double word_spacing
- PdfTextStateFlag flags

Detailed Description

PdfTextState.
PdfTextState structure containing the text state information.

Member Data Documentation

double PdfTextState::char_spacing

Character spacing.

PdfColorState PdfTextState::color_state

Fill and stroke color properties.

PdfTextStateFlag PdfTextState::flags

Test state flag.

PdfFont* PdfTextState::font

Text font.

double PdfTextState::font_size

Text font size.

double PdfTextState::word_spacing

Word spacing.

PdfWatermarkParams Struct Reference

PdfWatermarkParams.

Public Attributes

- PdfPageRangeParams page_range
- int order_top
- PdfHorizAlign h_align
- PdfVertAlign v_align
- int percentage_vals
- double h_value
- double v_value
- double scale
- double rotation
- double opacity

Detailed Description

PdfWatermarkParams.
Page rendering flags.

Member Data Documentation

PdfHorizAlign PdfWatermarkParams::h_align

The horizontal alignment to be used when adding the watermark to a page.

double PdfWatermarkParams::h_value

The horizontal offset value to be used when adding the watermark on a page. If percentageVals is 1, this value is a percentage of the page width, with 1.0 meaning 100 % . If percentageVals is 0, this value is in user units.

double PdfWatermarkParams::opacity

The opacity to be used when adding the watermark, with 0.0 meaning fully transparent and 1.0 meaning fully opaque.

int PdfWatermarkParams::order_top

An integer specifying where in the page z-order the watermark should be added. If it is 1, the watermark is added to the front of the page; if it is 0, it is added as a background.

PdfPageRangeParams PdfWatermarkParams::page_range

The page range of the document to which the watermark should be added.

int PdfWatermarkParams::percentage_vals

An integer specifying the units of horizValue and vertValue. If it is 1, horizValue and vertValue represent percentages of the page dimensions. If it is 0, horizValue and vertValue are in user units.

double PdfWatermarkParams::rotation

The counter-clockwise rotation, in degrees, to be used when adding the watermark.

double PdfWatermarkParams::scale

The scale factor to be used when adding the watermark, with 1.0 meaning 100%.

PdfVertAlign PdfWatermarkParams::v_align

The vertical alignment to be used when adding the watermark to a page.

double PdfWatermarkParams::v_value

The vertical offset value to be used when adding the watermark on a page. If percentageVals is 1, this value is a percentage of the page height, with 1.0 meaning 100 % . If percentageVals is 0, this value is in user units.

PdfWhitespaceParams Struct Reference

PdfWhitespaceParams.

Public Attributes

- double width
- double height

Detailed Description

PdfWhitespaceParams.
Whitespace Cover parameters.

Member Data Documentation

double PdfWhitespaceParams::height

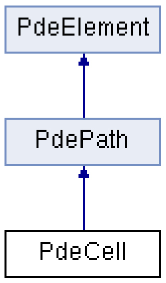
Minimum height of whitespace area on the page.

double PdfWhitespaceParams::width

Minimum with of whitespace area on the page.

PdeCell Struct Reference

PdeCell class.
Inheritance diagram for PdeCell:



Public Member Functions

- int GetRowSpan ()=0
- int GetColSpan ()=0
  Returns the number of columns spanned by the cell.

Detailed Description

PdeCell class.
A PdeCell class represents a single cell of PdeTable element.

Member Function Documentation

int PdeCell::GetColSpan ()

Returns the number of columns spanned by the cell.

Returns:

Cell colspan, 0 if the cell is merged with another cell.

See also:

PdeTable::GetCell

int PdeCell::GetRowSpan ()

Returns the number of rows spanned by the cell. The default value is 0, which indicates that this cell is merged. NOTE: Ignore such cells in further processing.

Returns:

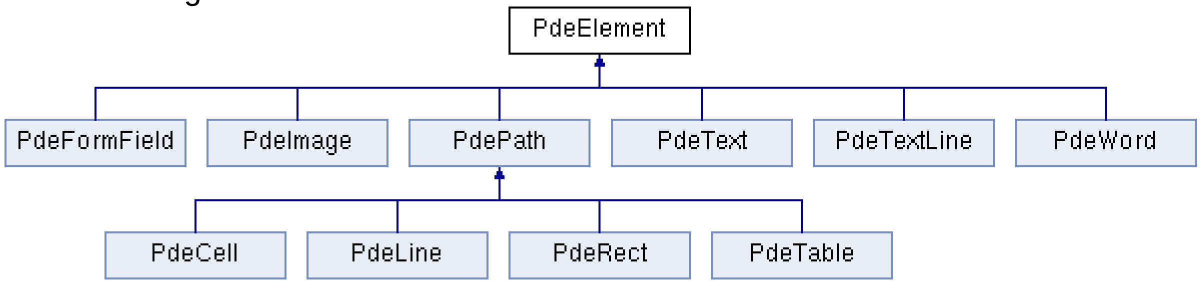
Cell rowspan, 0 if the cell is merged with another cell.

See also:

PdeTable::GetCell

PdeElement Struct Reference

PdeElement class.
Inheritance diagram for PdeElement:



Public Member Functions

- PdfElementType GetType ()=0

Gets the type of an element.

- void GetBBox (PdfRect *bbox)=0
- int GetNumChildren ()=0
  Gets the number of child elements in an element object.
- PdeElement * GetChild (int index)=0
  Gets the requested child element from an element.
- int GetId ()=0
  Gets the id of an element. The id is unique number on a page.
- bool Save (const wchar_t *path, PdfImageFormat format, PdfPageView *page_view)=0
  Saves the element into an image file.

Detailed Description

PdeElement class.
PdeElement is the base class for elements of a pagemap (PdePageMap). The general PdeElement methods allow you to get and set general element properties. PdeElement is an abstract superclass from which the PdeText, PdeTextLine, PdeWord, PdeTable, PdeImage, PdePath, PdeLine, PdeRect, PdeTableCell, PdeFormField classes are derived. Use PdeElement::GetType method to find the type of an element.

Member Function Documentation

void PdeElement::GetBBox (PdfRect * *bbox*)

Gets the bounding box for an element in user space coordinates. To avoid the device-dependent effects of specifying objects in device space, PDF defines a device-independent coordinate system that always bears the same relationship to the current page, regardless of the output device on which printing or displaying occurs. This device-independent coordinate system is called user space. The origin of the user space(0, 0) represents the bottom-left corner of the PDF page. PDF files specify 72 points to 1 physical inch. The returned bounding box is guaranteed to encompass the element.

Parameters:

bbox	(filled by the method) A pointer to a PdfRect structure specifying the bounding box of an element, specified in user space coordinates.

PdeElement* PdeElement::GetChild (int *index*)

Gets the requested child element from an element.

Parameters:

index	The index of element to obtain.

Returns:

Requested element.

See also:

PdeElement::GetNumChildren

int PdeElement::GetId ()

Gets the id of an element. The id is unique number on a page.

Returns:

Unique number for the element.

int PdeElement::GetNumChildren ()

Gets the number of child elements in an element object.

Returns:

The number of children.

See also:

PdeElement::GetChild

PdfElementType PdeElement::GetType ()

Gets the type of an element.

Returns:

Element type, kElementUnknown otherwise.

bool PdeElement::Save (const wchar_t * *path*, PdfImageFormat *format*, PdfPageView * *page_view*)

Saves the element into an image file.

Parameters:

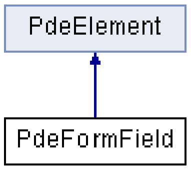
path	Path where to save image data in requested format.
format	PdfImageFormat.
page_view	PdfPageView for which the element should be saved.

Returns:

true if succeeded, false otherwise.

PdeFormField Struct Reference

PdeFormField class.
Inheritance diagram for PdeFormField:



Public Member Functions

- PdfWidgetAnnot * GetWidgetAnnot ()=0
  Gets the annotation object from a form field element.

Detailed Description

PdeFormField class.
A PdeFormField class is a page content element containing an interactive form.

Member Function Documentation

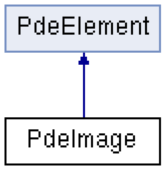
PdfWidgetAnnot* PdeFormField::GetWidgetAnnot ()

Gets the annotation object from a form field element.

Returns:
PdfWidgetAnnot object.

PdeImage Struct Reference

PdeImage class.
Inheritance diagram for PdeImage:



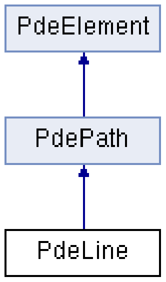
Additional Inherited Members

Detailed Description

PdeImage class.
A PdfImage class is a page content element containing an image graphics.

PdeLine Struct Reference

PdeLine class.
Inheritance diagram for PdeLine:



Additional Inherited Members

Detailed Description

PdeLine class.
A PdeLine class is a pagemap element containing a vector graphics in form of a line. It contains only horizontal and vertical lines.

PdePageMap Struct Reference

PdePageMap class.
Inherited by CPdePageMap.

Public Member Functions

- int GetNumElements ()=0
  Gets the number of elements in a pagemap.
- PdeElement * GetElement (int index)=0
- bool GetWhitespace (PdfWhitespaceParams *params, int index, PdfRect *bbox)=0

Searches for whitespaces at the page. They are sorted from biggest to smallest.

- void GetBBox (PdfRect *bbox)=0

Detailed Description

PdePageMap class.
The PdePageMap object is a storage of all PdeElements whose were recognized on the page. A PdePageMap may be obtained from an existing page with the PdfPage::GetPageMap method. Once your application has the page’s PdePageMap, it can get each logical element with GetElement method.

Member Function Documentation

void PdePageMap::GetBBox (PdfRect * *bbox*)

Gets the bounding box for a pagemap. The bounding box is the rectangle that encloses all text, graphics, and images on the page.

Parameters:

bbox	(filled by the method) A PdfRect specifying the page's box.

See also:

PdePageMap::GetWhitespace

PdeElement* PdePageMap::GetElement (int *index*)

Gets the requested element from a pagemap. You should never depend on these objects lasting the lifetime of the pagemap. You should extract the information you need from he object immediately and refer to it no further in your code. NOTE: This method does not copy the element, so do not destroy it.

Parameters:

index	Index of element to obtain.

Returns:

The requested element.

See also:

PdePageMap::GetNumElements

int PdePageMap::GetNumElements ()

Gets the number of elements in a pagemap.

Returns:

Number of elements in PdePageMap.

See also:

PdePageMap::GetElement

bool PdePageMap::GetWhitespace (PdfWhitespaceParams * *params*, int *index*, PdfRect * *bbox*)

Searches for whitespaces at the page. They are sorted from biggest to smallest.

```
void GetWhitespace(std::wstring doc_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());

  // load first page
  PdfPage* page = doc->AcquirePage(0);
  if (!page)
    throw std::runtime_error(pdfix->GetError());
  PdfPageMapParams params;
  PdePageMap* page_map = page->AcquirePageMap(&params, nullptr, nullptr);
  if (!page_map)
    throw std::runtime_error(pdfix->GetError());

  // Search for the best place where to put watermark
  PdfRect bbox;
  PdfWhitespaceParams whitespace_params;
  // set watermark width in user space coordinates
  whitespace_params.width = 100;
  // set watermark height in user space coordinates
  whitespace_params.height = 50;
  if (page_map->GetWhitespace(&whitespace_params, 0, &bbox)) {
    // use this bbox to place watermark into that place
  }

  // release resources
  doc->ReleasePage(0);
  doc->Close();
}
```

Parameters:

params	Whitespace parameters that specify which whitespace should be obtained.

