



Smart IDReader Library Reference

version 3.2.1

Generated by Doxygen 1.8.11

Contents

1 Overview	1
2 Class Documentation	1
2.1 se::smartid::ForensicField Class Reference	1
2.1.1 Detailed Description	2
2.1.2 Constructor & Destructor Documentation	2
2.1.3 Member Function Documentation	3
2.2 se::smartid::Image Class Reference	3
2.2.1 Detailed Description	5
2.2.2 Constructor & Destructor Documentation	5
2.2.3 Member Function Documentation	6
2.3 se::smartid::ImageField Class Reference	10
2.3.1 Detailed Description	11
2.3.2 Constructor & Destructor Documentation	11
2.4 se::smartid::IntegratedFieldState Class Reference	12
2.4.1 Detailed Description	12
2.5 se::smartid::MatchResult Class Reference	12
2.5.1 Detailed Description	13
2.5.2 Constructor & Destructor Documentation	13
2.6 se::smartid::OcrChar Class Reference	13
2.6.1 Detailed Description	14
2.6.2 Constructor & Destructor Documentation	14
2.6.3 Member Function Documentation	14
2.7 se::smartid::OcrCharVariant Class Reference	15
2.7.1 Detailed Description	15
2.7.2 Constructor & Destructor Documentation	15
2.8 se::smartid::OcrString Class Reference	16
2.8.1 Detailed Description	17
2.9 se::smartid::Point Class Reference	17

2.9.1	Detailed Description	17
2.9.2	Constructor & Destructor Documentation	17
2.10	se::smartid::ProcessingFeedback Class Reference	18
2.10.1	Detailed Description	18
2.10.2	Member Function Documentation	18
2.11	se::smartid::Quadrangle Class Reference	18
2.11.1	Detailed Description	19
2.11.2	Constructor & Destructor Documentation	19
2.11.3	Member Function Documentation	19
2.12	se::smartid::RecognitionEngine Class Reference	21
2.12.1	Detailed Description	22
2.12.2	Constructor & Destructor Documentation	22
2.12.3	Member Function Documentation	22
2.13	se::smartid::RecognitionResult Class Reference	23
2.13.1	Detailed Description	25
2.13.2	Member Function Documentation	26
2.14	se::smartid::RecognitionSession Class Reference	29
2.14.1	Detailed Description	30
2.14.2	Member Function Documentation	31
2.15	se::smartid::Rectangle Class Reference	36
2.15.1	Detailed Description	37
2.15.2	Constructor & Destructor Documentation	37
2.16	se::smartid::ResultReporterInterface Class Reference	37
2.16.1	Detailed Description	37
2.16.2	Member Function Documentation	38
2.17	se::smartid::SegmentationResult Class Reference	38
2.17.1	Detailed Description	39
2.17.2	Member Function Documentation	39
2.18	se::smartid::SessionSettings Class Reference	40
2.18.1	Detailed Description	42
2.18.2	Member Function Documentation	42
2.19	se::smartid::SessionState Class Reference	47
2.19.1	Detailed Description	48
2.19.2	Member Function Documentation	48
2.20	se::smartid::StringField Class Reference	49
2.20.1	Detailed Description	50
2.20.2	Constructor & Destructor Documentation	50
2.20.3	Member Function Documentation	50

1 Overview	1
3 File Documentation	51
3.1 smartid_common.h File Reference	51
3.1.1 Detailed Description	52
3.2 smartid_common.h	52
3.3 smartid_engine.h File Reference	54
3.3.1 Detailed Description	54
3.4 smartid_engine.h	54
3.5 smartid_result.h File Reference	57
3.5.1 Detailed Description	57
3.6 smartid_result.h	58
Index	65

1 Overview

The Smart ID Reader Library allows to recognize various ID documents on images or video data obtained either from cameras or from scanners.

This file contains a brief description of classes and members of the Library. Sample usage is shown in the `smartid_sample.cpp`.

Feel free to send any questions about the Library on support@smartengines.biz.

2 Class Documentation

2.1 se::smartid::ForensicField Class Reference

Class represents implementation of SmartID forensic field for document validity checks.

Public Member Functions

- [ForensicField](#) ()
Default constructor.
- [ForensicField](#) (const std::string &name, const std::string &value, bool is_accepted, double confidence, const std::map< std::string, std::string > &attributes={}) throw (std::exception)
ForensicField main ctor.
- [~ForensicField](#) ()
Destructor.
- const std::string & [GetName](#) () const
Getter for string field name.
- const std::string & [GetValue](#) () const
Getter for string field value (string representation)
- bool [IsAccepted](#) () const
Whether the system is confidence in field recognition result.
- double [GetConfidence](#) () const
The system's confidence level in field recognition result (in range [0..1])
- std::vector< std::string > [GetAttributeNames](#) () const
Returns a vector of attribute names.
- const std::map< std::string, std::string > & [GetAttributes](#) () const
Getter for attributes map.
- bool [HasAttribute](#) (const std::string &attribute_name) const
Check if attribute with given name is present.
- const std::string & [GetAttribute](#) (const std::string &attribute_name) const throw (std::exception)
Get attribute value by its name.

Private Attributes

- std::string [name_](#)
Field name.
- std::string [value_](#)
Fields' value.
- bool [is_accepted_](#)
Specifies whether the system is confident in field recognition result.
- double [confidence_](#)
Specifies the system's confidence level in field recognition result.
- std::map< std::string, std::string > [attributes_](#)
Field attributes.

2.1.1 Detailed Description

Class represents implementation of SmartID forensic field for document validity checks.

Definition at line 351 of file [smartid_result.h](#).

2.1.2 Constructor & Destructor Documentation

- 2.1.2.1 `se::smartid::ForensicField::ForensicField (const std::string & name, const std::string & value, bool is_accepted, double confidence, const std::map< std::string, std::string > & attributes = { }) throw std::exception)`

[ForensicField](#) main ctor.

Parameters

<i>name</i>	- name of the field
<i>value</i>	- string-representation of the field value
<i>is_accepted</i>	- whether the system is confident in the field's value
<i>confidence</i>	- system's confidence level in fields' value in range [0..1\
<i>attributes</i>	- additional field information

Exceptions

<i>std::invalid_argument</i>	if confidence value is not in range [0..1]
------------------------------	--

2.1.3 Member Function Documentation

2.1.3.1 `const std::string& se::smartid::ForensicField::GetAttribute (const std::string & attribute_name) const throw std::exception)`

Get attribute value by its name.

Parameters

<i>attribute_name</i>	key attribute name
-----------------------	--------------------

Returns

attribute value by its name

2.1.3.2 `bool se::smartid::ForensicField::HasAttribute (const std::string & attribute_name) const`

Check if attribute with given name is present.

Parameters

<i>attribute_name</i>	attribute name to check presence of
-----------------------	-------------------------------------

Returns

true if attribute with given name is present

2.2 se::smartid::Image Class Reference

Class for representing a bitmap image.

Public Member Functions

- [Image](#) ()
Default ctor, creates null image with no memory owning.
- [Image](#) (const std::string &image_filename) throw (std::exception)
smartid::Image ctor from image file
- [Image](#) (unsigned char *data, size_t data_length, int width, int height, int stride, int channels) throw (std::exception)
smartid::Image ctor from raw buffer
- [Image](#) (unsigned char *yuv_data, size_t yuv_data_length, int width, int height) throw (std::exception)
smartid::Image ctor from YUV buffer
- [Image](#) (const [Image](#) ©)
smartid::Image copy ctor
- [Image](#) & operator= (const [Image](#) &other)
smartid::Image assignment operator
- ~[Image](#) ()
Image dtor.
- void [Save](#) (const std::string &image_filename) const throw (std::exception)
Saves an image to file.
- int [GetRequiredBufferLength](#) () const
Returns required buffer size for copying image data, O(1)
- int [CopyToBuffer](#) (char *out_buffer, int buffer_length) const throw (std::exception)
Copies the image data to specified buffer.
- double [EstimateFocusScore](#) (double quantile=0.95) const throw (std::exception)
EstimateFocusScore.
- int [GetRequiredBase64BufferLength](#) () const throw (std::exception)
Returns required buffer size for Base64 JPEG representation of an image. WARNING: will perform one extra JPEG coding of an image.
- int [CopyBase64ToBuffer](#) (char *out_buffer, int buffer_length) const throw (std::exception)
Copy JPEG representation of the image to buffer (in base64 coding). The buffer must be large enough.
- std::string [GetBase64String](#) () const throw (std::exception)
Copies JPEG representation of the image and returns it as a simple string (in base64 coding).
- void [Clear](#) ()
Clears the internal image structure, deallocates memory if owns it.
- int [GetWidth](#) () const
Getter for image width.
- int [GetHeight](#) () const
Getter for image height.
- int [GetStride](#) () const
Getter for image stride.
- int [GetChannels](#) () const
Getter for number of image channels.
- bool [IsMemoryOwner](#) () const
Returns whether this instance owns (and will release) image data.
- void [ForceMemoryOwner](#) () throw (std::exception)
Forces memory ownership - allocates new image data and sets memown to true if memown was false, otherwise does nothing.
- void [Resize](#) (int new_width, int new_height) throw (std::exception)
Scale (resize) this image to new width and height, force memory ownership.
- void [Crop](#) (const [Quadrangle](#) &quad) throw (std::exception)
Projectively crop a region of image, forces memory ownership.

- void [Crop](#) (const [Quadrangle](#) &quad, int [width](#), int [height](#)) throw (std::exception)
Projectively crop a region of image, to a new size, forces memory ownership.
- void [MaskImageRegionRectangle](#) ([Rectangle](#) rect, int pixel_expand=0) throw (std::exception)
Masks image region specified by rectangle, forces memory ownership.
- void [MaskImageRegionQuadrangle](#) ([Quadrangle](#) quad, int pixel_expand=0) throw (std::exception)
Masks image region specified by quadrangle, forces memory ownership.
- void [FlipVertical](#) () throw (std::exception)
Flips an image around vertical axis.
- void [FlipHorizontal](#) () throw (std::exception)
Flips an image around horizontal axis.

Public Attributes

- char * [data](#)
Pointer to the first pixel of the first row.
- int [width](#)
Width of the image in pixels.
- int [height](#)
Height of the image in pixels.
- int [stride](#)
Difference in bytes between addresses of adjacent rows.
- int [channels](#)
Number of image channels.
- bool [memown](#)
Whether the image owns the memory itself.

2.2.1 Detailed Description

Class for representing a bitmap image.

Definition at line 166 of file [smartid_common.h](#).

2.2.2 Constructor & Destructor Documentation

2.2.2.1 se::smartid::Image::Image (const std::string & *image_filename*) throw std::exception)

[smartid::Image](#) ctor from image file

Parameters

<i>image_filename</i>	- path to an image. Supported formats: png, jpg, tif
-----------------------	--

Exceptions

<i>std::runtime_error</i>	if image loading failed
---------------------------	-------------------------

2.2.2.2 `se::smartid::Image::Image (unsigned char * data, size_t data_length, int width, int height, int stride, int channels)`
`throw std::exception)`

[smartid::Image](#) ctor from raw buffer

Parameters

<i>data</i>	- pointer to a buffer start
<i>data_length</i>	- length of the buffer
<i>width</i>	- width of the image
<i>height</i>	- height of the image
<i>stride</i>	- address difference between two vertically adjacent pixels in bytes
<i>channels</i>	- number of image channels (1-grayscale, 3-RGB, 4-BGRA)

resulting image is a memory-owning copy

Exceptions

<code>std::runtime_error</code>	if image creating failed
---------------------------------	--------------------------

2.2.2.3 `se::smartid::Image::Image (unsigned char * yuv_data, size_t yuv_data_length, int width, int height)`
`throw std::exception)`

[smartid::Image](#) ctor from YUV buffer

Parameters

<i>yuv_data</i>	- Pointer to the data buffer start
<i>yuv_data_length</i>	- Total length of image data buffer
<i>width</i>	- Image width
<i>height</i>	- Image height

Exceptions

<code>std::exception</code>	if image creating failed
-----------------------------	--------------------------

2.2.2.4 `se::smartid::Image::Image (const Image & copy)`

[smartid::Image](#) copy ctor

Parameters

<i>copy</i>	- an image to copy from. If 'copy' doesn't own memory then only the reference is copied. If 'copy' owns image memory then new image will be allocated with the same data as 'copy'.
-------------	---

2.2.3 Member Function Documentation

2.2.3.1 int se::smartid::Image::CopyBase64ToBuffer (char * *out_buffer*, int *buffer_length*) const throw std::exception)

Copy JPEG representation of the image to buffer (in base64 coding). The buffer must be large enough.