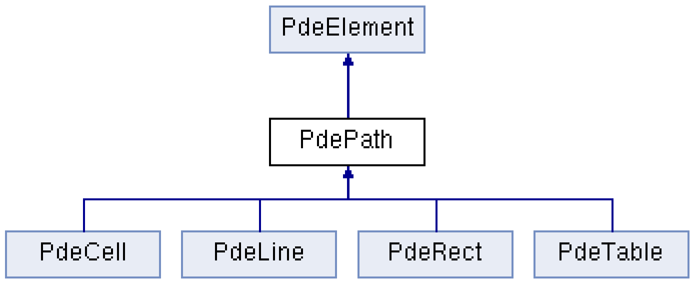
index	Index of whitespace to obtain. Set to zero for the first call. Update the index with each consecutive call of the method while result is true.
bbox	(filled by the method) A PdfRect specifying requested whitespace.

Returns:

This method returns true if whitespace with requested params exists, otherwise it returns false.

PdePath Struct Reference

PdePath class.
Inheritance diagram for PdePath:



Public Member Functions

- void GetGraphicState (PdfGraphicState *g_state)=0
  Gets the graphics state information for an element.

Detailed Description

PdePath class.
A PdePath class is a page content element containing a vector graphics. It can have fill and stroke attributes. It also has graphics state attributes.

Member Function Documentation

void PdePath::GetGraphicState (PdfGraphicState * *g_state*)

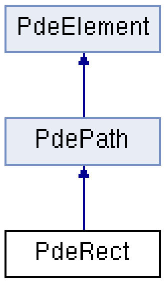
Gets the graphics state information for an element.

Parameters:

g_state	(filled by the method) Pointer to a PdfGraphicState structure that contains graphics state information for pdeElement.

PdeRect Struct Reference

PdeRect class.
Inheritance diagram for PdeRect:



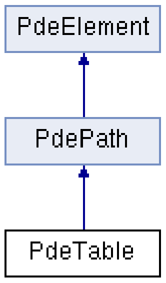
Additional Inherited Members

Detailed Description

PdeRect class.
A PdeRect class is a page content element containing a vector graphics with rectangular shape.

PdeTable Struct Reference

PdeTable class.
Inheritance diagram for PdeTable:



Public Member Functions

- int GetNumRows ()=0
  Returns the number of table rows.
- int GetNumCols ()=0

Returns the number of table columns.

- PdeCell * GetCell (int row, int col)=0
  Returns the cell object of table columns.

Detailed Description

PdeTable class.
PdeTable class represents tables extracted from PDF document. PdePageMap recognizes and decomposes tables in PDF documents and store the extracted data in a PdeTable class for easier reuse.

Member Function Documentation

PdeCell* PdeTable::GetCell (int *row*, int *col*)

Returns the cell object of table columns.

Parameters:

row	The row number of the requested cell.
col	The col number of the requested cell.

Returns:

A requested cell.

See also:

PdeTable::PdeTableGetNumRows, PdeTable::PdeTableGetNumCols

int PdeTable::GetNumCols ()

Returns the number of table columns.

Returns:

A number of table columns.

See also:

PdeTable::PdeTableGetNumRows, PdeTable::PdeTableGetCell

int PdeTable::GetNumRows ()

Returns the number of table rows.

Returns:

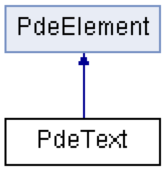
A number of table rows.

See also:

PdeTable::PdeTableGetNumCols, PdeTable::PdeTableGetCell

PdeText Struct Reference

PdeText class.
Inheritance diagram for PdeText:



Public Member Functions

- int GetText (wchar_t *buffer, int len)=0
  Get the text of the text element.
- bool HasTextState ()=0
- void GetTextState (PdfTextState *text_state)=0

Get the text state of the text element.

- int GetNumTextLines ()=0
  Get the number of lines of text in text element.
- PdeTextLine * GetTextLine (int index)=0
  Get the text line element from the text element.
- int GetNumWords ()=0
  Get the number of words of text in text element.
- PdeWord * GetWord (int index)=0
  Get the word from the text element.
- PdfTextAlignment GetAlignment ()=0
  Get the text element alignment.
- double GetLineSpacing ()=0
  Get the text element line spacing.
- double GetIndent ()=0
  Get the text element indent.

Detailed Description

PdeText class.
A PdeText object represents a group of text line objects which forms a paragraph in a PDF file.

Member Function Documentation

PdfTextAlignment PdeText::GetAlignment ()

Get the text element alignment.

Returns:
The text element alignent.

double PdeText::GetIndent ()

Get the text element indent.

Returns:
The text element indent.

double PdeText::GetLineSpacing ()

Get the text element line spacing.

Returns:
The text element line spacing.

int PdeText::GetNumTextLines ()

Get the number of lines of text in text element.

Returns:
Number of lines.

See also:
PdeText::GetTextLine

int PdeText::GetNumWords ()

Get the number of words of text in text element.

Returns:
Number of words.

See also:
PdeText::GetWord

int PdeText::GetText (wchar_t * *buffer*, int *len*)

Get the text of the text element.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdeTextLine::GetText, PdeWord::GetText

PdeTextLine* PdeText::GetTextLine (int *index*)

Get the text line element from the text element.

Parameters:

index	The index of line to get.

Returns:

PdeTextLine element.

See also:

PdeText::GetNumTextLines

void PdeText::GetTextState (PdfTextState * *text_state*)

Get the text state of the text element.

Parameters:

text_state	(filled by method) A pointer to a PdfTextState structure specifying the text state of a first text character.

See also:

PdeText::HasTextState

PdeWord* PdeText::GetWord (int *index*)

Get the word from the text element.

Parameters:

index	The index of word to get.

Returns:

PdeWord element.

See also:

PdeText::GetNumWords

bool PdeText::HasTextState ()

Checks whether the text state can be obtained. It means that an each text line of the text element has the same text state.

Returns:

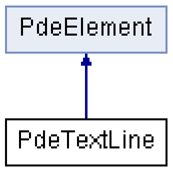
true if the text state is the same for the whole text, false otherwise. In that case use PdeTextLine::GetTextState to obtain correct values.

See also:

PdeText::GetTextState, PdeTextLine::GetTextState

PdeTextLine Struct Reference

PdeTextLine class.
Inheritance diagram for PdeTextLine:



Public Member Functions

- int GetText (wchar_t *buffer, int len)=0
  Get the text of the text line element.
- bool HasTextState ()=0
- void GetTextState (PdfTextState *text_state)=0
  Get the text state of the text line element.
- int GetNumWords ()=0
  Get the number of word elements in the text line element.
- PdeWord * GetWord (int index)=0
  Get the word element from the text line element.

Detailed Description

PdeTextLine class.
A PdeTextLine object represents a line of text in a PDF file. Each text line contains an array of PdeWord objects with in one or more styles.

Member Function Documentation

int PdeTextLine::GetNumWords ()

Get the number of word elements in the text line element.

Returns:

Number of word elements within the text line element.

int PdeTextLine::GetText (wchar_t * *buffer*, int *len*)

Get the text of the text line element.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdeWord::GetText, PdeText::GetText

void PdeTextLine::GetTextState (PdfTextState * *text_state*)

Get the text state of the text line element.

Parameters:

text_state	(filled by method) A pointer to a PdfTextState structure specifying the text state of a first line character.

See also:

PdeTextLine::HasTextState

PdeWord* PdeTextLine::GetWord (int *index*)

Get the word element from the text line element.

Parameters:

index	The index of word element to obtain.

Returns:

PdeWord element.

bool PdeTextLine::HasTextState ()

Checks whether the text state can be obtained. It means that an each word of the text line has the same text state.

Returns:

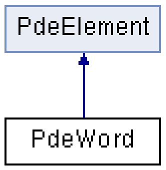
true if the text state is the same for the whole line, false otherwise. In that case use PdeWord::GetTextState to obtain correct values.

See also:

PdeTextLine::GetTextState, PdeWord::GetTextState

PdeWord Struct Reference

PdeWord class.
Inheritance diagram for PdeWord:



Public Member Functions

- int GetText (wchar_t *buffer, int len)=0
  Get the text of the word element.
- bool HasTextState ()=0
- void GetTextState (PdfTextState *text_state)=0
  Get the text state of the word element.
- int GetNumChars ()=0
  Get the number of characters in word element.
- int GetCharText (int index, wchar_t *buffer, int len)=0

Get the text of one character of the word.

- void GetCharTextState (int index, PdfTextState *text_state)=0
  Get the text state information of the word character.
- void GetCharBBox (int index, PdfRect *bbox)=0
  Gets the bounding box of one character in user space coordinates.

Detailed Description

PdeWord class.
A PdeWord object represents a word in a PDF file. Each word contains a sequence of characters in one or more styles.

Member Function Documentation

void PdeWord::GetCharBBox (int *index*, PdfRect * *bbox*)

Gets the bounding box of one character in user space coordinates.

Parameters:

index	The index of a character.
bbox	(filled by the method) A pointer to a PdfRect structure specifying the bounding box of a character, specified in user space coordinates.

int PdeWord::GetCharText (int *index*, wchar_t * *buffer*, int *len*)

Get the text of one character of the word.

Parameters:

index	The index of a character.
buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdeWord::GetNumChars

void PdeWord::GetCharTextState (int *index*, PdfTextState * *text_state*)

Get the text state information of the word character.

Parameters:

index	The index of a character.
text_state	(filled by method) A pointer to a PdfTextState structure specifying the text state of a character.

int PdeWord::GetNumChars ()

Get the number of characters in word element.

Returns:

Number of characters.

int PdeWord::GetText (wchar_t * *buffer*, int *len*)

Get the text of the word element.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdeTextLine::GetText, PdeText::GetText

void PdeWord::GetTextState (PdfTextState * *text_state*)

Get the text state of the word element.

Parameters:

text_state	(filled by method) A pointer to a PdfTextState structure specifying the text state of a word first character.

See also:

PdeWord::HasTextState

bool PdeWord::HasTextState ()

Checks whether the text state can be obtained. It means that an each character of the word has the same text state.

Returns:

true if the text state is the same for the whole word, false otherwise. In that case use PdeWord::GetCharTextState to obtain correct values.

See also:

PdeWord::GetTextState, PdeWord::GetCharTextState

PdfAction Struct Reference

PdfAction class.
Inherited by CPdfAction.

Public Member Functions

- PdfActionType GetSubtype ()=0
  Gets an action's subtype.
- int GetJavaScript (wchar_t *buffer, int len)=0
  Gets the string buffer from the JavaScript action.
- int GetURI (wchar_t *buffer, int len)=0

Detailed Description

PdfAction class.
The PdfAction are tasks that pdf viewer performs when a user clicks on a link or a bookmark.

Member Function Documentation

int PdfAction::GetJavaScript (wchar_t * *buffer*, int *len*)

Gets the string buffer from the JavaScript action.

Parameters:

buffer	(filled by method) if the buffer is null function returns required length of string.

len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

PdfActionType PdfAction::GetSubtype ()

Gets an action's subtype.

Returns:

The PdfActionType corresponding to the action's subtype.

int PdfAction::GetURI (wchar_t * *buffer*, int *len*)

Gets the string buffer from the URI action. A uniform resource identifier (URI) is a string that identifies a resource on the Internet — typically a file that is the destination of a hypertext link.

Parameters:

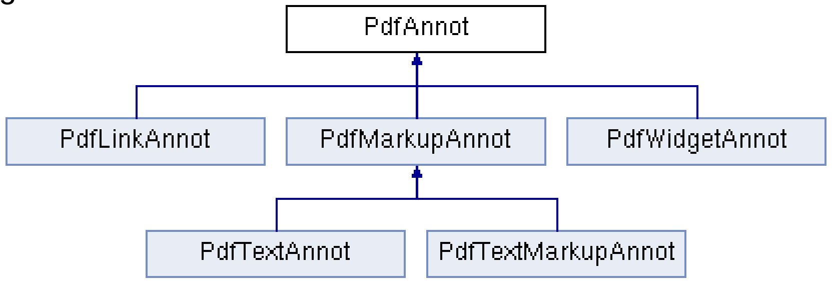
buffer	(filled by method) if the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

PdfAnnot Struct Reference

PdfAnnot class.
Inheritance diagram for PdfAnnot:



Public Member Functions

- PdfAnnotSubtype GetSubtype ()=0
  Gets an annotation's subtype.
- PdfAnnotFlags GetFlags ()=0

Gets an annotation's flags.

- void GetAppearance (PdfAnnotAppearance *appearance)=0
  Gets an annotation's appearance.
- void GetBBox (PdfRect *bbox)=0
  Gets the annotation bounding box.
- bool PointInAnnot (PdfPoint *point)=0
- bool RectInAnnot (PdfRect *rect)=0

Detailed Description

PdfAnnot class.
An annotation associates an object such as a note, sound, or movie with a location on a page of a PDF document, or provides a way to interact with the user by means of the mouse and keyboard.

Member Function Documentation

void PdfAnnot::GetAppearance (PdfAnnotAppearance * *appearance*)

Gets an annotation's appearance.

Parameters:

appearance	(filled by method) Pointer to a PdfAnnotAppearance structure.

void PdfAnnot::GetBBox (PdfRect * *bbox*)

Gets the annotation bounding box.

Parameters:

bbox	(filled by the method) Pointer to PdfRect structure to fill.

PdfAnnotFlags PdfAnnot::GetFlags ()

Gets an annotation's flags.

Returns:

The flags, or 0 if the annotation does not have a flags key.

PdfAnnotSubtype PdfAnnot::GetSubtype ()

Gets an annotation's subtype.

Returns:

The PdfAnnotSubtype corresponding to the annot's subtype.

bool PdfAnnot::PointInAnnot (PdfPoint * *point*)

Tests whether the specified point is within an annotation. If an annotation consists of more quads, it tests each quad individually.

Parameters:

point	The point to test.

Returns:

true if the point is within an annotation, false otherwise.

bool PdfAnnot::RectInAnnot (PdfRect * *rect*)

Tests whether the specified rect is within an annotation. If an annotation consists of more quads, it tests each quad individually.

Parameters:

rect	The rectangle to test.

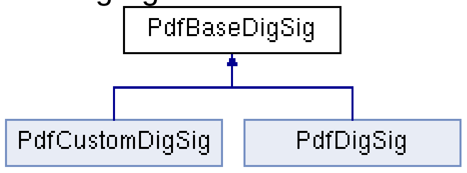
Returns:

true if the the whole rectangle is within an annotation, false otherwise.

PdfBaseDigSig Struct Reference

PdfBaseDigSig class.

Inheritance diagram for PdfBaseDigSig:



Public Member Functions

- void Destroy ()=0
  Destroys digital signature's resources.
- bool SetReason (const wchar_t *reason)=0
  Sets the reason for the signing.
- bool SetLocation (const wchar_t *location)=0
  Sets the location of signing.
- bool SetContactInfo (const wchar_t *contact)=0
  Sets the contact information of the signer.
- bool SetName (const wchar_t *name)=0
- bool SetTimeStampServer (const wchar_t *url, const wchar_t *user_name, const wchar_t *password)=0
  Set the timestamp server url and access credentials to apply the timestamp.
- bool SignDoc (PdfDoc *doc, const wchar_t *path)=0
  Apply the digital signature and save document to specified path.

Detailed Description

PdfBaseDigSig class.
A digital signature can be used to authenticate the identity of a user and the document's contents. It stores information about the signer and the state of the document when it was signed.

Member Function Documentation

void PdfBaseDigSig::Destroy ()

Destroys digital signature's resources.

See also:

CreatePdfDigSig

bool PdfBaseDigSig::SetContactInfo (const wchar_t * *contact*)

Sets the contact information of the signer.

Parameters:

contact	Information provided by the signer to enable a recipient to contact the signer to verify the signature, for example, a phone number, etc.

Returns:

true if was set successfully, false otherwise.

bool PdfBaseDigSig::SetLocation (const wchar_t * *location*)

Sets the location of signing.

Parameters:

location	The CPU host name or physical location of the signing.

Returns:

true if was set successfully, false otherwise.

bool PdfBaseDigSig::SetName (const wchar_t * *name*)

Sets the name of the person or authority signing the document. This value is be used when it is not possible to extract the name from the signature; for example, from the certificate of the signer or when PdfCustomDigSig is used.

Parameters:

name	Name for signing.

Returns:

true if was set successfully, false otherwise.

See also:

PdfCustomDigSig

bool PdfBaseDigSig::SetReason (const wchar_t * *reason*)

Sets the reason for the signing.

Parameters:

reason	Reason for the signing.

Returns:

true if was set successfully, false otherwise.

bool PdfBaseDigSig::SetTimeStampServer (const wchar_t * *url*, const wchar_t * *user_name*, const wchar_t * *password*)

Set the timestamp server url and access credentials to apply the timestamp.

Parameters:

url	The url of the timesramp server .
user_name	The user name for accessing the timestamp server.
password	The password for accessing the timestamp server.

Returns:

true if time stamp was set, false otherwise.

bool PdfBaseDigSig::SignDoc (PdfDoc * *doc*, const wchar_t * *path*)

Apply the digital signature and save document to specified path.

Parameters:

doc	The document to be signed.
path	The path where the signed document will be saved.

Returns:

true if document was signed, false otherwise.

PdfBookmark Struct Reference

PdfBookmark class.
Inherited by CPdfBookmark.

Public Member Functions

- int GetTitle (wchar_t *buffer, int len)=0
  Gets a bookmark's title.
- void GetAppearance (PdfBookmarkAppearance *appearance)=0
  Gets a bookmark's appearance.
- PdfAction * GetAction ()=0
  Gets a bookmark's action object.
- int GetNumChildren ()=0
  Gets the number of child bookmark in a parent bookmark object.
- PdfBookmark * GetChild (int index)=0
  Gets the requested child bookmark from a parent bookmark.
- PdfBookmark * GetParent ()=0
  Gets a bookmark's parent bookmark.

Detailed Description

PdfBookmark class.
A bookmark corresponds to an outline object in a PDF document. A document outline allows the user to navigate interactively from one part of the document to another. An outline consists of a tree-structured hierarchy of bookmarks, which display the document's structure to the user. Each bookmark has: A title that appears on screen. An action that specifies what happens when the user clicks on the bookmark.

Member Function Documentation

PdfAction* PdfBookmark::GetAction ()

Gets a bookmark's action object.

Returns:

The bookmark's action object or nullptr if bookmark does not have an action.

void PdfBookmark::GetAppearance (PdfBookmarkAppearance * *appearance*)

Gets a bookmark's appearance.

Parameters:

appearance	(filled by method) Pointer to a PdfBookmarkAppearance structure.

PdfBookmark* PdfBookmark::GetChild (int *index*)

Gets the requested child bookmark from a parent bookmark.

Parameters:

index	The index of bookmark to obtain.

Returns:

Requested bookmark.

See also:

PdfBookmark::GetNumChildren

int PdfBookmark::GetNumChildren ()

Gets the number of child bookmark in a parent bookmark object.

Returns:

The number of children.

See also:

PdfBookmark::GetChild

PdfBookmark* PdfBookmark::GetParent ()

Gets a bookmark's parent bookmark.

Returns:

Parent bookmark, or null if is the root bookmark of a document. NOTE: If parent is null, call only PdfBookmark::GetNumChildren and PdfBookmark::GetChild methods. Other methods return false.

int PdfBookmark::GetTitle (wchar_t * *buffer*, int *len*)

Gets a bookmark's title.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

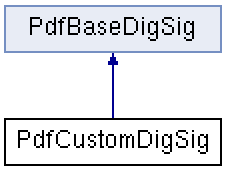
Returns:

Number of characters written into buffer of required length.

PdfCustomDigSig Struct Reference

PdfCustomDigSig class.

Inheritance diagram for PdfCustomDigSig:



Public Member Functions

- bool RegisterDigestDataProc (_callback_ PdfDigestDataProc proc, void *data)=0
  Registers a user-supplied procedure to call when PdfCustomDigSig::SignDoc is called.

Detailed Description

PdfCustomDigSig class.
Platfor independent digital signature object. Caller can handle signing process with callbacks.

Member Function Documentation

bool PdfCustomDigSig::RegisterDigestDataProc (_callback_ PdfDigestDataProc *proc*, void * *data*)

Registers a user-supplied procedure to call when PdfCustomDigSig::SignDoc is called.

Parameters:

proc	A user-supplied callback to call when the digital signature requests signed digests data.
data	A pointer to user-supplied data to pass to proc each time it is called.

Returns:

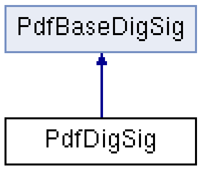
true if callback was registered, false otherwise.

See also:

PdfCustomDigSig::RegisterDigestDataLenProc

PdfDigSig Struct Reference

CPdfDigSig class.
Inheritance diagram for PdfDigSig:



Public Member Functions

- bool SetPfxFile (const wchar_t *pfx_file, const wchar_t *pfx_password)=0
  Set the pfx file for signing the document with digital signature.

Detailed Description

CPdfDigSig class.
Uses OpenSSL to handle certificates.

Member Function Documentation

bool PdfDigSig::SetPfxFile (const wchar_t * *pfx_file*, const wchar_t * *pfx_password*)

Set the pfx file for signing the document with digital signature.

```
void SetPfxFile(std::wstring doc_path,
  std::wstring pfx_path, std::wstring pfx_password,
  std::wstring save_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());
  // prepare OpenSSL digital signature
  PdfDigSig* dig_sig = pdfix->CreateDigSig();
  if (!dig_sig)
    throw std::runtime_error(pdfix->GetError());
  dig_sig->SetReason(L"Testing PDFix API");
  dig_sig->SetLocation(L"Location");
  dig_sig->SetContactInfo(L"info@pdfix.net");
  if (!dig_sig->SetPfxFile(pfx_path.c_str(), pfx_password.c_str()))
    throw std::runtime_error(pdfix->GetError());
  // sign document
  if (!dig_sig->SignDoc(doc, save_path.c_str()))
    throw std::runtime_error(pdfix->GetError());
  dig_sig->Destroy();
  doc->Close();
}
```

Parameters:

pfx_file	The path of the PFX signature file.
pfx_password	The password to open the PFX file.

Returns:

true if was set successfully, false otherwise.

PdfDoc Struct Reference

PdfDoc class.
Inherited by CPdfDoc.