Parameters

<i>out_buffer</i>	Destination buffer, must be preallocated
<i>buffer_length</i>	Size of buffer <i>out_buffer</i>

Returns

Number of bytes copied

Exceptions

<i>std::invalid_argument</i>	if buffer size (<i>buffer_length</i>) is not enough to store the image, or if <i>out_buffer</i> is NULL. <i>std::runtime_error</i> if unexpected error happened in the copying process
------------------------------	---

2.2.3.2 int se::smartid::Image::CopyToBuffer (char * *out_buffer*, int *buffer_length*) const throw std::exception)

Copies the image data to specified buffer.

Parameters

<i>out_buffer</i>	Destination buffer, must be preallocated
<i>buffer_length</i>	Size of buffer <i>out_buffer</i>

Returns

Number of bytes copied

Exceptions

<i>std::invalid_argument</i>	if buffer size (<i>buffer_length</i>) is not enough to store the image, or if <i>out_buffer</i> is NULL. <i>std::runtime_error</i> if unexpected error happened in the copying process
------------------------------	---

2.2.3.3 void se::smartid::Image::Crop (const Quadrangle & *quad*) throw std::exception)

Projectively crop a region of image, forces memory ownership.

Parameters

<i>quad</i>	- a region of image to crop
-------------	-----------------------------

2.2.3.4 void se::smartid::Image::Crop (const Quadrangle & *quad*, int *width*, int *height*) throw std::exception)

Projectively crop a region of image, to a new size, forces memory ownership.

Parameters

<i>quad</i>	- a region of image to crop
<i>width</i>	- new width of the cropped image
<i>height</i>	- new height of the cropped image

2.2.3.5 `double se::smartid::Image::EstimateFocusScore (double quantile = 0 . 95) const throw std::exception)`

EstimateFocusScore.

Returns

Estimated focus score of [Image](#) in range

2.2.3.6 `std::string se::smartid::Image::GetBase64String () const throw std::exception)`

Copies JPEG representation of the image and returns it as a simple string (in base64 coding).

Returns

std::string with serialized image

Exceptions

<i>std::runtime_error</i>	if unexpected error occurred
---------------------------	------------------------------

2.2.3.7 `int se::smartid::Image::GetChannels () const`

Getter for number of image channels.

Returns

[Image](#) number of channels

2.2.3.8 `int se::smartid::Image::GetHeight () const`

Getter for image height.

Returns

[Image](#) height in pixels

2.2.3.9 `int se::smartid::Image::GetRequiredBase64BufferLength () const throw std::exception)`

Returns required buffer size for Base64 JPEG representation of an image. WARNING: will perform one extra JPEG coding of an image.

Returns

Buffer size in bytes

Exceptions

<code>std::runtime_error</code>	if failed to calculate the necessary buffer size
---------------------------------	--

2.2.3.10 `int se::smartid::Image::GetRequiredBufferLength () const`

Returns required buffer size for copying image data, O(1)

Returns

Buffer size in bytes

2.2.3.11 `int se::smartid::Image::GetStride () const`

Getter for image stride.

Returns

[Image](#) row size in bytes

2.2.3.12 `int se::smartid::Image::GetWidth () const`

Getter for image width.

Returns

[Image](#) width in pixels

2.2.3.13 `bool se::smartid::Image::IsMemoryOwner () const`

Returns whether this instance owns (and will release) image data.

Returns

memown variable value

2.2.3.14 `void se::smartid::Image::MaskImageRegionQuadrangle (Quadrangle quad, int pixel_expand = 0) throw std::exception)`

Masks image region specified by quadrangle, forces memory ownership.

Parameters

<i>quad</i>	quadrangle to mask over
<i>pixel_expand</i>	expand offset in pixels for each point (0 by default)

2.2.3.15 `void se::smartid::Image::MaskImageRegionRectangle (Rectangle rect, int pixel_expand = 0) throw std::exception)`

Masks image region specified by rectangle, forces memory ownership.

Parameters

<i>rect</i>	bounding rectangle to mask over
<i>pixel_expand</i>	expand offset in pixels for each point (0 by default)

2.2.3.16 `Image& se::smartid::Image::operator= (const Image & other)`

[smartid::Image](#) assignment operator

Parameters

<i>other</i>	- an image to assign. If 'other' doesn't own memory then only the reference is assigned. If 'other' owns image memory then new image will be allocated with the same data as 'other'.
--------------	---

2.2.3.17 `void se::smartid::Image::Resize (int new_width, int new_height) throw std::exception)`

Scale (resize) this image to new width and height, force memory ownership.

Parameters

<i>new_width</i>	New image width
<i>new_height</i>	New image height

2.2.3.18 `void se::smartid::Image::Save (const std::string & image_filename) const throw std::exception)`

Saves an image to file.

Parameters

<i>image_filename</i>	- path to an image. Supported formats: png, jpg, tif, format is deduced from the filename extension
-----------------------	---

Exceptions

<code>std::runtime_error</code>	if image saving failed
---------------------------------	------------------------

2.3 se::smartid::ImageField Class Reference

Class represents implementation of SmartIDField for list of images.

Public Member Functions

- [ImageField](#) ()

ImageField Default ctor.

- [ImageField](#) (const std::string &name, const [Image](#) &value, bool is_accepted, double confidence) throw (std::exception)

ImageField main ctor.

- [~ImageField](#) ()

Default dtor.

- const std::string & [GetName](#) () const

Getter for image field name.

- const [Image](#) & [GetValue](#) () const

Getter for image field result.

- bool [IsAccepted](#) () const

Whether the system is confidence in field result.

- double [GetConfidence](#) () const

The system's confidence level in field result (in range [0..1])

Private Attributes

- std::string [name_](#)

Image field name.

- [Image](#) [value_](#)

Image field value (internal image storage)

- bool [is_accepted_](#)

Specifies whether the system is confident in result.

- double [confidence_](#)

Specifies the system's confidence level in result.

2.3.1 Detailed Description

Class represents implementation of SmartIDField for list of images.

Definition at line 256 of file [smartid_result.h](#).

2.3.2 Constructor & Destructor Documentation

- 2.3.2.1 `se::smartid::ImageField::ImageField (const std::string & name, const Image & value, bool is_accepted, double confidence) throw std::exception`

[ImageField](#) main ctor.

Parameters

<i>name</i>	- name of the field
<i>value</i>	- image (the field result)
<i>is_accepted</i>	- whether the system is confident in the field's value
<i>confidence</i>	- system's confidence level in fields' value in range [0..1]

Exceptions

<code>std::invalid_argument</code>	if confidence value is not in range [0..1] or if failed to decode utf8-string 'value'
------------------------------------	---

2.4 se::smartid::IntegratedFieldState Class Reference

[IntegratedFieldState](#) class - integrated field terminality state.

Public Member Functions

- [IntegratedFieldState](#) (bool is_terminal=false)
Default ctor.
- bool [IsTerminal](#) () const
Whether the systems regards that result for the field as 'final'.
- void [SetIsTerminal](#) (bool is_terminal)
Setter for IsTerminal flag.

Private Attributes

- bool [is_terminal_](#)

2.4.1 Detailed Description

[IntegratedFieldState](#) class - integrated field terminality state.

Definition at line 815 of file [smartid_result.h](#).

2.5 se::smartid::MatchResult Class Reference

Class represents SmartID match result.

Public Member Functions

- [MatchResult](#) ()
Default ctor.
- [MatchResult](#) (const std::string &tpl_type, const [Quadrangle](#) &quadrangle, bool accepted=false, double confidence=0.0, int standard_width=0, int standard_height=0)
MatchResult main ctor.
- [~MatchResult](#) ()
Destructor.
- const std::string & [GetTemplateType](#) () const
Getter for document type string.
- const [Quadrangle](#) & [GetQuadrangle](#) () const
Getter for document quadrangle.
- int [GetStandardWidth](#) () const
Getter for standard template width in pixels.
- int [GetStandardHeight](#) () const
Getter for standard template height in pixels.
- bool [GetAccepted](#) () const
Getter for acceptance field.
- double [GetConfidence](#) () const
Getter for confidence field.

Private Attributes

- `std::string` [template_type_](#)
Template type for this match result.
- [Quadrangle](#) [quadrangle_](#)
Quadrangle for this template.
- `int` [standard_width_](#)
Standard width for this template type.
- `int` [standard_height_](#)
Standard height for this template type.
- `bool` [accepted_](#)
Whether this result is ready to be visualized.
- `double` [confidence_](#)
System's confidence level in match result.

2.5.1 Detailed Description

Class represents SmartID match result.

Definition at line 300 of file [smartid_result.h](#).

2.5.2 Constructor & Destructor Documentation

2.5.2.1 `se::smartid::MatchResult::MatchResult (const std::string & tpl_type, const Quadrangle & quadrangle, bool accepted = false, double confidence = 0.0, int standard_width = 0, int standard_height = 0)`

[MatchResult](#) main ctor.

Parameters

<i>tpl_type</i>	- template type for this match result
<i>quadrangle</i>	- quadrangle of a template on image
<i>accepted</i>	- acceptance for visualization

2.6 se::smartid::OcrChar Class Reference

Contains all OCR information for a given character.

Public Member Functions

- [OcrChar](#) ()
Default ctor.
- [OcrChar](#) (const std::vector< [OcrCharVariant](#) > &ocr_char_variants, bool is_highlighted, bool is_corrected, const [Rectangle](#) &ocr_char_rect={})
Main ctor.
- [~OcrChar](#) ()

OcrChar dtor.

- `const std::vector< OcrCharVariant > & GetOcrCharVariants () const`
Vector with possible recognition results for a given character.
- `bool IsHighlighted () const`
Whether this character is 'highlighted' (not confident) by the system.
- `bool IsCorrected () const`
Whether this character was changed by context correction (postprocessing)
- `uint16_t GetUtf16Character () const throw (std::exception)`
Returns the most confident character as 16-bit utf16 character.
- `std::string GetUtf8Character () const throw (std::exception)`
Returns the most confident character as utf8 representation of 16-bit character.
- `const Rectangle & GetRectangle () const`
Returns the rect position of char on field's image.

Private Attributes

- `std::vector< OcrCharVariant > ocr_char_variants_`
- `bool is_highlighted_`
- `bool is_corrected_`
- `Rectangle rect_`

2.6.1 Detailed Description

Contains all OCR information for a given character.

Definition at line 77 of file [smartid_result.h](#).

2.6.2 Constructor & Destructor Documentation

2.6.2.1 `se::smartid::OcrChar::OcrChar (const std::vector< OcrCharVariant > & ocr_char_variants, bool is_highlighted, bool is_corrected, const Rectangle & ocr_char_rect = { })`

Main ctor.

Parameters

ocr_char_variants	- vector of char variants
is_highlighted	- whether this OcrChar is highlighted as unconfident
is_corrected	- whether this OcrChar was corrected by post-processing

2.6.3 Member Function Documentation

2.6.3.1 `uint16_t se::smartid::OcrChar::GetUtf16Character () const throw std::exception)`

Returns the most confident character as 16-bit utf16 character.

Exceptions

<code>std::out_of_range</code>	if variants are empty
--------------------------------	-----------------------

2.6.3.2 std::string se::smartid::OcrChar::GetUtf8Character () const throw std::exception)

Returns the most confident character as utf8 representation of 16-bit character.

Exceptions

<code>std::out_of_range</code>	if variants are empty
--------------------------------	-----------------------

2.7 se::smartid::OcrCharVariant Class Reference

Possible character recognition result.