Public Member Functions

- bool Save (const wchar_t *path, PdfSaveFlags flags)=0
- bool SaveToStream (PsStream *stream, PdfSaveFlags flags)=0
- bool Close ()=0
  Closes a document and releases its resources. Changes are not saved.
- bool AddWatermarkFromImage (PdfWatermarkParams *params, const wchar_t *path)=0
  Adds an image-based watermark to a page range in the given document.
- int GetNumPages ()=0
  Gets the number of pages in a document.
- PdfPage * AcquirePage (int page_num)=0
- bool ReleasePage (PdfPage *page)=0
  Releases page's resources.
- int GetNumDocumentJavaScripts ()=0
  Get the number of document JavaScript name/action pairs in the document JavaScript name tree.
- int GetDocumentJavaScript (int index, wchar_t *buffer, int len)=0
  Get the document JavaScript action by it's index in th documente JavaScript name tree.
- int GetDocumentJavaScriptName (int index, wchar_t *buffer, int len)=0
  Get the document JavaScript action name by it's index in th documente JavaScript name tree.
- int GetNumCalculatedFormFields ()=0
  Get the number of calculated form fields in AcroForm calculated order (CO) which is an array.
- PdfFormField * GetCalculatedFormField (int index)=0
  Get the calculated form field from AcroForm calculation order array (CO) by index.
- int GetNumFormFields ()=0
  Get the total number of form fields in document's AcroForm Fields tree.
- PdfFormField * GetFormField (int index)=0
  Get the form field in document's AcroForm Fields tree by index.
- PdfFormField * GetFormFieldByName (const wchar_t *buffer)=0
  Get the form field in document's AcroForm Fields tree by name.
- int GetInfo (const wchar_t *key, wchar_t *buffer, int len)=0
  Gets the value of a key in a document's Info dictionary.
- bool SetInfo (const wchar_t *key, const wchar_t *buffer)=0
  Set the value of a key in a document's Info dictionary.
- PdfBookmark * GetBookmarkRoot ()=0

Detailed Description

PdfDoc class.
A PdfDoc object represents a PDF document.

Member Function Documentation

PdfPage* PdfDoc::AcquirePage (int *page_num*)

Gets a PdfPage from a document. The page is cached, so that subsequent calls on the same PDPage return The same PdfPage. The page remains in the cache as long as document exists or ReleasePage was not called. NOTE: After you are done using the page, release it using ReleasePage to release resources.

Parameters:

page_num	The page number of the page to get. The first page is 0.

Returns:

The requested page.

See also:

PdfPage::ReleasePage

bool PdfDoc::AddWatermarkFromImage (PdfWatermarkParams * *params*, const wchar_t * *path*)

Adds an image-based watermark to a page range in the given document.

```
void AddWatermarkFromImage(std::wstring doc_path, std::wstring jpg_path, std::wstring save_path) {
  // get pdfix
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());

  // apply watermark
  PdfWatermarkParams params;
  params.page_range.start_page = 0;
  params.page_range.end_page = -1;
  params.page_range.page_range_spec = kAllPages;
  params.order_top = 1;
  params.percentage_vals = 0;
  params.h_align = kHorizLeft;
  params.v_align = kVertTop;
  params.h_value = 10;
  params.v_value = 10;
  params.scale = 0.5;
  params.rotation = 0;
  params.opacity = 0.5;
  if (!doc->AddWatermarkFromImage(&params, jpg_path.c_str()))
    throw std::runtime_error(pdfix->GetError());
  doc->Save(save_path.c_str(), kSaveFull);
  doc->Close();
}
```

Parameters:

params	Structure specifying how the watermark should be added to the document.
path	Path to the image file. Only JPEG format is supported for now.

Returns:

true if the watermark was added succsessfully, false otherwise.

bool PdfDoc::Close ()

Closes a document and releases its resources. Changes are not saved.

Returns:

true if document was closed. Return false if there are any outstanding references to objects in the document. Destroy such objects first and try Close again.

See also:

PdfDoc::Save

PdfBookmark* PdfDoc::GetBookmarkRoot ()

Gets the abstract root of the document's bookmark tree. This bookmark has no representation in PDF, it only holds top level of document's bookmarks. NOTE: Call only PdfBookmark::GetNumChildren and PdfBookmark::GetChild methods for this bookmark. Other methods return false.

```
void ProcessBookmark(PdfBookmark* bmk, std::wstring indent) {
  // get title of bookmark if it's not a root bookmark
  if (bmk->GetParent()) {
    std::wstring title;
    title.resize(bmk->GetTitle(nullptr, 0));
    bmk->GetTitle((wchar_t*)title.c_str(), title.size());
    std::wcout << indent + title << std::endl;
  }
  indent += L"   ";
  int num = bmk->GetNumChildren();
  if (num > 0) {
    for (int i = 0; i < num; i++) {
      PdfBookmark* child = bmk->GetChild(i);
      ProcessBookmark(child, indent);
    }
  }
}

void GetBookmarkRoot(std::wstring path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(path.c_str(), L"");
  if (!doc)
    throw pdfix->GetError();
  PdfBookmark* parent = doc->GetBookmarkRoot();
  if (!parent)
    throw std::runtime_error("Document has no bookmarks!");
  ProcessBookmark(parent, L"");
  doc->Close();
}
```

Returns:

The document's root bookmark.

PdfFormField* PdfDoc::GetCalculatedFormField (int *index*)

Get the calculated form field from AcroForm calculation order array (CO) by index.

Parameters:

index	The index of a form field to retrieve.

Returns:

The PdfFormField object or nullptr in case of error.

See also:

PdfDoc::GetNumCalculatedFormFields
PdfDoc::GetNumFormFieldCounts
PdfDoc:GetFormField

int PdfDoc::GetDocumentJavaScript (int *index*, wchar_t * *buffer*, int *len*)

Get the document JavaScript action by it's index in th documente JavaScript name tree.

Parameters:

index	The index of a JavaScript action name to retrieve
buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdfDoc::GetNumDocumentJavaScripts
PdfDoc::GetDocumentJavaScriptName

int PdfDoc::GetDocumentJavaScriptName (int *index*, wchar_t * *buffer*, int *len*)

Get the document JavaScript action name by it's index in th documente JavaScript name tree.

Parameters:

index	The index of a JavaScript action name to retrieve
buffer	(filled by method) If the buffer is null function returns required length of string.

len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdfDoc::GetNumDocumentJavaScripts
PdfDoc::GetDocumentJavaScript

PdfFormField* PdfDoc::GetFormField (int *index*)

Get the form field in document's AcroForm Fields tree by index.

Returns:

The PdfFormField object or nullptr in case of error.

See also:

PdfDoc::GetNumCalculatedFormFields
PdfDoc::GetCalculatedFormField
PdfDoc::GetNumFormFieldCounts

PdfFormField* PdfDoc::GetFormFieldByName (const wchar_t * *buffer*)

Get the form field in document's AcroForm Fields tree by name.

Returns:

The PdfFormField object or nullptr in case of error.

See also:

PdfDoc::GetNumCalculatedFormFields
PdfDoc::GetCalculatedFormField
PdfDoc::GetNumFormFieldCounts

int PdfDoc::GetInfo (const wchar_t * *key*, wchar_t * *buffer*, int *len*)

Gets the value of a key in a document's Info dictionary.

Parameters:

key	The name of the Info dictionary key whose value is obtained.
buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfDoc::GetNumCalculatedFormFields ()

Get the number of calculated form fields in AcroForm calculated order (CO) which is an array.

Returns:

Number of calculated form fields in the document.

See also:

PdfDoc::GetCalculatedFormField
PdfDoc::GetNumFormFieldCounts
PdfDoc:GetFormField

int PdfDoc::GetNumDocumentJavaScripts ()

Get the number of document JavaScript name/action pairs in the document JavaScript name tree.

Returns:

Number document name/action pairs in the document's JavaScript name tree.

See also:

PdfDoc::GetDocumentJavaScript
PdfDoc::GetDocumentJavaScriptName

int PdfDoc::GetNumFormFields ()

Get the total number of form fields in document's AcroForm Fields tree.

Returns:

Number of form fields in the document.

See also:

PdfDoc::GetNumCalculatedFormFields
PdfDoc::GetCalculatedFormField
PdfDoc:GetFormField

int PdfDoc::GetNumPages ()

Gets the number of pages in a document.

Returns:

Number of pages in the document.

bool PdfDoc::ReleasePage (PdfPage * *page*)

Releases page's resources.

Parameters:

page	The page to release.

Returns:

true if page was released, false otherwise.

See also:

PdfPage::AcquirePage

bool PdfDoc::Save (const wchar_t * *path*, PdfSaveFlags *flags*)

Saves a document to disk. NOTE: You must call PdfDoc::Close to release resources.

Parameters:

path	The path to which the file is saved.
flags	A PdfSaveFlags value. If kSaveIncremental is specified in flags, then path should be NULL. If kSaveFull is specified and path is the same as the file's original path, the new file is saved to a file system-determined temporary path, then the old file is deleted and the new file is renamed to path.

See also:

PdfDoc::Close

bool PdfDoc::SaveToStream (PsStream * *stream*, PdfSaveFlags *flags*)

Saves a document to a stream. NOTE: You must call PdfDoc::Close to release resources.

Parameters:

stream	The stream to which the file is saved.
flags	A PdfSaveFlags value. If kSaveIncremental is specified in flags, then path should be NULL. If kSaveFull is specified and path is the same as the file's original path, the new file is saved to a file system-determined temporary path, then the old file is deleted and the new file is renamed to path.

See also:

PdfDoc::Close, Pdfix::CreateStream

bool PdfDoc::SetInfo (const wchar_t * *key*, const wchar_t * *buffer*)

Set the value of a key in a document's Info dictionary.

Parameters:

key	The name of the Info dictionary key whose value is obtained.
buffer	String value to be set for the specific Info dictionary entry.

Returns:

true if optaining the font state was succsessfull, false otherwise.

PdfFont Struct Reference

PdfFont class.
Inherited by CPdfFont.

Public Member Functions

- int GetFontName (wchar_t *buffer, int len)=0
  Gets the name of a font.
- int GetFaceName (wchar_t *buffer, int len)=0
  Gets the face of a font.
- void GetFontState (PdfFontState *font_state)=0
  Gets the font state of a font.
- int GetSystemFontName (wchar_t *buffer, int len)=0
  Gets the name of a font which is a system replacement for the font.
- PdfFontCharset GetSystemFontCharset ()=0
  Gets the charset of a font which is a system replacement for the font.
- bool GetSystemFontBold ()=0
  Gets the the system font bold flag.
- bool GetSystemFontItalic ()=0
  Gets the the system font italic flag.

Detailed Description

PdfFont class.
PdfFont class.

Member Function Documentation

int PdfFont::GetFaceName (wchar_t * *buffer*, int *len*)

Gets the face of a font.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfFont::GetFontName (wchar_t * *buffer*, int *len*)

Gets the name of a font.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

void PdfFont::GetFontState (PdfFontState * *font_state*)

Gets the font state of a font.

Parameters:

font_state	(filled by method) Pointer to font state structure to be filled in.

Returns:

true if optaining the font state was succsessfull, false otherwise.

bool PdfFont::GetSystemFontBold ()

Gets the the system font bold flag.

Returns:

true is font is bold, false otherwise.

PdfFontCharset PdfFont::GetSystemFontCharset ()

Gets the charset of a font which is a system replacement for the font.

Returns:

Number of charset of a font.

bool PdfFont::GetSystemFontItalic ()

Gets the the system font italic flag.

Returns:

true is font is italic, false otherwise.

int PdfFont::GetSystemFontName (wchar_t * *buffer*, int *len*)

Gets the name of a font which is a system replacement for the font.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

PdfFormField Struct Reference

PdfFormField class.

Inherited by CPdfFormField.