Public Member Functions

- [OcrCharVariant](#) ()
Default ctor.
- [~OcrCharVariant](#) ()
OcrCharVariant dtor.
- [OcrCharVariant](#) (uint16_t utf16_char, double confidence) throw (std::exception)
Ctor from utf16 character and confidence.
- [OcrCharVariant](#) (const std::string &utf8_char, double confidence) throw (std::exception)
Ctor from utf8 character and confidence.
- uint16_t [GetUtf16Character](#) () const
Getter for character in Utf16 form.
- std::string [GetUtf8Character](#) () const
Getter for character in Utf8 form.
- double [GetConfidence](#) () const
Variant confidence (pseudoprobability), in range [0..1].

Private Attributes

- uint16_t **character_**
- double **confidence_**

2.7.1 Detailed Description

Possible character recognition result.

Definition at line 31 of file [smartid_result.h](#).

2.7.2 Constructor & Destructor Documentation

2.7.2.1 se::smartid::OcrCharVariant::OcrCharVariant (uint16_t utf16_char, double confidence) throw std::exception)

Ctor from utf16 character and confidence.

Parameters

<i>utf16_char</i>	- Utf16-character of a symbol
<i>confidence</i>	- double confidence in range [0..1]

Exceptions

<i>std::invalid_argument</i>	if confidence is not in range [0..1]
------------------------------	--------------------------------------

2.7.2.2 `se::smartid::OcrCharVariant::OcrCharVariant (const std::string & utf8_char, double confidence) throw std::exception)`

Ctor from utf8 character and confidence.

Parameters

<i>utf8_char</i>	- utf8-representation of a 2-byte symbol in std::string form
<i>confidence</i>	- double confidence in range [0..1]

Exceptions

<i>std::invalid_argument</i>	if confidence is not in range [0..1] or if utf8_char is not a correct utf8 representation of 2-byte symbol
------------------------------	--

2.8 se::smartid::OcrString Class Reference

Contains additional OCR information for the whole string.

Public Member Functions

- [OcrString \(\)](#)
Default ctor.
- [OcrString \(const std::vector< \[OcrChar\]\(#\) > &ocr_chars\)](#)
Ctor from vector of OcrChars.
- [OcrString \(const std::string &utf8_string\)](#)
OcrString ctor from plain utf8 string.
- [~OcrString \(\)](#)
OcrString dtor.
- `const std::vector< OcrChar > & GetOcrChars () const`
Vector with OCR information for each character.
- `std::string GetUtf8String () const`
Returns the most-confident string representation.
- `std::vector< uint16_t > GetUtf16String () const`
Returns the most-confident string representation.

Private Attributes

- `std::vector< OcrChar > ocr_chars_`
Vector with OCR information for each character.

2.8.1 Detailed Description

Contains additional OCR information for the whole string.

Definition at line 137 of file [smartid_result.h](#).

2.9 se::smartid::Point Class Reference

Class for representing a point on an image.

Public Member Functions

- [Point](#) ()
Default Constructor ($x = y = 0$)
- [~Point](#) ()
Destructor.
- [Point](#) (double [x](#), double [y](#))
Constructor.

Public Attributes

- double [x](#)
x-coordinate in pixels (top-left corner is origin)
- double [y](#)
y-coordinate in pixels (top-left corner is origin)

2.9.1 Detailed Description

Class for representing a point on an image.

Definition at line 67 of file [smartid_common.h](#).

2.9.2 Constructor & Destructor Documentation

2.9.2.1 se::smartid::Point::Point (double [x](#), double [y](#))

Constructor.

Parameters

x	- x-coordinate of a point in pixels (top-left corner is origin)
y	- y-coordinate of a point in pixels (top-left corner is origin)

2.10 `se::smartid::ProcessingFeedback` Class Reference

Feedback data that is returned by the [ResultReporterInterface](#)'s `FeedbackReceived` method, containing useful user-oriented information such as additional visualization, advisory information etc.

Public Member Functions

- [ProcessingFeedback](#) ()
Default constructor.
- [ProcessingFeedback](#) (const std::map< std::string, [Quadrangle](#) > &quadrangles)
Main constructor.
- [~ProcessingFeedback](#) ()
Destructor.
- const std::map< std::string, [Quadrangle](#) > & [GetQuadrangles](#) () const
Getter for arbitrary quadrangles feedback data.

Private Attributes

- std::map< std::string, [Quadrangle](#) > [quadrangles_](#)
quadrangle data

2.10.1 Detailed Description

Feedback data that is returned by the [ResultReporterInterface](#)'s `FeedbackReceived` method, containing useful user-oriented information such as additional visualization, advisory information etc.

Definition at line 740 of file [smartid_result.h](#).

2.10.2 Member Function Documentation

2.10.2.1 `const std::map<std::string, Quadrangle>& se::smartid::ProcessingFeedback::GetQuadrangles () const`

Getter for arbitrary quadrangles feedback data.

Returns

map with quadrangles feedback data

2.11 `se::smartid::Quadrangle` Class Reference

Class for representing a quadrangle on an image.

Public Member Functions

- [Quadrangle](#) ()
Constructor.
- [~Quadrangle](#) ()
Destructor.
- [Quadrangle](#) ([Point](#) a, [Point](#) b, [Point](#) c, [Point](#) d)
Constructor.
- [Point](#) & [operator\[\]](#) (int index) throw (std::exception)
Returns the quadrangle vertex at the given `index` as a modifiable reference.
- const [Point](#) & [operator\[\]](#) (int index) const throw (std::exception)
Returns the quadrangle vertex at the given `index` as a constant reference.
- const [Point](#) & [GetPoint](#) (int index) const throw (std::exception)
Returns the quadrangle vertex at the given `index` as a constant reference.
- void [SetPoint](#) (int index, const [Point](#) &value) throw (std::exception)
Sets the quadrangle vertex at the given `index` to specified `value`.
- [Rectangle](#) [GetBoundingRectangle](#) () const
Computes and returns bounding rectangle for quadrangle's points.

Private Attributes

- [Point](#) [points](#) [4]
Vector of quadrangle vertices in order: top-left, top-right, bottom-right, bottom-left.

2.11.1 Detailed Description

Class for representing a quadrangle on an image.

Definition at line 93 of file [smartid_common.h](#).

2.11.2 Constructor & Destructor Documentation

2.11.2.1 se::smartid::Quadrangle::Quadrangle ([Point](#) a, [Point](#) b, [Point](#) c, [Point](#) d)

Constructor.

Parameters

<i>a</i>	Top-left vertex of the quadrangle
<i>b</i>	Top-right vertex of the quadrangle
<i>c</i>	Bottom-right vertex of the quadrangle
<i>d</i>	Bottom-left vertex of the quadrangle

2.11.3 Member Function Documentation

2.11.3.1 **Rectangle** `se::smartid::Quadrangle::GetBoundingRectangle () const`

Computes and returns bounding rectangle for quadrangle's points.

Returns

computed bounding rectangle

2.11.3.2 `const Point& se::smartid::Quadrangle::GetPoint (int index) const throw std::exception)`

Returns the quadrangle vertex at the given `index` as a constant reference.

Parameters

<i>index</i>	Index position for quadrangle vertex, from 0 till 3
--------------	---

Exceptions

<code>std::out_of_range</code>	if index is not in range [0 ... 3]
--------------------------------	------------------------------------

2.11.3.3 `Point& se::smartid::Quadrangle::operator[] (int index) throw std::exception)`

Returns the quadrangle vertex at the given `index` as a modifiable reference.

Parameters

<i>index</i>	Index position for quadrangle vertex, from 0 till 3
--------------	---

Exceptions

<code>std::out_of_range</code>	if index is not in range [0 ... 3]
--------------------------------	------------------------------------

2.11.3.4 `const Point& se::smartid::Quadrangle::operator[] (int index) const throw std::exception)`

Returns the quadrangle vertex at the given `index` as a constant reference.

Parameters

<i>index</i>	Index position for quadrangle vertex, from 0 till 3
--------------	---

Exceptions

<code>std::out_of_range</code>	if index is not in range [0 ... 3]
--------------------------------	------------------------------------

2.11.3.5 `void se::smartid::Quadrangle::SetPoint (int index, const Point & value) throw std::exception)`

Sets the quadrangle vertex at the given `index` to specified `value`.

Parameters

<i>index</i>	Index position for quadrangle vertex, from 0 till 3
<i>value</i>	New value for quadrangle vertex

Exceptions

<code>std::out_of_range</code>	if index is not in range [0 ... 3]
--------------------------------	------------------------------------

2.12 se::smartid::RecognitionEngine Class Reference

The [RecognitionEngine](#) class - a factory for RecognitionSessions, holds configured internal engines.

Public Member Functions

- [RecognitionEngine](#) (const std::string &config_path, bool lazy_configuration=true) throw (std::exception)
RecognitionEngine ctor from configuration path.
- [RecognitionEngine](#) (unsigned char *config_data, size_t data_length, bool lazy_configuration=true) throw (std::exception)
RecognitionEngine ctor from configuration buffer. Only for configuration from ZIP archive buffers.
- [~RecognitionEngine](#) ()
Recognition Engine dtor.
- [SessionSettings * CreateSessionSettings](#) () const throw (std::exception)
Factory method for creating 'default' session settings with options loaded from configured bundle and no enabled documents.
- [RecognitionSession * SpawnSession](#) (const [SessionSettings](#) &session_settings, [ResultReporterInterface](#) *result_reporter=0) const throw (std::exception)
Sessions for videostream recognition (one document - multiple frames)

Static Public Member Functions

- static std::string [GetVersion](#) ()
Gets RecognitionEngine library version.

Private Member Functions

- [RecognitionEngine](#) (const [RecognitionEngine](#) ©)
Disabled copy constructor.
- void [operator=](#) (const [RecognitionEngine](#) &other)
Disabled assignment operator.

Private Attributes

- class [RecognitionEngineImpl](#) * [pimpl_](#)
pointer to internal implementation

2.12.1 Detailed Description

The [RecognitionEngine](#) class - a factory for RecognitionSessions, holds configured internal engines.

Definition at line 528 of file [smartid_engine.h](#).

2.12.2 Constructor & Destructor Documentation

2.12.2.1 `se::smartid::RecognitionEngine::RecognitionEngine (const std::string & config_path, bool lazy_configuration = true) throw std::exception)`

[RecognitionEngine](#) ctor from configuration path.

Parameters

<i>config_path</i>	- path to configuration file
<i>lazy_configuration</i>	- whether to use engine's lazy component configuration capabilities

Exceptions

<i>std::exception</i>	if configuration error occurs
-----------------------	-------------------------------

2.12.2.2 `se::smartid::RecognitionEngine::RecognitionEngine (unsigned char * config_data, size_t data_length, bool lazy_configuration = true) throw std::exception)`

[RecognitionEngine](#) ctor from configuration buffer. Only for configuration from ZIP archive buffers.

Parameters

<i>config_data</i>	- pointer to configuration ZIP buffer start
<i>data_length</i>	- size of the configuration ZIP buffer
<i>lazy_configuration</i>	- whether to use engine's lazy component configuration capabilities

Exceptions

<i>std::exception</i>	if configuration error occurs
-----------------------	-------------------------------

2.12.3 Member Function Documentation

2.12.3.1 `SessionSettings* se::smartid::RecognitionEngine::CreateSessionSettings () const throw std::exception)`

Factory method for creating 'default' session settings with options loaded from configured bundle and no enabled documents.

Returns

Allocated session settings, caller is responsible for destruction

Exceptions

<code>std::exception</code>	if settings creation failed
-----------------------------	-----------------------------

2.12.3.2 static std::string se::smartid::RecognitionEngine::GetVersion () [static]

Gets [RecognitionEngine](#) library version.

Returns

std::string version representation

2.12.3.3 RecognitionSession* se::smartid::RecognitionEngine::SpawnSession (const SessionSettings & session_settings, ResultReporterInterface * result_reporter = 0) const throw std::exception)

Sessions for videostream recognition (one document - multiple frames)

Factory method for creating a session for SmartId internal engine

Parameters

<code>session_settings</code>	- runtime session settings
<code>result_reporter</code>	- pointer to optional processing reporter implementation

Returns

pointer to created recognition session. The caller is responsible for session's destruction.

Exceptions

<code>std::exception</code>	if session creation failed
-----------------------------	----------------------------

2.13 se::smartid::RecognitionResult Class Reference

Class represents SmartID recognition result.