Public Member Functions

- PdfFieldType GetType ()=0
  Gets the type of field.
- PdfFieldFlags GetFlags ()=0
  Gets the form field's flags.
- int GetValue (wchar_t *buffer, int len)=0
  Gets the field’s value as string.
- bool SetValue (const wchar_t *buffer)=0
  Sets the field’s value as string. Multiple values should be comma-separated.
- int GetDefaultValue (wchar_t *buffer, int len)=0
  Gets the field’s default value as string.
- int GetFullName (wchar_t *buffer, int len)=0
  Gets the field’s full name within the document AcroForm field tree.
- int GetTooltip (wchar_t *buffer, int len)=0
  Gets the field’s tooltip.
- int GetOptionCount ()=0
  Gets the number of elements in the Opt array.
- int GetOptionValue (int index, wchar_t *buffer, int len)=0
  Gets the field’s option value.
- int GetOptionCaption (int index, wchar_t *buffer, int len)=0
  Gets the field’s option caption.
- PdfAction * GetAction ()=0
  Gets a field's action object.
- PdfAction * GetAAction (PdfActionEventType event)=0
  Gets a field's additional action object.
- int GetMaxLength ()=0
  Gets maximum length of the field’s text, in characters.
- int GetWidgetExportValue (PdfAnnot *annot, wchar_t *buffer, int len)=0
  Gets the field’s widget export value.

Detailed Description

PdfFormField class.
PdfFormField object represents interactive form dictionary that shall be referenced from the AcroForm entry in the document catalogue

Member Function Documentation

PdfAction* PdfFormField::GetAAction (PdfActionEventType *event*)

Gets a field's additional action object.

Parameters:

event	The event which additional action to get.

Returns:

The annotation's additional action object or nullptr if annotation does not have an action for specified event type.

See also:

GetAction

PdfAction* PdfFormField::GetAction ()

Gets a field's action object.

Returns:

The annotation's action object or nullptr if annotation does not have an action.

See also:

GetAAction

int PdfFormField::GetDefaultValue (wchar_t * *buffer*, int *len*)

Gets the field's default value as string.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

PdfFieldFlags PdfFormField::GetFlags ()

Gets the form field's flags.

Returns:

The form field's flags.

int PdfFormField::GetFullName (wchar_t * *buffer*, int *len*)

Gets the field’s full name within the document AcroForm field tree.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfFormField::GetMaxLength ()

Gets maximum length of the field’s text, in characters.

Returns:

The maximum number of characters.

int PdfFormField::GetOptionCaption (int *index*, wchar_t * *buffer*, int *len*)

Gets the field’s option caption.

Parameters:

index	The index of option to retrieve.
buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfFormField::GetOptionCount ()

Gets the number of elements in the Opt array.

Returns:

Number of field's options.

int PdfFormField::GetOptionValue (int *index*, wchar_t * *buffer*, int *len*)

Gets the field’s option value.

Parameters:

index	The index of option to retrieve.
buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfFormField::GetTooltip (wchar_t * *buffer*, int *len*)

Gets the field’s tooltip.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

PdfFieldType PdfFormField::GetType ()

Gets the type of field.

Returns:

The form field type.

int PdfFormField::GetValue (wchar_t * *buffer*, int *len*)

Gets the field's value as string.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfFormField::GetWidgetExportValue (PdfAnnot * *annot*, wchar_t * *buffer*, int *len*)

Gets the field's widget export value.

Parameters:

annot	The widget annotation which export value is to be retrieved.
buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

bool PdfFormField::SetValue (const wchar_t * *buffer*)

Sets the field's value as string. Multiple values should be comma-separated.

```
void SetFormFieldValue(std::wstring doc_path, std::wstring save_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
```

```
    throw std::runtime_error(pdfix->GetError());
  PdfFormField* field = doc->GetFormFieldByName(L"Text1");
  if (field) {
    std::wstring value;
    value.resize(field->GetValue(nullptr, 0));
    field->GetValue((wchar_t*)value.c_str(), value.size());
    if (value.length() == 0)
      value = L"new value";
    else
      std::reverse(std::begin(value), std::end(value));
    field->SetValue(value.c_str());
  }
  doc->Save(save_path.c_str(), kSaveFull);
  doc->Close();
}
```

Parameters:

buffer	The new form field string value

Returns:

true if succeeded, false otherwise.

PdfImage Struct Reference

PdfImage class.
Inherited by CPdfImage.

Public Member Functions

- bool Save (const wchar_t *path, PdfImageFormat format)=0
  Saves the image data into a file.
- bool SaveRect (PdfDevRect *dev_rect, const wchar_t *path, PdfImageFormat format)=0
  Saves a clip of the image data into a file.

Detailed Description

PdfImage class.
PdfImage contains an image data in an internal format.

Member Function Documentation

bool PdfImage::Save (const wchar_t * *path*, PdfImageFormat *format*)

Saves the image data into a file.

Parameters:

path	Path where to save image data in requested format.
format	PdfImageFormat.

Returns:

true if succeeded, false otherwise.

See also:

PdfImageFormat

bool PdfImage::SaveRect (PdfDevRect * *dev_rect*, const wchar_t * *path*, PdfImageFormat *format*)

Saves a clip of the image data into a file.

Parameters:

dev_rect	Clip area of the image data that needs to be saved.
path	Path where to save image data in requested format.
format	PdfImageFormat.

Returns:

true if succeeded, false otherwise.

See also:

PdfImageFormat

Pdfix Struct Reference

Pdfix class.
Inherited by CPdfix.

Public Member Functions

- void Destroy ()=0
  Destroys Pdfix resources.
- bool Authorize (const wchar_t *email, const wchar_t *serial_number)=0

Authorizes Pdfix.

- PdfErrorType GetErrorType ()=0
- const char * GetError ()=0
- int GetVersionMajor ()=0
- int GetVersionMinor ()=0
- int GetVersionPatch ()=0
- PdfDoc * OpenDoc (const wchar_t *path, const wchar_t *password)=0
- PdfDoc * OpenDocFromStream (PsStream *stream, const wchar_t *password)=0
- PdfDigSig * CreateDigSig ()=0
- PdfCustomDigSig * CreateCustomDigSig ()=0
- PsRegex * CreateRegex ()=0
- PsStream * CreateStream ()=0
- bool RegisterEvent (PdfEventType type, _callback_ PdfEventProc proc, void *data)=0
  Registers a user-supplied procedure to call when the specified event occurs.
- bool UnregisterEvent (PdfEventType type, PdfEventProc proc, void *data)=0

Detailed Description

Pdfix class.
Pdfix loads and unloads library. It initialized all necessary resources and also takes care about releasing it.

Member Function Documentation

bool Pdfix::Authorize (const wchar_t * *email*, const wchar_t * *serial_number*)

Authorizes Pdfix.

Returns:

true if Pdfix was authorized successfully, false otherwise.

PdfCustomDigSig* Pdfix::CreateCustomDigSig ()

Creates a new PdfCustomDigSig object. Call PdfDigSig::Destroy method to release resources.

Returns:

Initialized PdfCustomDigSig object.

See also:

PdfDigSig::Destroy

PdfDigSig* Pdfix::CreateDigSig ()

Creates a new PdfDigSig object. Call PdfDigSig::Destroy method to release resources.

Returns:

Initialized PdfDigSig object.

See also:

PdfDigSig::Destroy

PsRegex* Pdfix::CreateRegex ()

Creates a new PsRegex object. Call PsRegex::Destroy to release all regex resources.

Returns:

Initialized PsRegex object.

See also:

PsRegex::Destroy

PsStream* Pdfix::CreateStream ()

Creates a new PsStream object. It's a data stream that may be a buffer in memory or a file. Call PsStream::Destroy to release all stream resources.

Returns:

Initialized PsStream object.

See also:

PsStream::Destroy, Pdfix::OpenDocFromStream, Pdfix::CreateImageFromStream

void Pdfix::Destroy ()

Destroys Pdfix resources.

See also:

Pdfix::CreatePdfix

const char* Pdfix::GetError ()

Returns the latest error message from the library. The error message is set each time, when any library method fails.

Returns:

The last error, empty string otherwise.

PdfErrorType Pdfix::GetErrorType ()

Returns the latest error type from the library. The error type is set each time, when any library method fails.

Returns:

The last error type.

int Pdfix::GetVersionMajor ()

Returns the major version. This is the first integer in a version number and is increased whenever significant changes are made.

Returns:

The major version number.

int Pdfix::GetVersionMinor ()

Returns the minor version. This is the second integer in a compound version number. This is normally set to 0 after each major release and increased whenever smaller features or significant bug fixes have been added.

Returns:

The minor version number.

int Pdfix::GetVersionPatch ()

Returns the patch version. The (optional) third integer is the patch number, sometimes also called the revision number. Changes in patch number should imply no change to the actual API interface, only changes to the behavior of the API.

Returns:

The patch version number.

PdfDoc* Pdfix::OpenDoc (const wchar_t * *path*, const wchar_t * *password*)

Opens the specified document. If the document is already open, returns a reference to the already opened PdfDoc. NOTE: You must call PdfDoc::Close once for every successful open.

Parameters:

path	Path to the file.
password	File password.

Returns:

The newly created document or null.

See also:

Close

PdfDoc* Pdfix::OpenDocFromStream (PsStream * *stream*, const wchar_t * *password*)

Opens the specified document from memory. If the document is already open, returns a reference to the already opened PdfDoc. You must call PdfDoc::Close once for every successful open.

Parameters:

stream	PsStream object.
password	File password.

Returns:

The newly created document or null.

See also:

Close

bool Pdfix::RegisterEvent (PdfEventType *type*, _callback_ PdfEventProc *proc*, void * *data*)

Registers a user-supplied procedure to call when the specified event occurs.

```
void DocDidOpenCallback(PdfEventParams* event, void* data) {
  if (event->type != kEventDocDidOpen)
    throw std::runtime_error("This should never happen!");
  if (event->doc == nullptr)
    throw std::runtime_error("This should never happen!");
  std::cout << "Document was opened!" << std::endl;
  // get title of currently opened document
  std::wstring title;
  title.resize(event->doc->GetInfo(L"Title", nullptr, 0));
  event->doc->GetInfo(L"Title", (wchar_t*)title.c_str(), title.size());
  std::wcout << title << std::endl;
}

void DocWillCallback(PdfEventParams* event, void* data) {
  switch (event->type) {
  case kEventDocWillClose:
    std::cout << "Document will be closed!" << std::endl;
    break;
  case kEventDocWillSave:
    std::cout << "Document will be saved!" << std::endl;
    break;
  }
}

void RegisterEvent(std::wstring doc_path) {
  // get pdfix
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  // add events
  pdfix->RegisterEvent(kEventDocDidOpen, &DocDidOpenCallback, nullptr);
  pdfix->RegisterEvent(kEventDocWillClose, &DocWillCallback, nullptr);
  pdfix->RegisterEvent(kEventDocWillSave, &DocWillCallback, nullptr);
  // open document
  PdfDoc* doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());
  doc->Close();
}
```

Parameters:

type	The event type.
proc	A user-supplied callback to call when the event occurs.
data	A pointer to user-supplied data to pass to proc each time it is called.

Returns:

true if event was registered, false otherwise.

bool Pdfix::UnregisterEvent (PdfEventType *type*, PdfEventProc *proc*, void * *data*)

Unregisters a user-supplied procedure to call when the specified event occurs. To un-register, you must use same type, proc and data that were used when the event was registered using Pdfix::RegisterEvent.

Parameters:

type	The registered event type.
proc	A registered user-supplied callback.
data	A pointer to registered user-supplied data.

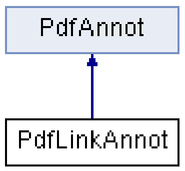
Returns:

true if event was registered, false otherwise.

PdfLinkAnnot Struct Reference

PdfLinkAnnot class.

Inheritance diagram for PdfLinkAnnot:



Public Member Functions

- int GetNumQuads ()=0
  Gets the number of quads for the link.
- void GetQuad (int index, PdfQuad *quad)=0
- bool AddQuad (PdfQuad *quad)=0
  Adds a new quad to the link annot.
- bool RemoveQuad (int index)=0
  Removes a quad with the specified index.
- PdfAction * GetAction ()=0
  Gets an link's action object.