Public Member Functions

- [RecognitionResult](#) ()
Default ctor.
- [RecognitionResult](#) (const std::map< std::string, [StringField](#) > &string_fields, const std::map< std::string, [ImageField](#) > &image_fields, const std::map< std::string, [ForensicField](#) > &forensic_fields, const std::map< std::string, [StringField](#) > &raw_string_fields, const std::map< std::string, [ImageField](#) > &raw_image_fields, const std::string &document_type, const std::vector< [MatchResult](#) > &match_results, const std::vector< [SegmentationResult](#) > &segmentation_results, bool is_terminal)
RecognitionResult main ctor.

- [~RecognitionResult \(\)](#)
RecognitionResult dtor.
- `std::vector< std::string > GetStringFieldNames () const`
Returns a vector of unique string field names.
- `bool HasStringField (const std::string &name) const`
Checks if there is a string field with given name.
- `const StringField & GetStringField (const std::string &name) const throw (std::exception)`
Gets string field by name.
- `const std::map< std::string, StringField > & GetStringFields () const`
Getter for string fields map.
- `std::map< std::string, StringField > & GetStringFields ()`
Getter for (mutable) string fields map.
- `void SetStringFields (const std::map< std::string, StringField > &string_fields)`
Setter for string fields map.
- `std::vector< std::string > GetImageFieldNames () const`
Returns a vector of unique image field names.
- `bool HasImageField (const std::string &name) const`
Checks if there is a image field with given name.
- `const ImageField & GetImageField (const std::string &name) const throw (std::exception)`
Gets image field by name.
- `const std::map< std::string, ImageField > & GetImageFields () const`
Getter for image fields map.
- `std::map< std::string, ImageField > & GetImageFields ()`
Getter for (mutable) image fields map.
- `void SetImageFields (const std::map< std::string, ImageField > &image_fields)`
Setter for image fields map.
- `std::vector< std::string > GetForensicFieldNames () const`
Returns a vector of unique forensic field names.
- `bool HasForensicField (const std::string &name) const`
Checks if there is a forensic field with given name.
- `const ForensicField & GetForensicField (const std::string &name) const throw (std::exception)`
Gets forensic field by name.
- `const std::map< std::string, ForensicField > & GetForensicFields () const`
Getter for forensic fields map.
- `std::map< std::string, ForensicField > & GetForensicFields ()`
Getter for (mutable) forensic fields map.
- `void SetForensicFields (const std::map< std::string, ForensicField > &forensic_fields)`
Setter for forensic fields map.
- `std::vector< std::string > GetRawStringFieldNames () const`
Returns a vector of unique raw string field names.
- `bool HasRawStringField (const std::string &name) const`
Checks if there is a raw string field with given name.
- `const StringField & GetRawStringField (const std::string &name) const throw (std::exception)`
Gets raw string field by name.
- `const std::map< std::string, StringField > & GetRawStringFields () const`
Getter for raw string fields map.
- `std::map< std::string, StringField > & GetRawStringFields ()`
Getter for (mutable) raw string fields map.
- `void SetRawStringFields (const std::map< std::string, StringField > &raw_string_fields)`
Setter for raw string fields map.
- `std::vector< std::string > GetRawImageFieldNames () const`

- Returns a vector of unique raw image field names.*
- bool [HasRawImageField](#) (const std::string &name) const
Checks if there is a raw image field with given name.
- const [ImageField](#) & [GetRawImageField](#) (const std::string &name) const throw (std::exception)
Gets raw image field by name.
- const std::map< std::string, [ImageField](#) > & [GetRawImageFields](#) () const
Getter for raw image fields map.
- std::map< std::string, [ImageField](#) > & [GetRawImageFields](#) ()
Getter for (mutable) raw image fields map.
- void [SetRawImageFields](#) (const std::map< std::string, [ImageField](#) > &raw_image_fields)
Setter for raw image fields map.
- const std::string & [GetDocumentType](#) () const
Getter for document type name. Empty string means empty result (no document match happened yet)
- void [SetDocumentType](#) (const std::string &doctype)
Setter for document type name.
- const std::vector< [MatchResult](#) > & [GetMatchResults](#) () const
Getter for match results - contains the most 'fresh' template quadrangles information available.
- void [SetMatchResults](#) (const std::vector< [MatchResult](#) > &match_results)
Setter for match results.
- const std::vector< [SegmentationResult](#) > & [GetSegmentationResults](#) () const
Getter for segmentation results - contains the most 'fresh' raw fields and fields location information available.
- void [SetSegmentationResults](#) (const std::vector< [SegmentationResult](#) > &segmentation_results)
Setter for segmentation results.
- bool [IsTerminal](#) () const
Whether the systems regards that result as 'final'. Could be (optionally) used to stop the recognition session.
- void [SetIsTerminal](#) (bool is_terminal)
Setter for IsTerminal flag.
- const std::string & [GetJpegCompression](#) () const
Getter for source application or device, that performed jpeg compression. Empty string means that document is not jpeg-compressed or has unknown source.
- void [SetJpegCompression](#) (const std::string &jpeg_compression)
Setter for jpeg compression.

Private Attributes

- std::map< std::string, [StringField](#) > **string_fields_**
- std::map< std::string, [ImageField](#) > **image_fields_**
- std::map< std::string, [ForensicField](#) > **forensic_fields_**
- std::map< std::string, [StringField](#) > **raw_string_fields_**
- std::map< std::string, [ImageField](#) > **raw_image_fields_**
- std::string **document_type_**
- std::vector< [MatchResult](#) > **match_results_**
- std::vector< [SegmentationResult](#) > **segmentation_results_**
- bool **is_terminal_**
- std::string **jpeg_compression_**

2.13.1 Detailed Description

Class represents SmartID recognition result.

Definition at line 484 of file [smartid_result.h](#).

2.13.2 Member Function Documentation

2.13.2.1 `const ForensicField& se::smartid::RecognitionResult::GetForensicField (const std::string & name) const throw std::exception)`

Gets forensic field by name.

Parameters

<i>name</i>	- name of a forensic field
-------------	----------------------------

Exceptions

<code>std::invalid_argument</code>	if there is no such field
------------------------------------	---------------------------

2.13.2.2 `const std::map<std::string, ForensicField>& se::smartid::RecognitionResult::GetForensicFields () const`

Getter for forensic fields map.

Returns

constref for forensic fields map

2.13.2.3 `std::map<std::string, ForensicField>& se::smartid::RecognitionResult::GetForensicFields ()`

Getter for (mutable) forensic fields map.

Returns

ref for forensic fields map

2.13.2.4 `const ImageField& se::smartid::RecognitionResult::GetImageField (const std::string & name) const throw std::exception)`

Gets image field by name.

Parameters

<i>name</i>	- name of an image field
-------------	--------------------------

Exceptions

<code>std::invalid_argument</code>	if there is no such field
------------------------------------	---------------------------

2.13.2.5 `const std::map<std::string, ImageField>& se::smartid::RecognitionResult::GetImageFields () const`

Getter for image fields map.

Returns

constref for image fields map

2.13.2.6 `std::map<std::string, ImageField>& se::smartid::RecognitionResult::GetImageFields ()`

Getter for (mutable) image fields map.

Returns

ref for image fields map

2.13.2.7 `const ImageField& se::smartid::RecognitionResult::GetRawImageField (const std::string & name) const throw std::exception)`

Gets raw image field by name.

Parameters

<i>name</i>	- raw name of an image field
-------------	------------------------------

Exceptions

<i>std::invalid_argument</i>	if there is no such field
------------------------------	---------------------------

2.13.2.8 `const std::map<std::string, ImageField>& se::smartid::RecognitionResult::GetRawImageFields () const`

Getter for raw image fields map.

Returns

constref for raw image fields map

2.13.2.9 `std::map<std::string, ImageField>& se::smartid::RecognitionResult::GetRawImageFields ()`

Getter for (mutable) raw image fields map.

Returns

ref for raw image fields map

2.13.2.10 `const StringField& se::smartid::RecognitionResult::GetRawStringField (const std::string & name) const throw std::exception)`

Gets raw string field by name.

Parameters

<i>name</i>	- name of a raw string field
-------------	------------------------------

Exceptions

<code>std::invalid_argument</code>	if there is no such field
------------------------------------	---------------------------

2.13.2.11 `const std::map<std::string, StringField>& se::smartid::RecognitionResult::GetRawStringFields () const`

Getter for raw string fields map.

Returns

constref for raw string fields map

2.13.2.12 `std::map<std::string, StringField>& se::smartid::RecognitionResult::GetRawStringFields ()`

Getter for (mutable) raw string fields map.

Returns

ref for raw string fields map

2.13.2.13 `const StringField& se::smartid::RecognitionResult::GetStringField (const std::string & name) const throw std::exception)`

Gets string field by name.

Parameters

<code>name</code>	- name of a string field
-------------------	--------------------------

Exceptions

<code>std::invalid_argument</code>	if there is no such field
------------------------------------	---------------------------

2.13.2.14 `const std::map<std::string, StringField>& se::smartid::RecognitionResult::GetStringFields () const`

Getter for string fields map.

Returns

constref for string fields map

2.13.2.15 `std::map<std::string, StringField>& se::smartid::RecognitionResult::GetStringFields ()`

Getter for (mutable) string fields map.

Returns

ref for string fields map

2.13.2.16 void se::smartid::RecognitionResult::SetForensicFields (const std::map< std::string, ForensicField > & *forensic_fields*)

Setter for forensic fields map.

Parameters

<i>forensic_fields</i>	- string fields map
------------------------	---------------------

2.13.2.17 void se::smartid::RecognitionResult::SetImageFields (const std::map< std::string, ImageField > & *image_fields*)

Setter for image fields map.

Parameters

<i>image_fields</i>	- image fields map
---------------------	--------------------

2.13.2.18 void se::smartid::RecognitionResult::SetRawImageFields (const std::map< std::string, ImageField > & *raw_image_fields*)

Setter for raw image fields map.

Parameters

<i>raw_image_fields</i>	- raw image fields map
-------------------------	------------------------

2.13.2.19 void se::smartid::RecognitionResult::SetRawStringFields (const std::map< std::string, StringField > & *raw_string_fields*)

Setter for raw string fields map.

Parameters

<i>raw_string_fields</i>	- raw string fields map
--------------------------	-------------------------

2.13.2.20 void se::smartid::RecognitionResult::SetStringFields (const std::map< std::string, StringField > & *string_fields*)

Setter for string fields map.

Parameters

<i>string_fields</i>	- string fields map
----------------------	---------------------

2.14 se::smartid::RecognitionSession Class Reference

[RecognitionSession](#) class - main interface for SmartID document recognition in videostream.

Public Member Functions

- virtual [~RecognitionSession](#) ()
RecognitionSession dtor.
- virtual [RecognitionResult ProcessSnapshot](#) (unsigned char *data, size_t data_length, int width, int height, int stride, int channels, const [Rectangle](#) &roi, ImageOrientation image_orientation=[Landscape](#))=0 throw (std::exception)
Processes the uncompressed RGB image stored in memory line by line.
- virtual [RecognitionResult ProcessSnapshot](#) (unsigned char *data, size_t data_length, int width, int height, int stride, int channels, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Processes the uncompressed RGB image stored in memory line by line. Same as ProcessSnapshot with ROI, but with this method the ROI is full image.
- virtual [RecognitionResult ProcessYUVSnapshot](#) (unsigned char *yuv_data, size_t yuv_data_length, int width, int height, const [Rectangle](#) &roi, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Processes the uncompressed YUV image stored in memory line by line.
- virtual [RecognitionResult ProcessYUVSnapshot](#) (unsigned char *yuv_data, size_t yuv_data_length, int width, int height, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Processes the uncompressed YUV image stored in memory line by line. Same as ProcessYUVSnapshot with ROI, but with this method the ROI is full image.
- virtual [RecognitionResult ProcessImage](#) (const [Image](#) &image, const [Rectangle](#) &roi, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified smartid::Image.
- virtual [RecognitionResult ProcessImage](#) (const [Image](#) &image, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified smartid::Image. Same as ProcessImage with ROI, but with this method the ROI is full image.
- virtual [RecognitionResult ProcessImageFile](#) (const std::string &image_file, const [Rectangle](#) &roi, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified file.
- virtual [RecognitionResult ProcessImageFile](#) (const std::string &image_file, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified file. Same as ProcessImageFile with ROI, but with this method the ROI is full image.
- virtual [RecognitionResult ProcessImageData](#) (unsigned char *data, size_t data_length, const [Rectangle](#) &roi, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG)
- virtual [RecognitionResult ProcessImageData](#) (unsigned char *data, size_t data_length, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG), using ROI as full image.
- virtual [RecognitionResult ProcessImageDataBase64](#) (const std::string &base64_image_data, const [Rectangle](#) &roi, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG) encoded in base64.
- virtual [RecognitionResult ProcessImageDataBase64](#) (const std::string &base64_image_data, ImageOrientation image_orientation=[Landscape](#)) throw (std::exception)
Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG) encoded in base64.
- virtual [SessionState * GetSessionState](#) () const =0 throw (std::exception)
Gets session state object - optional information about OCR state.
- virtual void [Reset](#) ()=0
Resets the internal state of the session.

2.14.1 Detailed Description

[RecognitionSession](#) class - main interface for SmartID document recognition in videostream.

Definition at line 268 of file [smartid_engine.h](#).

2.14.2 Member Function Documentation

2.14.2.1 **virtual SessionState*** se::smartid::RecognitionSession::GetSessionState () const throw std::exception)
[pure virtual]

Gets session state object - optional information about OCR state.

Returns

[SessionState](#) object. Caller is responsible for deallocation.

Exceptions

<code>std::exception</code>	if the session state cannot be created
-----------------------------	--

2.14.2.2 **virtual RecognitionResult** se::smartid::RecognitionSession::ProcessImage (const Image & *image*, const Rectangle & *roi*, ImageOrientation *image_orientation* = Landscape) throw std::exception) [virtual]

Runs recognition process on the specified [smartid::Image](#).

Parameters

<i>image</i>	An Image to process
<i>roi</i>	Rectangle of interest (the system will not process anything outside this rectangle)
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If file doesn't exist or can't be processed, or if processing error occurs
-----------------------------	--

2.14.2.3 **virtual RecognitionResult** se::smartid::RecognitionSession::ProcessImage (const Image & *image*, ImageOrientation *image_orientation* = Landscape) throw std::exception) [virtual]

Runs recognition process on the specified [smartid::Image](#). Same as ProcessImage with ROI, but with this method the ROI is full image.

Parameters

<i>image</i>	An Image to process
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If file doesn't exist or can't be processed, or if processing error occurs
-----------------------------	--

2.14.2.4 `virtual RecognitionResult se::smartid::RecognitionSession::ProcessImageData (unsigned char * data, size_t data_length, const Rectangle & roi, ImageOrientation image_orientation = Landscape) throw std::exception`
[virtual]

Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG)

Parameters

<i>data</i>	Pointer to the (e.g. compressed) image data
<i>data_length</i>	Compressed image data length in bytes
<i>roi</i>	Rectangle of interest (the system will not process anything outside this rectangle)
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If image data can't be decoded or if processing error occurs
-----------------------------	--

2.14.2.5 `virtual RecognitionResult se::smartid::RecognitionSession::ProcessImageData (unsigned char * data, size_t data_length, ImageOrientation image_orientation = Landscape) throw std::exception` [virtual]

Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG), using ROI as full image.

Parameters

<i>data</i>	Pointer to the (e.g. compressed) image data
<i>data_length</i>	Compressed image data length in bytes
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If image data can't be decoded or if processing error occurs
-----------------------------	--

2.14.2.6 **virtual RecognitionResult** se::smartid::RecognitionSession::ProcessImageDataBase64 (const std::string & *base64_image_data*, const **Rectangle** & *roi*, **ImageOrientation** *image_orientation* = **Landscape**) throw std::exception) [virtual]

Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG) encoded in base64.

Parameters

<i>base64_image_data</i>	Encoded image
<i>roi</i>	Rectangle of interest (the system will not process anything outside this rectangle)
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<i>std::exception</i>	If image data can't be decoded or if processing error occurs
-----------------------	--

2.14.2.7 **virtual RecognitionResult** se::smartid::RecognitionSession::ProcessImageDataBase64 (const std::string & *base64_image_data*, **ImageOrientation** *image_orientation* = **Landscape**) throw std::exception) [virtual]

Runs recognition process on the specified image data (e.g. compressed with JPEG or PNG) encoded in base64.

Parameters

<i>base64_image_data</i>	Encoded image
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<i>std::exception</i>	If image data can't be decoded or if processing error occurs
-----------------------	--

2.14.2.8 **virtual RecognitionResult** se::smartid::RecognitionSession::ProcessImageFile (const std::string & *image_file*, const **Rectangle** & *roi*, **ImageOrientation** *image_orientation* = **Landscape**) throw std::exception) [virtual]

Runs recognition process on the specified file.

Parameters

<i>image_file</i>	Image file path
<i>roi</i>	Rectangle of interest (the system will not process anything outside this rectangle)
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<i>std::exception</i>	If file doesn't exist or can't be processed, or if processing error occurs
-----------------------	--

2.14.2.9 `virtual RecognitionResult se::smartid::RecognitionSession::ProcessImageFile (const std::string & image_file, ImageOrientation image_orientation = Landscape) throw std::exception) [virtual]`

Runs recognition process on the specified file. Same as ProcessImageFile with ROI, but with this method the ROI is full image.

Parameters

<i>image_file</i>	Image file path
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<i>std::exception</i>	If file doesn't exist or can't be processed, or if processing error occurs
-----------------------	--

2.14.2.10 `virtual RecognitionResult se::smartid::RecognitionSession::ProcessSnapshot (unsigned char * data, size_t data_length, int width, int height, int stride, int channels, const Rectangle & roi, ImageOrientation image_orientation = Landscape) throw std::exception) [pure virtual]`

Processes the uncompressed RGB image stored in memory line by line.

Parameters

<i>data</i>	Pointer to the data buffer beginning
<i>data_length</i>	Length of the data buffer
<i>width</i>	Image width
<i>height</i>	Image height
<i>stride</i>	Difference between the pointers to the consequent image lines, in bytes
<i>channels</i>	Number of channels (1, 3 or 4). 1-channel image is treated as grayscale image, 3-channel image is treated as RGB image, 4-channel image is treated as BGRA.
<i>roi</i>	Rectangle of interest (the system will not process anything outside this rectangle)
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If processing error occurs
-----------------------------	----------------------------

2.14.2.11 virtual **RecognitionResult** se::smartid::RecognitionSession::ProcessSnapshot (unsigned char * *data*, size_t *data_length*, int *width*, int *height*, int *stride*, int *channels*, ImageOrientation *image_orientation* = Landscape) throw std::exception) [virtual]

Processes the uncompressed RGB image stored in memory line by line. Same as ProcessSnapshot with ROI, but with this method the ROI is full image.

Parameters

<i>data</i>	Pointer to the data buffer beginning
<i>data_length</i>	Length of the data buffer
<i>width</i>	Image width
<i>height</i>	Image height
<i>stride</i>	Difference between the pointers to the consequent image lines, in bytes
<i>channels</i>	Number of channels (1, 3 or 4). 1-channel image is treated as grayscale image, 3-channel image is treated as RGB image, 4-channel image is treated as BGRA.
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If processing error occurs
-----------------------------	----------------------------

2.14.2.12 virtual **RecognitionResult** se::smartid::RecognitionSession::ProcessYUVSnapshot (unsigned char * *yuv_data*, size_t *yuv_data_length*, int *width*, int *height*, const [Rectangle](#) & *roi*, ImageOrientation *image_orientation* = Landscape) throw std::exception) [virtual]

Processes the uncompressed YUV image stored in memory line by line.

Parameters

<i>yuv_data</i>	Pointer to the data buffer start
<i>yuv_data_length</i>	Total length of image data buffer
<i>width</i>	Image width
<i>height</i>	Image height
<i>roi</i>	Rectangle of interest (the system will not process anything outside this rectangle)
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If processing error occurs
-----------------------------	----------------------------

2.14.2.13 virtual **RecognitionResult** **se::smartid::RecognitionSession::ProcessYUVSnapshot** (unsigned char * *yuv_data*, size_t *yuv_data_length*, int *width*, int *height*, ImageOrientation *image_orientation* = Landscape) throw **std::exception** [virtual]

Processes the uncompressed YUV image stored in memory line by line. Same as ProcessYUVSnapshot with ROI, but with this method the ROI is full image.

Parameters

<i>yuv_data</i>	Pointer to the data buffer start
<i>yuv_data_length</i>	Total length of image data buffer
<i>width</i>	Image width
<i>height</i>	Image height
<i>image_orientation</i>	Current image orientation to perform proper rotation to landscape

Returns

recognition result (integrated in the session)

Exceptions

<code>std::exception</code>	If processing error occurs
-----------------------------	----------------------------

2.15 se::smartid::Rectangle Class Reference

Class for representing a rectangle on an image.

Public Member Functions

- [Rectangle](#) ()
Constructor ($x = y = width = height = 0$)
- [~Rectangle](#) ()
Destructor.
- [Rectangle](#) (int *x*, int *y*, int *width*, int *height*)
Constructor from coordinates.

Public Attributes

- int *x*
x-coordinate of a top-left point in pixels
- int *y*
r-coordinate of a top-left point in pixels
- int *width*
rectangle width in pixels
- int *height*
rectangle height in pixels

2.15.1 Detailed Description

Class for representing a rectangle on an image.

Definition at line 36 of file [smartid_common.h](#).

2.15.2 Constructor & Destructor Documentation

2.15.2.1 se::smartid::Rectangle::Rectangle (int x, int y, int width, int height)

Constructor from coordinates.

Parameters

<i>x</i>	- Top-left rectangle point x-coordinate in pixels
<i>y</i>	- Top-left rectangle point y-coordinate in pixels
<i>width</i>	- Rectangle width in pixels
<i>height</i>	- Rectangle height in pixels

2.16 se::smartid::ResultReporterInterface Class Reference

Callback interface to obtain recognition results. Must be implemented to get the results as they appear during the stream processing.

Public Member Functions

- virtual void [SnapshotRejected](#) ()
Callback tells that last snapshot is not going to be processed/recognized. Optional.
- virtual void [FeedbackReceived](#) (const [ProcessingFeedback](#) &processing_feedback)
FeedbackReceived.
- virtual void [DocumentMatched](#) (const std::vector< [MatchResult](#) > &match_results)
Callback tells that last snapshot has valid document and contains document match result. Optional.
- virtual void [DocumentSegmented](#) (const std::vector< [SegmentationResult](#) > &segmentation_results)
Callback tells that last snapshot was segmented into raw fields for each match result. Optional.
- virtual void [SnapshotProcessed](#) (const [RecognitionResult](#) &recog_result)=0
Callback tells that last snapshot was processed successfully and returns current result. Required.
- virtual void [SessionEnded](#) ()
Internal callback to stop the session (determined by internal timer)
- virtual [~ResultReporterInterface](#) ()
Destructor.

2.16.1 Detailed Description

Callback interface to obtain recognition results. Must be implemented to get the results as they appear during the stream processing.

Definition at line 765 of file [smartid_result.h](#).

2.16.2 Member Function Documentation

2.16.2.1 `virtual void se::smartid::ResultReporterInterface::DocumentMatched (const std::vector< MatchResult > & match_results) [virtual]`

Callback tells that last snapshot has valid document and contains document match result. Optional.

Parameters

<i>match_results</i>	Document match result - vector of found templates
----------------------	---

2.16.2.2 `virtual void se::smartid::ResultReporterInterface::DocumentSegmented (const std::vector< SegmentationResult > & segmentation_results) [virtual]`

Callback tells that last snapshot was segmented into raw fields for each match result. Optional.

Parameters

<i>segmentation_results</i>	Segmentation results for each corresponding MatchResult
-----------------------------	---

2.16.2.3 `virtual void se::smartid::ResultReporterInterface::FeedbackReceived (const ProcessingFeedback & processing_feedback) [virtual]`

FeedbackReceived.