Detailed Description

PdfLinkAnnot class.
A link annotation represents either a hypertext link to a destination elsewhere in the document or an action to be performed.

Member Function Documentation

bool PdfLinkAnnot::AddQuad (PdfQuad * *quad*)

Adds a new quad to the link annot.

Parameters:

quad	Pointer to PdfQuad to add.

Returns:

true if quad was added sucessfully, false otherwise.

See also:

PdfLinkAnnot::GetNumQuads

PdfAction* PdfLinkAnnot::GetAction ()

Gets an link's action object.

Returns:

The link's action object or nullptr if link does not have an action.

int PdfLinkAnnot::GetNumQuads ()

Gets the number of quads for the link.

Returns:

Number of quads.

See also:

PdfLinkAnnot::GetQuad

void PdfLinkAnnot::GetQuad (int *index*, PdfQuad * *quad*)

Gets the requested quad. The coordinates of the quadrilaterals are in default user space that comprise the region in which the link should be activated.

Parameters:

index	Index of an link quad to retrieve.
quad	(filled by the method) Pointer to PdfQuad structure to fill.

See also:

PdfLinkAnnot::GetNumQuads

bool PdfLinkAnnot::RemoveQuad (int *index*)

Removes a quad with the specified index.

Parameters:

index	The index of the quad to remove.

Returns:

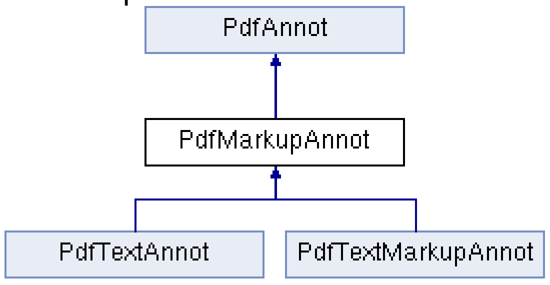
true if quad was removed, false otherwise.

See also:

PdfLinkAnnot::GetNumQuads

PdfMarkupAnnot Struct Reference

PdfMarkupAnnot class.
Inheritance diagram for PdfMarkupAnnot:



Public Member Functions

- int GetContents (wchar_t *buffer, int len)=0
- bool SetContents (const wchar_t *buffer)=0
- int GetAuthor (wchar_t *buffer, int len)=0

  Get the author of the markup annotation.
- bool SetAuthor (const wchar_t *buffer)=0

  Set the author of the markup annotation.
- int GetNumReplies ()=0

  Both annotations must be on the same page of the document.
- PdfAnnot * GetReply (int index)=0

  Both annotations must be on the same page of the document.
- PdfAnnot * AddReply (const wchar_t *author, const wchar_t *text)=0

  Adds a new reply to the markup annotation.

Detailed Description

PdfMarkupAnnot class.
Markup annotations represent a markup annotation in a pdf document.

Member Function Documentation

PdfAnnot* PdfMarkupAnnot::AddReply (const wchar_t * *author*, const wchar_t * *text*)

Adds a new reply to the markup annotation.

```
void AddCommentWithReply(std::wstring doc_path, std::wstring save_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());
  PdfPage* page = doc->AcquirePage(0);
  PdfRect crop_box;;
  page->GetCropBox(&crop_box);
  // place annotation to the middle of the page
  PdfRect annot_rect;
  annot_rect.left = (crop_box.right + crop_box.left) / 2. - 10;
  annot_rect.bottom = (crop_box.top + crop_box.bottom) / 2. - 10;
  annot_rect.right = (crop_box.right + crop_box.left) / 2. + 10;
  annot_rect.top = (crop_box.top + crop_box.bottom) / 2. + 10;
  PdfTextAnnot* annot = page->AddTextAnnot(-1, &annot_rect);
  annot->SetAuthor(L"Peter Brown");
  annot->SetContents(L"This is my comment.");
  annot->AddReply(L"Mark Fish", L"This is some reply.");
  doc->ReleasePage(page);
  doc->Save(save_path.c_str(), kSaveFull);
  doc->Close();
}
```

Parameters:

author	The author of the reply.
text	The content of the reply to add.

Returns:

Requested PdfAnnot that is reply to the current annotation or nullptr in case of error.

See also:

PdfMarkupAnnot::AddReply

int PdfMarkupAnnot::GetAuthor (wchar_t * *buffer*, int *len*)

Get the author of the markup annotation.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdfMarkupAnnot::GetAuthor

int PdfMarkupAnnot::GetContents (wchar_t * *buffer*, int *len*)

Get the contents of the markup annotation. It's a text to be displayed for the annotation or, if this type of annotation does not display text, an alternate description of the annotation’s contents in human - readable form. In either case, this text is useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes.

Parameters:

buffer	(filled by method) if the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

See also:

PdfMarkupAnnot::GetContents

int PdfMarkupAnnot::GetNumReplies ()

Both annotations must be on the same page of the document.

Gets the requested reply. Reply is a reference to another PdfAnnot object that was created

Returns:

Number of replies.

See also:

PdfMarkupAnnot::GetNumReplies

PdfAnnot* PdfMarkupAnnot::GetReply (int *index*)

Both annotations must be on the same page of the document.

Gets the requested reply. Reply is a reference to another PdfAnnot object that was created

Returns:

Requested PdfAnnot that is reply to the current annotation.

See also:

PdfMarkupAnnot::GetReply

bool PdfMarkupAnnot::SetAuthor (const wchar_t * *buffer*)

Set the author of the markup annotation.

Parameters:

buffer	The content string to be set.

Returns:

true if the author was set, false otherwise.

See also:

PdfMarkupAnnot::SetAuthor

bool PdfMarkupAnnot::SetContents (const wchar_t * *buffer*)

Set the contents of the markup annotation. It's a text to be displayed for the annotation or, if this type of annotation does not display text, an alternate description of the annotation’s contents in human - readable form. In either case, this text is useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes.

Parameters:

buffer	The content string to be set.

Returns:

true if the content was set, false otherwise.

See also:

PdfMarkupAnnot::SetContents

PdfPage Struct Reference

PdfPage class.
Inherited by CPdfPage.

Public Member Functions

- void GetCropBox (PdfRect *crop_box)=0
  Gets the crop box for a page. The crop box is the region of the page to display and print.
- void GetMediaBox (PdfRect *media_box)=0
- PdfRotate GetRotate ()=0
  Gets the rotation value for a page.
- void GetDefaultMatrix (PdfMatrix *matrix)=0
- int GetNumber ()=0
  Gets the page number for the specified page.
- PdePageMap * AcquirePageMap (PdfPageMapParams *params, _callback_ PdfCancelProc cancel_proc, void *cancel_data)=0
- bool ReleasePageMap ()=0
- PdfPageView * AcquirePageView (double zoom, PdfRotate rotate)=0
- bool ReleasePageView (PdfPageView *page_view)=0
- int GetNumAnnots ()=0
- PdfAnnot * GetAnnot (int index)=0
  Gets the requested annotation on the page.
- bool RemoveAnnot (int index, PdfRemoveAnnotFlags flags)=0
- PdfTextAnnot * AddTextAnnot (int index, PdfRect *rect)=0
  Adds a text annotation to the page.
- PdfLinkAnnot * AddLinkAnnot (int index, PdfRect *rect)=0
  Adds a link annotation to the page.
- PdfTextMarkupAnnot * AddTextMarkupAnnot (int index, PdfRect *rect, PdfAnnotSubtype subtype)=0
  Adds a text markup annotation to the page.
- int GetNumAnnotsAtPoint (PdfPoint *point)=0
  Gets the number of annotations that reside under the given point.
- PdfAnnot * GetAnnotAtPoint (PdfPoint *point, int index)=0

Gets the requested annotation that resides under the given point.

- int GetNumAnnotsAtRect (PdfRect *rect)=0
- PdfAnnot * GetAnnotAtRect (PdfRect *rect, int index)=0
  Gets the requested annotation that resides under the given rectangle.

Detailed Description

PdfPage class.
A PdfPage is a page in a document. Among other associated objects, a page contains PdePageMap, that represents the page content.

Member Function Documentation

PdePageMap* PdfPage::AcquirePageMap (PdfPageMapParams * *params*, _callback_ PdfCancelProc *cancel_proc*, void * *cancel_data*)

Generates a PdePageMap from the PdfPage's elements. The PdePageMap is cached, so that subsequent calls on the same PDPage return the same PdePageMap. The PdePageMap remains in the cache as long as page exists or ReleasePageMap was not called. Call ReleasePageMap to release pagemap resources if necessary.

Parameters:

params	Page map parameters that allow modify the page map algorithm.
cancel_proc	Callback to check for canceling operations. A CancelProc is typically passed to some method that takes a long time to complete. At frequent intervals, the method calls the CancelProc. If it returns true, then the method cancels its operation; if false, it continues.
cancel_data	Pointer to client data for the cancel procedure.

Returns:

PdePageMap for the current page.

See also:

PdfPage::ReleasePageMap

PdfPageView* PdfPage::AcquirePageView (double *zoom*, PdfRotate *rotate*)

Generates a PdfPageView from the PdfPage's elements. The PdfPageView is cached, so that subsequent calls on the same PDPage and same input parameters return the same PdePageMap. The PdePageMap remains in the cache as long as page exists or ReleasePageView was not called. Call ReleasePageView to release pagemap resources if necessary.

Parameters:

zoom	Expected zoom of the page view.
rotate	Expected rotation of the page view.

Returns:

An acquired page view or null.

See also:

PdfPage::ReleasePageView

PdfLinkAnnot* PdfPage::AddLinkAnnot (int *index*, PdfRect * *rect*)

Adds a link annotation to the page.

Parameters:

index	Where to add the annotation in the page's annotation array.
rect	Pointer to a rectangle specifying the annotation's bounds, specified in user space coordinates. If it's null, use PdfLinkAnnot::AddQuad to specify the size and location of an annotation on its page.

Returns:

The newly created PdfLinkAnnot.

See also:

PdfPage::AddTextAnnot, PdfLinkAnnot::AddQuad

PdfTextAnnot* PdfPage::AddTextAnnot (int *index*, PdfRect * *rect*)

Adds a text annotation to the page.

```
void AddCommentWithReply(std::wstring doc_path, std::wstring save_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());
  PdfPage* page = doc->AcquirePage(0);
  PdfRect crop_box;;
  page->GetCropBox(&crop_box);
  // place annotation to the middle of the page
  PdfRect annot_rect;
  annot_rect.left = (crop_box.right + crop_box.left) / 2. - 10;
  annot_rect.bottom = (crop_box.top + crop_box.bottom) / 2. - 10;
  annot_rect.right = (crop_box.right + crop_box.left) / 2. + 10;
  annot_rect.top = (crop_box.top + crop_box.bottom) / 2. + 10;
  PdfTextAnnot* annot = page->AddTextAnnot(-1, &annot_rect);
  annot->SetAuthor(L"Peter Brown");
  annot->SetContents(L"This is my comment.");
  annot->AddReply(L"Mark Fish", L"This is some reply.");
  doc->ReleasePage(page);
  doc->Save(save_path.c_str(), kSaveFull);
  doc->Close();
}
```

Parameters:

index	Where to add the annotation in the page's annotation array. Passing a value of -1 adds the annotation to the end of the array (this is generally what you should do unless you have a need to place the annotation at a special location in the array). Passing a value of 0 adds the

	annotation to the beginning of the array.
rect	Pointer to a rectangle specifying the annotation's bounds, specified in user space coordinates.