Parameters

<i>processing_feedback</i>	processing feedback data returned by the core
----------------------------	---

2.16.2.4 `virtual void se::smartid::ResultReporterInterface::SnapshotProcessed (const RecognitionResult & recog_result) [pure virtual]`

Callback tells that last snapshot was processed successfully and returns current result. Required.

Parameters

<i>recog_result</i>	Current recognition result
---------------------	----------------------------

2.17 se::smartid::SegmentationResult Class Reference

Class represents SmartID segmentation result containing found raw fields location information.

Public Member Functions

- [SegmentationResult](#) ()
Default constructor.

- [SegmentationResult](#) (const std::map< std::string, [Quadrangle](#) > &raw_fields_quadrangles, const std::map< std::string, [Quadrangle](#) > &raw_fields_template_quadrangles, bool accepted=false)
Main constructor.
- [~SegmentationResult](#) ()
Destructor.
- std::vector< std::string > [GetRawFieldsNames](#) () const
Getter for raw fields names which are keys for RawFieldQuadrangles map.
- bool [HasRawFieldQuadrangle](#) (const std::string &raw_field_name) const
Checks if there is a raw field quadrangle with given raw field name.
- const [Quadrangle](#) & [GetRawFieldQuadrangle](#) (const std::string &raw_field_name) const throw (std::exception)
Get raw field quadrangle for raw field name.
- const std::map< std::string, [Quadrangle](#) > & [GetRawFieldQuadrangles](#) () const
Getter for raw field quadrangles (raw field name -> quadrangle).
- const [Quadrangle](#) & [GetRawFieldTemplateQuadrangle](#) (const std::string &raw_field_name) const throw (std::exception)
Get raw field quadrangle for raw field name in template coordinates.
- const std::map< std::string, [Quadrangle](#) > & [GetRawFieldTemplateQuadrangles](#) () const
Getter for raw field quadrangles in template coordinates (raw field name -> quadrangle).
- bool [GetAccepted](#) () const
Getter for accepted field.

Private Attributes

- std::map< std::string, [Quadrangle](#) > [raw_field_quadrangles_](#)
[raw field name, quadrangle]
- std::map< std::string, [Quadrangle](#) > [raw_field_template_quadrangles_](#)
[raw field name, quadrangle in template coords]
- bool [accepted_](#)
Whether this result is ready to be visualized.

2.17.1 Detailed Description

Class represents SmartID segmentation result containing found raw fields location information.

Definition at line 425 of file [smartid_result.h](#).

2.17.2 Member Function Documentation

- 2.17.2.1 **const [Quadrangle](#)& se::smartid::SegmentationResult::GetRawFieldQuadrangle (const std::string & raw_field_name) const throw std::exception)**

Get raw field quadrangle for raw field name.

Parameters

<i>raw_field_name</i>	Raw field name
-----------------------	----------------

Returns

Raw field quadrangle for raw field name

Exceptions

<code>std::invalid_argument</code>	if <code>raw_field_name</code> is not present in raw field quadrangles
------------------------------------	--

2.17.2.2 `const Quadrangle& se::smartid::SegmentationResult::GetRawFieldTemplateQuadrangle (const std::string & raw_field_name) const throw std::exception)`

Get raw field quadrangle for raw field name in template coordinates.

Parameters

<code>raw_field_name</code>	Raw field name
-----------------------------	----------------

Returns

Raw field quadrangle for raw field name in template coordinates

Exceptions

<code>std::invalid_argument</code>	if <code>raw_field_name</code> is not present in raw field quadrangles
------------------------------------	--

2.18 se::smartid::SessionSettings Class Reference

The [SessionSettings](#) class - runtime parameters of the recognition session.

Public Member Functions

- virtual [~SessionSettings](#) ()
SessionSettings dtor.
- virtual [SessionSettings * Clone](#) () const =0
Clones session settings and creates a new object on heap.
- const std::vector< std::string > & [GetEnabledDocumentTypes](#) () const
Get enabled document types with which recognition session will be created.
- void [AddEnabledDocumentTypes](#) (const std::string &doctype_mask)
Add enabled document types conforming to [GetSupportedDocumentTypes\(\)](#). Both exact string type names or wildcard expression can be used, for example: "rus.passport.national", "rus.", "/*.passport.*", "*".*
- void [RemoveEnabledDocumentTypes](#) (const std::string &doctype_mask)
Remove enabled document types conforming to [GetEnabledDocumentTypes\(\)](#). Both exact string type names or wildcard expression can be used, for example: "rus.passport.national", "rus.", "/*.passport.*", "*".*
- void [SetEnabledDocumentTypes](#) (const std::vector< std::string > &document_types)
Set enabled document types. Clears all enabled types and then calls [AddEnabledDocumentTypes\(\)](#) for each document type in the document_types.
- const std::vector< std::vector< std::string > > & [GetSupportedDocumentTypes](#) () const

Gets all supported document types for each engine of configured bundle. Recognition session can only be spawned with the set of document types corresponding to some single engine.

- `const std::map< std::string, std::string > & GetOptions () const`
Get full map of additional session settings.
- `std::map< std::string, std::string > & GetOptions ()`
Get full map of additional session settings.
- `std::vector< std::string > GetOptionNames () const`
Get all option names.
- `bool HasOption (const std::string &name) const`
Checks is there is a set additional option by name.
- `const std::string & GetOption (const std::string &name) const throw (std::exception)`
Get an additional option value by name.
- `void SetOption (const std::string &name, const std::string &value)`
Set(modify) an additional option value by name.
- `void RemoveOption (const std::string &name) throw (std::exception)`
Remove an option from session settings (by name)
- `const std::map< std::string, std::vector< std::string > > & GetEnabledFieldNames () const`
Get list of enabled fields for document type.
- `void EnableField (const std::string &doctype, const std::string &field_name)`
Enable fields by name.
- `void DisableField (const std::string &doctype, const std::string &field_name)`
Disable string fields by name.
- `void SetEnabledFields (const std::string &doctype, const std::vector< std::string > &field_names)`
Set(modify) an enabled string fields by names.
- `const std::vector< std::string > & GetSupportedFieldNames (const std::string &doctype) throw (std::exception)`
Get set of enabled string fields.
- `const std::map< std::string, std::vector< std::string > > & GetEnabledForensicFieldNames () const`
Get list of enabled document forensics field for given document type.
- `void EnableForensicField (const std::string &doctype, const std::string &field_name)`
Enable document forensic fields by name.
- `void DisableForensicField (const std::string &doctype, const std::string &field_name)`
Disable document forensic fields by name.
- `void SetEnabledForensicFields (const std::string &doctype, const std::vector< std::string > &field_names)`
Set(modify) an enabled document forensic fields by names.
- `const std::vector< std::string > & GetSupportedForensicFieldNames (const std::string &doctype) throw (std::exception)`
Get set of enabled document forensic fields.
- `const std::string & GetCurrentMode () const`
Returns current bundle mode.
- `void SetCurrentMode (const std::string &mode) throw (std::exception)`
Sets current bundle mode.
- `const std::vector< std::string > & GetAvailableModes () const`
Gets list of available bundle mode names.

Protected Member Functions

- `SessionSettings ()`
Disabled default constructor - use [RecognitionEngine](#) factory method instead.

Protected Attributes

- `std::vector< std::string >` **supported_modes_**
- `std::string` **current_mode_**
- `std::map< std::string, std::vector< std::vector< std::string > > >` **supported_document_types_**
- `std::map< std::string, std::vector< std::string > >` **enabled_document_types_**
- `std::map< std::string, std::string >` **options_**
- `std::map< std::string, std::map< std::string, std::vector< std::string > > >` **supported_fields_**
- `std::map< std::string, std::map< std::string, std::vector< std::string > > >` **enabled_fields_**
- `std::map< std::string, std::map< std::string, std::vector< std::string > > >` **supported_forensic_fields_**
- `std::map< std::string, std::map< std::string, std::vector< std::string > > >` **enabled_forensic_fields_**

2.18.1 Detailed Description

The [SessionSettings](#) class - runtime parameters of the recognition session.

Definition at line 43 of file [smartid_engine.h](#).

2.18.2 Member Function Documentation

2.18.2.1 `void se::smartid::SessionSettings::AddEnabledDocumentTypes (const std::string & doctype_mask)`

Add enabled document types conforming to [GetSupportedDocumentTypes\(\)](#). Both exact string type names or wildcard expression can be used, for example: "rus.passport.national", "rus.*", "/*.passport.*", "*".

Parameters

<i>doctype_mask</i>	Document type name or wildcard expression
---------------------	---

2.18.2.2 `virtual SessionSettings* se::smartid::SessionSettings::Clone () const` [pure virtual]

Clones session settings and creates a new object on heap.

Returns

new allocated object which is a copy of this

2.18.2.3 `void se::smartid::SessionSettings::DisableField (const std::string & doctype, const std::string & field_name)`

Disable string fields by name.

Parameters

<i>doctype</i>	- type of document
<i>field_name</i>	- name of field

2.18.2.4 void se::smartid::SessionSettings::DisableForensicField (const std::string & *doctype*, const std::string & *field_name*)

Disable document forensic fields by name.

Parameters

<i>doctype</i>	- type of document
<i>field_name</i>	- name of field

2.18.2.5 void se::smartid::SessionSettings::EnableField (const std::string & *doctype*, const std::string & *field_name*)

Enable fields by name.

Parameters

<i>doctype</i>	- type of document
<i>field_name</i>	- name of field

2.18.2.6 void se::smartid::SessionSettings::EnableForensicField (const std::string & *doctype*, const std::string & *field_name*)

Enable document forensic fields by name.

Parameters

<i>doctype</i>	- type of document
<i>field_name</i>	- name of field

2.18.2.7 const std::vector<std::string>& se::smartid::SessionSettings::GetAvailableModes () const

Gets list of available bundle mode names.

Returns

list of available modes

2.18.2.8 const std::string& se::smartid::SessionSettings::GetCurrentMode () const

Returns current bundle mode.

Returns

string name of current bundle mode

2.18.2.9 const std::vector<std::string>& se::smartid::SessionSettings::GetEnabledDocumentTypes () const

Get enabled document types with which recognition session will be created.

Returns

a vector of enabled document types (exact types without wildcards)

2.18.2.10 const std::string& se::smartid::SessionSettings::GetOption (const std::string & *name*) const throw std::exception)

Get an additional option value by name.

Parameters

<i>name</i>	- string representation of option name
-------------	--

Returns

string value of an option

Exceptions

<i>std::invalid_argument</i>	if there is no such option
------------------------------	----------------------------

2.18.2.11 `std::vector<std::string> se::smartid::SessionSettings::GetOptionNames () const`

Get all option names.

Returns

vector of all additional option names

2.18.2.12 `const std::map<std::string, std::string>& se::smartid::SessionSettings::GetOptions () const`

Get full map of additional session settings.

Returns

constref map of additional options

Option name is a string consisting of two components: <INTERNAL_ENGINE>.<OPTION_NAME>. Option value syntax is dependent on the option.

2.18.2.13 `std::map<std::string, std::string>& se::smartid::SessionSettings::GetOptions ()`

Get full map of additional session settings.

Returns

ref map of additional options

2.18.2.14 `const std::vector<std::vector<std::string> >& se::smartid::SessionSettings::GetSupportedDocumentTypes () const`

Gets all supported document types for each engine of configured bundle. Recognition session can only be spawned with the set of document types corresponding to some single engine.

Returns

[engine][i_doctype_string] two dimensional vector const ref

2.18.2.15 `const std::vector<std::string>& se::smartid::SessionSettings::GetSupportedFieldNames (const std::string & doctype) throw std::exception)`

Get set of enabled string fields.

Parameters

<i>doctype</i>	- type of document
----------------	--------------------

Returns

list of supported field names for document

Exceptions

<i>std::invalid_argument</i>	if there is no such document
------------------------------	------------------------------

2.18.2.16 `const std::vector<std::string> & se::smartid::SessionSettings::GetSupportedForensicFieldNames (const std::string & doctype) throw std::exception)`

Get set of enabled document forensic fields.

Parameters

<i>doctype</i>	- type of document
----------------	--------------------

Returns

list of supported field names for document

Exceptions

<i>std::invalid_argument</i>	if there is no such document
------------------------------	------------------------------

2.18.2.17 `bool se::smartid::SessionSettings::HasOption (const std::string & name) const`

Checks is there is a set additional option by name.

Parameters

<i>name</i>	- string representation of option name
-------------	--

Returns

true if there is a set option with provided name

2.18.2.18 `void se::smartid::SessionSettings::RemoveEnabledDocumentTypes (const std::string & doctype_mask)`

Remove enabled document types conforming to [GetEnabledDocumentTypes\(\)](#). Both exact string type names or wildcard expression can be used, for example: "rus.passport.national", "rus.*", "/*.passport.*", "*".

Parameters

<i>doctype_mask</i>	Document type name or wildcard expression
---------------------	---

2.18.2.19 void se::smartid::SessionSettings::RemoveOption (const std::string & *name*) throw std::exception)

Remove an option from session settings (by name)

Parameters

<i>name</i>	- string representation of option name
-------------	--

Exceptions

<i>std::invalid_argument</i>	if there is no such option
------------------------------	----------------------------

2.18.2.20 void se::smartid::SessionSettings::SetCurrentMode (const std::string & *mode*) throw std::exception)

Sets current bundle mode.

Parameters

<i>mode</i>	- string name of new current bundle mode
-------------	--

2.18.2.21 void se::smartid::SessionSettings::SetEnabledDocumentTypes (const std::vector< std::string > & *document_types*)

Set enabled document types. Clears all enabled types and then calls [AddEnabledDocumentTypes\(\)](#) for each document type in the *document_types*.

Parameters

<i>document_types</i>	a vector of enabled document types
-----------------------	------------------------------------

2.18.2.22 void se::smartid::SessionSettings::SetEnabledFields (const std::string & *doctype*, const std::vector< std::string > & *field_names*)

Set(modify) an enabled string fields by names.

Parameters

<i>doctype</i>	- type of document
<i>field_names</i>	- list of string field names

2.18.2.23 void se::smartid::SessionSettings::SetEnabledForensicFields (const std::string & *doctype*, const std::vector< std::string > & *field_names*)

Set(modify) an enabled document forensic fields by names.

Parameters

<i>doctype</i>	- type of document
<i>field_names</i>	- list of string field names

2.18.2.24 void se::smartid::SessionSettings::SetOption (const std::string & *name*, const std::string & *value*)

Set(modify) an additional option value by name.

Parameters

<i>name</i>	- string representation of option name
<i>value</i>	- value of option to set

2.19 se::smartid::SessionState Class Reference

[SessionState](#) class - optional recognition session information.

Public Member Functions

- std::vector< std::string > [GetIntegratedFieldStateNames](#) () const
Returns a vector of unique integrated field state names.
- bool [HasIntegratedFieldState](#) (const std::string &name) const
Checks if there is an integrated field state with given name.
- const [IntegratedFieldState](#) & [GetStringFieldState](#) (const std::string &name) const throw (std::exception)
Gets integrated field state by name.
- const std::map< std::string, [IntegratedFieldState](#) > & [GetIntegratedFieldStates](#) () const
Getter for integrated field states map.
- std::map< std::string, [IntegratedFieldState](#) > & [GetIntegratedFieldStates](#) ()
Getter for (mutable) integrated field states map.
- void [SetIntegratedFieldStates](#) (const std::map< std::string, [IntegratedFieldState](#) > &integrated_field_states)
Setter for integrated field states map.
- int [GetSnapshotsProcessed](#) () const

Protected Member Functions

- [SessionState](#) (int snapshots_processed)
Disabled default constructor - use ... instead.

Protected Attributes

- std::map< std::string, [IntegratedFieldState](#) > [integrated_field_states_](#)
- int [snapshots_processed_](#)

2.19.1 Detailed Description

[SessionState](#) class - optional recognition session information.

Definition at line 836 of file [smartid_result.h](#).

2.19.2 Member Function Documentation

2.19.2.1 `const std::map<std::string, IntegratedFieldState>& se::smartid::SessionState::GetIntegratedFieldStates ()`
`const`

Getter for integrated field states map.

Returns

constref for integrated field states map

2.19.2.2 `std::map<std::string, IntegratedFieldState>& se::smartid::SessionState::GetIntegratedFieldStates ()`

Getter for (mutable) integrated field states map.

Returns

ref for integrated field states map

2.19.2.3 `const IntegratedFieldState& se::smartid::SessionState::GetStringFieldState (const std::string & name)` `const`
`throw std::exception)`

Gets integrated field state by name.

Parameters

<i>name</i>	- name of an integrated field state
-------------	-------------------------------------

Exceptions

<code>std::invalid_argument</code>	if there is no such field state
------------------------------------	---------------------------------

2.19.2.4 `void se::smartid::SessionState::SetIntegratedFieldStates (const std::map< std::string, IntegratedFieldState > & integrated_field_states)`

Setter for integrated field states map.

Parameters

<i>integrated_field_states</i>	- integrated field states map
--------------------------------	-------------------------------

2.20 se::smartid::StringField Class Reference

Class represents implementation of SmartID document Field for string fields.

Public Member Functions

- [StringField](#) ()
Default constructor.
- [StringField](#) (const std::string &name, const [OcrString](#) &value, bool is_accepted, double confidence, const std::map< std::string, std::string > &attributes={}) throw (std::exception)
StringField main ctor.
- [StringField](#) (const std::string &name, const std::string &value, bool is_accepted, double confidence, const std::map< std::string, std::string > &attributes={}) throw (std::exception)
StringField ctor from utf8-string value.
- [~StringField](#) ()
Destructor.
- const std::string & [GetName](#) () const
Getter for string field name.
- const [OcrString](#) & [GetValue](#) () const
Getter for string field value (OcrString representation)
- std::string [GetUtf8Value](#) () const
Getter for string field value (Utf8-string representation)
- bool [IsAccepted](#) () const
Whether the system is confident in field recognition result.
- double [GetConfidence](#) () const
The system's confidence level in field recognition result (in range [0..1])
- std::vector< std::string > [GetAttributeNames](#) () const
Returns a vector of attribute names.
- const std::map< std::string, std::string > & [GetAttributes](#) () const
Getter for attributes map.
- bool [HasAttribute](#) (const std::string &attribute_name) const
Check if attribute with given name is present.
- const std::string & [GetAttribute](#) (const std::string &attribute_name) const throw (std::exception)
Get attribute value by its name.

Private Attributes

- std::string [name_](#)
Field name.
- [OcrString](#) [value_](#)
Fields' OcrString value.
- bool [is_accepted_](#)
Specifies whether the system is confident in field recognition result.
- double [confidence_](#)
Specifies the system's confidence level in field recognition result.
- std::map< std::string, std::string > [attributes_](#)
Field attributes.

2.20.1 Detailed Description

Class represents implementation of SmartID document Field for string fields.

Definition at line 167 of file [smartid_result.h](#).

2.20.2 Constructor & Destructor Documentation

2.20.2.1 `se::smartid::StringField::StringField (const std::string & name, const OcrString & value, bool is_accepted, double confidence, const std::map< std::string, std::string > & attributes = { }) throw std::exception)`

[StringField](#) main ctor.

Parameters

<i>name</i>	- name of the field
<i>value</i>	- OcrString-representation of the field value
<i>is_accepted</i>	- whether the system is confident in the field's value
<i>confidence</i>	- system's confidence level in fields' value in range [0..1\
<i>attributes</i>	- additional field information

Exceptions

<code>std::invalid_argument</code>	if confidence value is not in range [0..1]
------------------------------------	--

2.20.2.2 `se::smartid::StringField::StringField (const std::string & name, const std::string & value, bool is_accepted, double confidence, const std::map< std::string, std::string > & attributes = { }) throw std::exception)`

[StringField](#) ctor from utf8-string value.

Parameters

<i>name</i>	- name of the field
<i>value</i>	- utf8-string representation of the field value
<i>is_accepted</i>	- whether the system is confident in the field's value
<i>confidence</i>	- system's confidence level in fields' value in range [0..1]
<i>attributes</i>	- additional field information

Exceptions

<code>std::invalid_argument</code>	if confidence value is not in range [0..1] or if failed to decode utf8-string 'value'
------------------------------------	---

2.20.3 Member Function Documentation

2.20.3.1 `const std::string& se::smartid::StringField::GetAttribute (const std::string & attribute_name) const throw std::exception)`

Get attribute value by its name.

Parameters

<i>attribute_name</i>	key attribute name
-----------------------	--------------------

Returns

attribute value by its name

2.20.3.2 `bool se::smartid::StringField::HasAttribute (const std::string & attribute_name) const`

Check if attribute with given name is present.

Parameters

<i>attribute_name</i>	attribute name to check presence of
-----------------------	-------------------------------------

Returns

true if attribute with given name is present

3 File Documentation

3.1 smartid_common.h File Reference

Common classes used in SmartIdEngine.

Classes

- class [se::smartid::Rectangle](#)
Class for representing a rectangle on an image.
- class [se::smartid::Point](#)
Class for representing a point on an image.
- class [se::smartid::Quadrangle](#)
Class for representing a quadrangle on an image.
- class [se::smartid::Image](#)
Class for representing a bitmap image.

Variables

- [Landscape](#)
image is in the proper orientation, nothing needs to be done
- [Portrait](#)
image is in portrait, needs to be rotated 90° clockwise
- [InvertedLandscape](#)
image is upside-down, needs to be rotated 180°

3.1.1 Detailed Description

Common classes used in SmartIdEngine.

Definition in file [smartid_common.h](#).

3.2 smartid_common.h

```

00001 /*
00002 Copyright (c) 2012-2017, Smart Engines Ltd
00003 All rights reserved.
00004 */
00005
00011 #ifndef SMARTID_ENGINE_SMARTID_COMMON_H_INCLUDED_
00012 #define SMARTID_ENGINE_SMARTID_COMMON_H_INCLUDED_
00013
00014 #if defined _MSC_VER
00015 #pragma warning(push)
00016 #pragma warning(disable : 4290)
00017 #endif
00018
00019 #if defined _WIN32 && SMARTID_ENGINE_EXPORTS
00020 # define SMARTID_DLL_EXPORT __declspec(dllexport)
00021 #else
00022 # if defined(__clang__) || defined(__GNUC__)
00023 #  define SMARTID_DLL_EXPORT __attribute__((visibility ("default")))
00024 # else
00025 #  define SMARTID_DLL_EXPORT
00026 # endif
00027 #endif
00028
00029 #include <stdexcept>
00030
00031 namespace se { namespace smartid {
00032
00036 class SMARTID_DLL_EXPORT Rectangle {
00037 public:
00041     Rectangle();
00042
00046     ~Rectangle();
00047
00055     Rectangle(int x, int y, int width, int height);
00056
00057 public:
00058     int x;
00059     int y;
00060     int width;
00061     int height;
00062 };
00063
00067 class SMARTID_DLL_EXPORT Point {
00068 public:
00072     Point();
00073
00077     ~Point();
00078
00084     Point(double x, double y);
00085
00086     double x;
00087     double y;
00088 };
00089
00093 class SMARTID_DLL_EXPORT Quadrangle {
00094 public:
00098     Quadrangle();
00099
00103     ~Quadrangle();
00104
00112     Quadrangle(Point a, Point b, Point c, Point d);
00113
00121     Point& operator[](int index) throw(std::exception);
00122
00130     const Point& operator[](int index) const throw(std::exception);
00131
00139     const Point& GetPoint(int index) const throw(std::exception);
00140
00149     void SetPoint(int index, const Point& value) throw(std::exception);
00150
00155     Rectangle GetBoundingRectangle() const;
00156