Returns:

The newly created PdfTextAnnot.

See also:

PdfPage::GetNumAnnots

PdfTextMarkupAnnot* PdfPage::AddTextMarkupAnnot (int *index*, PdfRect * *rect*, PdfAnnotSubtype *subtype*)

Adds a text markup annotation to the page.

Parameters:

subtype	Define a subtype of the text markup annotation. Must be one of kAnnotHighlight, kAnnotUnderline, kAnnotSquiggly, kAnnotStrikeOut.
index	Where to add the annotation in the page's annotation array.
rect	Pointer to a rectangle specifying the annotation's bounds, specified in user space coordinates. If it's null, use PdfTextMarkupAnnot::AddQuad to specify the size and location of an annotation on its page.

Returns:

The newly created PdfTextMarkupAnnot.

See also:

PdfPage::AddTextAnnot, PdfTextMarkupAnnot::AddQuad

PdfAnnot* PdfPage::GetAnnot (int *index*)

Gets the requested annotation on the page.

Parameters:

index	The index of annotation to obtain.

Returns:

Requested annotation object.

See also:

PdfPage::GetNumAnnots

PdfAnnot* PdfPage::GetAnnotAtPoint (PdfPoint * *point*, int *index*)

Gets the requested annotation that resides under the given point.

Parameters:

point	The point to test.

index	(filled by the method) Index of annotation to obtain.

Returns:

Pointer to the requested annotation, nullptr in a case of error.

See also:

PdfPage::GetAnnotAtRect

PdfAnnot* PdfPage::GetAnnotAtRect (PdfRect * *rect*, int *index*)

Gets the requested annotation that resides under the given rectangle.

Parameters:

rect	The rectangle to test.
index	(filled by the method) Index of annotation to obtain.

Returns:

Pointer to the requested annotation, nullptr in a case of error.

See also:

PdfPage::GetAnnotAtRect

void PdfPage::GetCropBox (PdfRect * *crop_box*)

Gets the crop box for a page. The crop box is the region of the page to display and print.

Parameters:

crop_box	(filled by the method) Pointer to a rectangle specifying the page's crop box, specified in user space coordinates.

void PdfPage::GetDefaultMatrix (PdfMatrix * *matrix*)

Gets the matrix that transforms user space coordinates to rotated and cropped coordinates. The origin of this space is the bottom - left of the rotated, cropped page. Y is increasing.

Parameters:

matrix	(filled by the method) Pointer to the default transformation matrix.

void PdfPage::GetMediaBox (PdfRect * *media_box*)

Gets the media box for a page. The media box is the 'natural size' of the page, for example, the dimensions of an A4 sheet of paper.

Parameters:

media_box	(filled by the method) Pointer to a rectangle specifying the page's media box, specified in user space coordinates.

int PdfPage::GetNumAnnots ()

Gets the number of annotations on a page. Annotations associated with pop-up windows (such as strikeouts) are counted as two annotations. Widget annotations(form fields) are included in the count.

Returns:

The number of annotations on a page.

See also:

PdfPage::GetAnnot

int PdfPage::GetNumAnnotsAtPoint (PdfPoint * *point*)

Gets the number of annotations that reside under the given point.

Parameters:

point	The point to test.

Returns:

Number of annotations under the given point.

See also:

PdfPage::GetAnnotAtPoint

int PdfPage::GetNumAnnotsAtRect (PdfRect * *rect*)

Gets the number of annotations that reside under the given rectangle. It returns each annotation that have intersection the given rectangle.

Parameters:

rect	The rectangle to test.

Returns:

Number of annotations under the given rectangle.

See also:

PdfPage::GetAnnotAtPoint

int PdfPage::GetNumber ()

Gets the page number for the specified page.

Returns:

The page within the document. The first page is 0.

PdfRotate PdfPage::GetRotate ()

Gets the rotation value for a page.

Returns:

Rotation value for the given page. Must be one of the PdfRotate values.

See also:

PdfRotate

bool PdfPage::ReleasePageMap ()

Releases the pagemap resources at the current page. NOTE: The caller can call ReleasePageMap to optimize a memory handling. Otherwise the page is responsible for freeing PdePageMap resources.

Returns:

true if succeeded, false otherwise.

See also:

PdfPage::AcquirePageMap

bool PdfPage::ReleasePageView (PdfPageView * *page_view*)

Releases the page view resources. NOTE: The caller can call ReleasePageView to optimize a memory handling. Otherwise the page is responsible for freeing PdfPageViews resources.

Parameters:

page_view	The page view to delete.

Returns:

true if succeeded, false if page view with specific params was not found.

See also:

PdfPage::AcquirePageView

bool PdfPage::RemoveAnnot (int *index*, PdfRemoveAnnotFlags *flags*)

Removes an annotation from the specified page. Annotations are stored in arrays, which are automatically compressed when an annotation is removed. For this reason, if you use a loop in which you remove annotations, structure the code so the loop processes from the highest to the lowest index.

```
void RemoveCommentsWithReply(std::wstring doc_path, std::wstring save_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());
  PdfPage* page = doc->AcquirePage(0);
  // remove from first highlight annot with it's popup and all replies
  for (auto i = 0; i < page->GetNumAnnots(); i++) {
    PdfAnnot* annot = page->GetAnnot(i);
    if (annot->GetSubtype() == kAnnotHighlight) {
      page->RemoveAnnot(i, kRemoveAnnotPopup | kRemoveAnnotReply);
      break;
    }
  }
  doc->ReleasePage(page);
  doc->Save(save_path.c_str(), kSaveFull);
  doc->Close();
}
```

Parameters:

index	The index of annotation to remove.
flags	PdfRemoveAnnotFlags to specify what other connected annotations will be removed.

Returns:

true if annotation was removed, false otherwise.

See also:

PdfPage::GetNumAnnots

PdfPageView Struct Reference

PdfPageView class.
Inherited by CPdfPageView.

Public Member Functions

- int GetDeviceWidth ()=0

Returns a width of the page view in device space coordinates.

- int GetDeviceHeight ()=0

  Returns a height of the page view in device space coordinates.
- bool DrawPage (PdfPageRenderParams *params, _callback_ PdfCancelProc cancel_proc, void *cancel_data)=0
- PdfImage * GetImage ()=0
- void RectToDevice (PdfRect *rect, PdfDevRect *dev_rect)=0
- void PointToDevice (PdfPoint *point, PdfDevPoint *dev_point)=0

  Transforms a point's coordinates from user space to device space.

Detailed Description

PdfPageView class.
A PdfPageView has methods to display the contents of a document page.

Member Function Documentation

bool PdfPageView::DrawPage (PdfPageRenderParams * *params*, _callback_ PdfCancelProc *cancel_proc*, void * *cancel_data*)

Draws the contents of a page into the page view PdfImage. This method just draws a bitmap. Provides control over the rendering with respect to PdfPageRenderParams. The PdfImage remains in the cache as the page view class exists or next PdfPageViewDrawPage method is called.

```
void DrawPage(std::wstring doc_path, std::wstring image_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
    throw std::runtime_error(pdfix->GetError());
  // render first page to jpg image
  PdfPage* page = doc->AcquirePage(0);
  if (!page)
    throw std::runtime_error(pdfix->GetError());
  PdfPageView* page_view = page->AcquirePageView(2, kRotate0);
  if (!page_view)
    throw std::runtime_error(pdfix->GetError());
  PdfPageRenderParams params;
  params.render_flags = kRenderAnnot | kRenderGrayscale;
  page_view->DrawPage(&params, nullptr, nullptr);
  PdfImage* image = page_view->GetImage();
  if (!image)
    throw std::runtime_error(pdfix->GetError());
  image->Save(image_path.c_str(), kImageFormatJpg);
  page->ReleasePageView(page_view);     // not necessary
  page->ReleasePageMap();               // not necessary
  doc->ReleasePage(page);               // not necessary
  doc->Close();
}
```

Parameters:

params	Rendering parameters.
cancel_proc	Callback to check for canceling operations. A CancelProc is typically

	passed to some method that takes a long time to complete. At frequent intervals, the method calls the CancelProc. If it returns true, then the method cancels its operation; if false, it continues.
cancel_data	Pointer to client data for the cancel procedure.

Returns:

true if page was rendered, false otherwise.

See also:

PdfPageView::GetImage

int PdfPageView::GetDeviceHeight ()

Returns a height of the page view in device space coordinates.

Returns:

A page view height.

See also:

PdfPageView::GetDeviceWidth

int PdfPageView::GetDeviceWidth ()

Returns a width of the page view in device space coordinates.

Returns:

A page view width.

See also:

PdfPageView::GetDeviceHeight

PdfImage* PdfPageView::GetImage ()

Gets the image data for a page view. You should never depend on these objects lasting the lifetime of the document. You should extract the information you need from the object immediately and refer to it no further in your code. NOTE: Do not destroy the returned PdfImage when done with it.

Returns:

Acquired Image data for page view. Returns null if there are no image data.

See also:

PdfPageView::DrawPage

void PdfPageView::PointToDevice (PdfPoint * *point*, PdfDevPoint * *dev_point*)

Transforms a point's coordinates from user space to device space.

Parameters:

point	Pointer to the point whose coordinates are transformed, specified in user space coordinates.

dev_point	(filled by the method) Pointer to a point containing the device space coordinates corresponding to point.

See also:

PdfPageView::RectToDevice

void PdfPageView::RectToDevice (PdfRect * *rect*, PdfDevRect * *dev_rect*)

Transforms a rectangle's coordinates from user space to device space. The resulting AVRect will be normalized, that is, left < right and top < bottom.

Parameters:

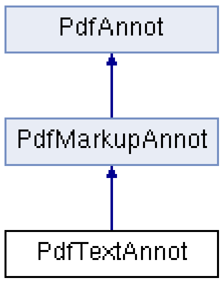
rect	Pointer to the rectangle whose coordinates are transformed, specified in user space coordinates.
dev_rect	(filled by the method) Pointer to a rectangle containing the device space coordinates corresponding to rect.

See also:

PdfPageView::PointToDevice

PdfTextAnnot Struct Reference

PdfTextAnnot class.
Inheritance diagram for PdfTextAnnot:



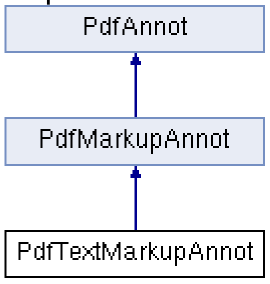
Additional Inherited Members

Detailed Description

PdfTextAnnot class.
A text annotation represents a “sticky note” attached to a point in the PDF document. When closed, the annotation appears as an icon; when open, it displays a pop-up window containing the text of the note in a font and size chosen by the viewer application. Text annotations do not scale and rotate with the page.

PdfTextMarkupAnnot Struct Reference

PdfTextMarkupAnnot class.
Inheritance diagram for PdfTextMarkupAnnot:



Public Member Functions

- int GetNumQuads ()=0
  Gets the number of quads for the annotation.
- void GetQuad (int index, PdfQuad *quad)=0
- bool AddQuad (PdfQuad *quad)=0
  Adds a new quad to the text markup annot.
- bool RemoveQuad (int index)=0
  Removes a quad with the specified index.

Detailed Description

PdfTextMarkupAnnot class.
Text markup annotations appear as highlights, underlines, strikeouts, or jagged ("squiggly") underlines in the text of a document.

Member Function Documentation

bool PdfTextMarkupAnnot::AddQuad (PdfQuad * *quad*)

Adds a new quad to the text markup annot.

Parameters:

quad	Pointer to PdfQuad to add.

Returns:

true if quad was added sucessfully, false otherwise.

See also:

PdfTextMarkupAnnot::GetNumQuads

int PdfTextMarkupAnnot::GetNumQuads ()

Gets the number of quads for the annotation.

Returns:

Number of quads.

See also:

PdfTextMarkupAnnot::GetQuad

void PdfTextMarkupAnnot::GetQuad (int *index*, PdfQuad * *quad*)

Gets the requested quad. The coordinates of the quadrilaterals are in default user space that comprise the region in which the annotation should be activated.

Parameters:

index	Index of an annotation quad to retrieve.
quad	(filled by the method) Pointer to PdfQuad structure to fill.

See also:

PdfTextMarkupAnnot::GetNumQuads

bool PdfTextMarkupAnnot::RemoveQuad (int *index*)

Removes a quad with the specified index.

Parameters:

index	The index of the quad to remove.

Returns:

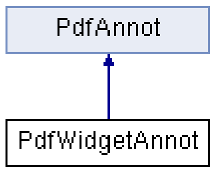
true if quad was removed, false otherwise.

See also:

PdfTextMarkupAnnot::GetNumQuads

PdfWidgetAnnot Struct Reference

PdfWidgetAnnot class.
Inheritance diagram for PdfWidgetAnnot:



Public Member Functions

- int GetCaption (wchar_t *buffer, int len)=0
  Gets an annotation's caption.
- int GetFontName (wchar_t *buffer, int len)=0
  Gets an annotation's font name used for the annotation's appearance.
- PdfAction * GetAction ()=0
  Gets an annotation's action object.
- PdfAction * GetAAction (PdfActionEventType event)=0
  Gets an annotation's additional action object.
- PdfFormField * GetFormField ()=0
  Gets a PdfFormField object related to the annotation. Valid only for Widget annotation.

Detailed Description

PdfWidgetAnnot class.
Interactive forms use widget annotations to represent the appearance of fields and to manage user interactions.

Member Function Documentation

PdfAction* PdfWidgetAnnot::GetAAction (PdfActionEventType *event*)

Gets an annotation's additional action object.

Parameters:

event	The eventwhich additional action to get.

Returns:

The annotation's additional action object or nullptr if annotation does not have an action for specified event type.

See also:

PdfWidgetAnnot::GetAction

PdfAction* PdfWidgetAnnot::GetAction ()

Gets an annotation's action object.

Returns:

The annotation's action object or nullptr if annotation does not have an action.

See also:

PdfWidgetAnnot::GetAAction

int PdfWidgetAnnot::GetCaption (wchar_t * *buffer*, int *len*)

Gets an annotation's caption.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

int PdfWidgetAnnot::GetFontName (wchar_t * *buffer*, int *len*)

Gets an annotation's font name used for the annotation's appearance.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of string.
len	Length of a buffer to be filled in.

Returns:

Number of characters written into buffer of required length.

PdfFormField* PdfWidgetAnnot::GetFormField ()

Gets a PdfFormField object related to the annotation. Valid only for Widget annotation.

Returns:

The PdfFormField object or nullptr if no such form field field object exists.

PsRegex Struct Reference

PsRegex class.
Inherited by CPsRegex.

Public Member Functions

- void Destroy ()=0
  Destroys PsRegex resources.
- bool SetPattern (const wchar_t *pattern)=0
  Sets a regular expression to search for.
- bool Search (const wchar_t *text, int position)=0
  Searches for a match in a string. Use positions parameter to find more patterns.
- int GetText (wchar_t *buffer, int len)=0
  Gets a buffer containing the matched text if it finds a match, otherwise it returns 0.
- int GetPosition ()=0
- int GetLength ()=0
  Gets a length of the matched text.

Detailed Description

PsRegex class.
A regular expression is an object that describes a pattern of characters. Regular expressions are used to perform pattern-matching functions on text. It helps to recognize a logical structure in a document. NOTE: Use Perl Regular Expression Syntax to create a new pattern.

Member Function Documentation

void PsRegex::Destroy ()

Destroys PsRegex resources.

See also:

PsRegex::CreatePsRegex

int PsRegex::GetLength ()

Gets a length of the matched text.

Returns:

Length of the matched text, otherwise it returns 0.

int PsRegex::GetPosition ()

Gets a position of the matched text from the start position defined in PsRegex::Search method. NOTE: It's not a position from text buffer beginning.

Returns:

Position of the matched text, otherwise it returns -1.

int PsRegex::GetText (wchar_t * *buffer*, int *len*)

Gets a buffer containing the matched text if it finds a match, otherwise it returns 0.

Parameters:

buffer	(filled by method) If the buffer is null function returns required length of the buffer.
len	Length of the buffer to be filled in.

Returns:

Number of characters written into the buffer of required length.

bool PsRegex::Search (const wchar_t * *text*, int *position*)

Searches for a match in a string. Use positions parameter to find more patterns.

```
void PsRegexSearch(std::wstring doc_path) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PdfDoc* doc = nullptr;
  doc = pdfix->OpenDoc(doc_path.c_str(), L"");
  if (!doc)
```

```
    throw std::runtime_error(pdfix->GetError());

  // load first page
  PdfPage* page = doc->AcquirePage(0);
  if (!page)
    throw std::runtime_error(pdfix->GetError());
  PdfPageMapParams params;
  PdePageMap* page_map = page->AcquirePageMap(&params, nullptr, nullptr);
  if (!page_map)
    throw std::runtime_error(pdfix->GetError());

  // initialize regex object to match credit card numbers
  // credit card numbers generally come as a string of 16 - digits,
  // separated into groups of 4 - digits, and separated by either a space or a hyphen
  PsRegex* regex = pdfix->CreateRegex();
  std::wstring card_pattern = L"(\\d{4}[- ]){3}\\d{4}";
  regex->SetPattern(card_pattern.c_str());

  // iterate through all PdeText and search for pattern
  int count = page_map->GetNumElements();
  for (int i = 0; i < count; i++) {
    PdeElement* elem = page_map->GetElement(i);
    if (elem->GetType() == kPdeText) {
      PdeText* text_elem = static_cast<PdeText*>(elem);
      std::wstring text;
      text.resize(text_elem->GetText(nullptr, 0));
      text_elem->GetText((wchar_t*)text.c_str(), text.size());
      int start_pos = 0;
      while (start_pos < (int)text.length()) {
        if (regex->Search(text.c_str(), start_pos)) {
          int pos = regex->GetPosition();
          int len = regex->GetLength();
          std::wstring match = text.substr(start_pos + pos, len);
          std::wcout << match << std::endl;
          start_pos += pos + 1;
        }
        else
          start_pos = text.length(); // finish
      }
    }
  }

  // release resources
  regex->Destroy();
  doc->ReleasePage(0);
  doc->Close();
}
```

Parameters:

text	The string to be searched.
position	A position in the text where to start search.

Returns:

This method returns true if it finds a match, otherwise it returns false.

bool PsRegex::SetPattern (const wchar_t * *pattern*)

Sets a regular expression to search for.

```
// Uses ECMAScript regular expressions pattern syntax.
void PsRegexSetPattern(std::wstring text) {
  Pdfix* pdfix = GetPdfix();
  if (!pdfix)
    throw std::runtime_error("Pdfix was not initialized!");
  PsRegex* regex = pdfix->CreateRegex();
  std::wstring pattern[10];
  // All major credit cards regex
  pattern[0] = L"^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6011[0-9]{12}|622((12[6-9]|"
    "1[3-9][0-9])|([2-8][0-9][0-9])|(9(([0-1][0-9])|(2[0-5]))))[0-9]{10}|64[4-9][0-9]{13}|"
    "65[0-9]{14}|3(?:0[0-5]|[68][0-9])[0-9]{11}|3[47][0-9]{13})*$";
  // American Express Credit Card
  pattern[1] = L"^(3[47][0-9]{13})*$";
  // MasterCard Credit Card
  pattern[2] = L"^(5[1-5][0-9]{14})*$";
  // Visa Credit Card
  pattern[3] = L"^(4[0-9]{12}(?:[0-9]{3})?)*$";
  // Phone Numbers(North American)
  pattern[4] = L"^((([0-9]{1})*[- .(]*([0-9]{3})[- .)]*[0-9]{3}[- .]*[0-9]{4})+)*$";
  // Social Security Numbers
  pattern[5] = L"^([0-9]{3}[-]*[0-9]{2}[-]*[0-9]{4})*$";
  // UK Postal Codes
  pattern[6] = L"^([A-Z]{1,2}[0-9][A-Z0-9]? [0-9][ABD-HJLNP-UW-Z]{2})*$";
  // URLs
  pattern[7] = L"^((http|https|ftp)://)?([[a-zA-Z0-9]\\-\\.])+(\\.)([[a-zA-Z0-9]]){2,4}"
    "([[a-zA-Z0-9]/+=%&_\\.~?\\-]*)$";
  // Emails
  pattern[8] = L"^[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}$";
  // Passwords
  pattern[9] = L"(?=^.{6,}$)((?=.*[A-Za-z0-9])(?=.*[A-Z])(?=.*[a-z]))^.*";

  for (int i = 0; i < 10; i++) {
    regex->SetPattern(pattern[i].c_str());
    // fint first occurence of pattern
    if (regex->Search(text.c_str(), 0)) {
      int pos = regex->GetPosition();
      int len = regex->GetLength();
      std::wstring match_text;
      match_text.resize(regex->GetText(nullptr, 0));
      regex->GetText((wchar_t*)match_text.c_str(), match_text.size());
      std::wcout << match_text << std::endl;
    }
  }
}
```

Parameters:

pattern	The Regular expression.

Returns:

true if pattern was set, false otherwise.

See also:

CPsRegex::AddPatternType, CPsRegex::Search

PsStream Struct Reference

PsStream class.
Inherited by CPsStream.

Public Member Functions

- void Destroy ()=0

Destroys PsStream resources.

- int Write (unsigned char *buffer, int size)=0
  Writes data from a memory buffer into a stream, beginning at the current seek position.
- int GetEof ()=0
  Gets the current size of a stream.
- int Read (unsigned char *buffer, int size)=0
  Reads data from PsStream into memory.
- int GetPos ()=0
- bool SetPos (int pos)=0

Detailed Description

PsStream class.
A PsStream is a data stream that may be a buffer in memory, a file, or an arbitrary user-written procedure. You typically would use an PsStream to import/export data to/from/ a PDF file. PsStream methods allow you to open and close streams, and to read and write data.

Member Function Documentation

void PsStream::Destroy ()

Destroys PsStream resources.

See also:

PsRegex::CreatePsStream

int PsStream::GetEof ()

Gets the current size of a stream.

Returns:

The size of the stream.

See also:

PsStream::Read

int PsStream::GetPos ()

Gets the current seek position in a file. This is the position at which the next read or write will begin.

Returns:

The current seek position.

int PsStream::Read (unsigned char * *buffer*, int *size*)

Reads data from PsStream into memory.

Parameters:

buffer	(Filled by the method) A buffer into which data is written. The buffer must be able to hold at least 'size' bytes.
size	The number of bytes to read.

Returns:

The number of bytes actually read from the stream.

bool PsStream::SetPos (int *pos*)

Seeks to the specified position in a stream. This is the position at which the next read or write will begin.

Parameters:

pos	The position to seek.

int PsStream::Write (unsigned char * *buffer*, int *size*)

Writes data from a memory buffer into a stream, beginning at the current seek position.

Parameters:

buffer	A buffer holding the data that is to be written. The buffer must be able to hold at least count bytes.
size	The number of bytes to write.

Returns:

The number of bytes actually written to the stream.

See also:

PsStream::Destroy