```

```

00157 private:
00160     Point points[4];
00161 };
00162
00166 class SMARTID_DLL_EXPORT Image {
00167 public:
00169     Image();
00170
00177     Image(const std::string& image_filename) throw(std::exception);
00178
00193     Image(unsigned char* data, size_t data_length, int width, int height,
00194           int stride, int channels) throw(std::exception);
00195
00205     Image(unsigned char* yuv_data, size_t yuv_data_length,
00206           int width, int height) throw(std::exception);
00207
00214     Image(const Image& copy);
00215
00222     Image& operator=(const Image& other);
00223
00225     ~Image();
00226
00234     void Save(const std::string& image_filename) const throw(std::exception);
00235
00240     int GetRequiredBufferLength() const;
00241
00253     int CopyToBuffer(
00254         char* out_buffer, int buffer_length) const throw(std::exception);
00255
00261     double EstimateFocusScore(double quantile = 0.95) const throw(std::exception);
00262
00270     int GetRequiredBase64BufferLength() const throw(std::exception);
00271
00283     int CopyBase64ToBuffer(
00284         char* out_buffer, int buffer_length) const throw(std::exception);
00285
00293     // TODO
00294     std::string GetBase64String() const throw(std::exception);
00295
00299     void Clear();
00300
00305     int GetWidth() const;
00306
00311     int GetHeight() const;
00312
00317     int GetStride() const;
00318
00323     int GetChannels() const;
00324
00329     bool IsMemoryOwner() const;
00330
00335     void ForceMemoryOwner() throw(std::exception);
00336
00342     void Resize(int new_width, int new_height) throw(std::exception);
00343
00348     void Crop(const Quadrangle& quad) throw(std::exception);
00349
00356     void Crop(const Quadrangle& quad, int width, int height) throw(std::exception);
00357
00363     void MaskImageRegionRectangle(Rectangle rect, int pixel_expand = 0) throw(std::exception);
00364
00370     void MaskImageRegionQuadrangle(Quadrangle quad, int pixel_expand = 0) throw(std::exception);
00371
00375     void FlipVertical() throw(std::exception);
00376
00380     void FlipHorizontal() throw(std::exception);
00381
00382 public:
00383     char* data;
00384     int width;
00385     int height;
00386     int stride;
00387     int channels;
00388     bool memown;
00389 };
00390
00394 enum SMARTID_DLL_EXPORT ImageOrientation {
00395     Landscape,
00396     Portrait,
00397     InvertedLandscape,
00398     InvertedPortrait
00399 };
00401
00402 } } // namespace se::smartid
00403
00404 #if defined _MSC_VER
00405 #pragma warning(pop)

```



```

00406 #endif
00407
00408 #endif // SMARTID_ENGINE_SMARTID_COMMON_H_INCLUDED

```

3.3 smartid_engine.h File Reference

Main processing classes.

Classes

- class [se::smartid::SessionSettings](#)
The *SessionSettings* class - runtime parameters of the recognition session.
- class [se::smartid::RecognitionSession](#)
The *RecognitionSession* class - main interface for SmartID document recognition in videostream.
- class [se::smartid::RecognitionEngine](#)
The *RecognitionEngine* class - a factory for RecognitionSessions, holds configured internal engines.

3.3.1 Detailed Description

Main processing classes.

Copyright (c) 2012-2017, Smart Engines Ltd All rights reserved.

Definition in file [smartid_engine.h](#).

3.4 smartid_engine.h

```

00001
00011 #ifndef SMARTID_ENGINE_SMARTID_ENGINE_H_INCLUDED_
00012 #define SMARTID_ENGINE_SMARTID_ENGINE_H_INCLUDED_
00013
00014 #if defined _MSC_VER
00015 #pragma warning(push)
00016 #pragma warning(disable : 4290)
00017 #endif
00018
00019 #include <string>
00020 #include <vector>
00021
00022 #include "smartid_common.h"
00023 #include "smartid_result.h"
00024
00037 namespace se { namespace smartid {
00038
00043 class SMARTID_DLL_EXPORT SessionSettings {
00044 public:
00046     virtual ~SessionSettings();
00047
00052     virtual SessionSettings * Clone() const = 0;
00053
00058     const std::vector<std::string>& GetEnabledDocumentTypes() const;
00059
00066     void AddEnabledDocumentTypes(const std::string &doctype_mask);
00067
00074     void RemoveEnabledDocumentTypes(const std::string &doctype_mask);
00075
00081     void SetEnabledDocumentTypes(const std::vector<std::string>& document_types);
00082
00089     const std::vector<std::vector<std::string> >& GetSupportedDocumentTypes() const;
00090
00099     const std::map<std::string, std::string>& GetOptions() const;
00100
00105     std::map<std::string, std::string>& GetOptions();
00106

```

```

00111     std::vector<std::string> GetOptionNames() const;
00112
00118     bool HasOption(const std::string& name) const;
00119
00127     const std::string& GetOption(
00128         const std::string& name) const throw(std::exception);
00129
00135     void SetOption(const std::string& name, const std::string& value);
00136
00143     void RemoveOption(const std::string& name) throw(std::exception);
00144
00148     const std::map<std::string, std::vector<std::string> >&
00149         GetEnabledFieldNames() const;
00150
00156     void EnableField(const std::string& doctype, const std::string& field_name);
00157
00163     void DisableField(const std::string& doctype, const std::string& field_name);
00164
00170     void SetEnabledFields(
00171         const std::string& doctype, const std::vector<std::string>& field_names);
00172
00180     const std::vector<std::string>& GetSupportedFieldNames(
00181         const std::string& doctype) throw(std::exception);
00182
00186     const std::map<std::string, std::vector<std::string> >&
00187         GetEnabledForensicFieldNames() const;
00188
00194     void EnableForensicField(const std::string& doctype,
00195                             const std::string& field_name);
00196
00202     void DisableForensicField(const std::string& doctype,
00203                             const std::string& field_name);
00204
00210     void SetEnabledForensicFields(
00211         const std::string& doctype, const std::vector<std::string>& field_names);
00212
00220     const std::vector<std::string>& GetSupportedForensicFieldNames(
00221         const std::string& doctype) throw(std::exception);
00222
00227     const std::string& GetCurrentMode() const;
00228
00233     void SetCurrentMode(const std::string& mode) throw(std::exception);
00234
00239     const std::vector<std::string>& GetAvailableModes() const;
00240
00241 protected:
00242     std::vector<std::string> supported_modes_;
00243     std::string current_mode_;
00244
00245     std::map<std::string, std::vector<std::vector<std::string> > >
00246         supported_document_types_;
00247     std::map<std::string, std::vector<std::string> > enabled_document_types_;
00248
00249     std::map<std::string, std::string> options_;
00250     std::map<std::string, std::map<std::string, std::vector<std::string> > >
00251         supported_fields_;
00252     std::map<std::string, std::map<std::string, std::vector<std::string> > >
00253         enabled_fields_;
00254
00255     std::map<std::string, std::map<std::string, std::vector<std::string> > >
00256         supported_forensic_fields_;
00257     std::map<std::string, std::map<std::string, std::vector<std::string> > >
00258         enabled_forensic_fields_;
00259
00261     SessionSettings();
00262 };
00263
00268 class SMARTID_DLL_EXPORT RecognitionSession {
00269 public:
00271     virtual ~RecognitionSession();
00272
00293     virtual RecognitionResult ProcessSnapshot(
00294         unsigned char* data,
00295         size_t data_length,
00296         int width,
00297         int height,
00298         int stride,
00299         int channels,
00300         const Rectangle& roi,
00301         ImageOrientation image_orientation = Landscape) throw(std::exception) = 0;
00302
00323     virtual RecognitionResult ProcessSnapshot(
00324         unsigned char* data,
00325         size_t data_length,
00326         int width,
00327         int height,
00328         int stride,

```

```

00329         int channels,
00330         ImageOrientation image_orientation = Landscape) throw(std::exception);
00331
00346     virtual RecognitionResult ProcessYUVSnapshot(
00347         unsigned char* yuv_data,
00348         size_t yuv_data_length,
00349         int width,
00350         int height,
00351         const Rectangle& roi,
00352         ImageOrientation image_orientation = Landscape) throw(std::exception);
00353
00368     virtual RecognitionResult ProcessYUVSnapshot(
00369         unsigned char* yuv_data,
00370         size_t yuv_data_length,
00371         int width,
00372         int height,
00373         ImageOrientation image_orientation = Landscape) throw(std::exception);
00374
00387     virtual RecognitionResult ProcessImage(
00388         const Image& image,
00389         const Rectangle& roi,
00390         ImageOrientation image_orientation = Landscape) throw(std::exception);
00391
00404     virtual RecognitionResult ProcessImage(
00405         const Image& image,
00406         ImageOrientation image_orientation = Landscape) throw(std::exception);
00407
00420     virtual RecognitionResult ProcessImageFile(
00421         const std::string& image_file,
00422         const Rectangle& roi,
00423         ImageOrientation image_orientation = Landscape) throw(std::exception);
00424
00437     virtual RecognitionResult ProcessImageFile(
00438         const std::string& image_file,
00439         ImageOrientation image_orientation = Landscape) throw(std::exception);
00440
00455     virtual RecognitionResult ProcessImageData(
00456         unsigned char* data,
00457         size_t data_length,
00458         const Rectangle& roi,
00459         ImageOrientation image_orientation = Landscape) throw(std::exception);
00460
00473     virtual RecognitionResult ProcessImageData(
00474         unsigned char* data,
00475         size_t data_length,
00476         ImageOrientation image_orientation = Landscape) throw(std::exception);
00477
00491     virtual RecognitionResult ProcessImageDataBase64(
00492         const std::string& base64_image_data,
00493         const Rectangle& roi,
00494         ImageOrientation image_orientation = Landscape) throw(std::exception);
00495
00507     virtual RecognitionResult ProcessImageDataBase64(
00508         const std::string& base64_image_data,
00509         ImageOrientation image_orientation = Landscape) throw(std::exception);
00510
00516     virtual SessionState* GetSessionState() const throw(std::exception) = 0;
00517
00521     virtual void Reset() = 0;
00522 };
00523
00528 class SMARTID_DLL_EXPORT RecognitionEngine {
00529 public:
00538     RecognitionEngine(const std::string& config_path,
00539         bool lazy_configuration = true) throw(std::exception);
00540
00551     RecognitionEngine(unsigned char* config_data,
00552         size_t data_length,
00553         bool lazy_configuration = true) throw(std::exception);
00554
00556     ~RecognitionEngine();
00557
00564     SessionSettings* CreateSessionSettings() const throw(std::exception);
00565
00567
00578     RecognitionSession* SpawnSession(
00579         const SessionSettings& session_settings,
00580         ResultReporterInterface* result_reporter = 0) const throw(std::exception);
00581
00586     static std::string GetVersion();
00587
00588 private:
00590     RecognitionEngine(const RecognitionEngine& copy);
00592     void operator=(const RecognitionEngine& other);
00593
00594 private:
00595     class RecognitionEngineImpl* pimpl_;

```

```

00596 };
00597 } } // namespace se::smartid
00598
00599 #if defined _MSC_VER
00600 #pragma warning(pop)
00601 #endif
00602
00603 #endif // SMARTID_ENGINE_SMARTID_ENGINE_H_INCLUDED

```

3.5 smartid_result.h File Reference

Recognition result classes.

Classes

- class [se::smartid::OcrCharVariant](#)
Possible character recognition result.
- class [se::smartid::OcrChar](#)
Contains all OCR information for a given character.
- class [se::smartid::OcrString](#)
Contains additional OCR information for the whole string.
- class [se::smartid::StringField](#)
Class represents implementation of SmartID document Field for string fields.
- class [se::smartid::ImageField](#)
Class represents implementation of SmartIDField for list of images.
- class [se::smartid::MatchResult](#)
Class represents SmartID match result.
- class [se::smartid::ForensicField](#)
Class represents implementation of SmartID forensic field for document validity checks.
- class [se::smartid::SegmentationResult](#)
Class represents SmartID segmentation result containing found raw fields location information.
- class [se::smartid::RecognitionResult](#)
Class represents SmartID recognition result.
- class [se::smartid::ProcessingFeedback](#)
Feedback data that is returned by the [ResultReporterInterface](#)'s FeedbackReceived method, containing useful user-oriented information such as additional visualization, advisory information etc.
- class [se::smartid::ResultReporterInterface](#)
Callback interface to obtain recognition results. Must be implemented to get the results as they appear during the stream processing.
- class [se::smartid::IntegratedFieldState](#)
[IntegratedFieldState](#) class - integrated field terminality state.
- class [se::smartid::SessionState](#)
[SessionState](#) class - optional recognition session information.

3.5.1 Detailed Description

Recognition result classes.

Copyright (c) 2012-2017, Smart Engines Ltd All rights reserved.

Definition in file [smartid_result.h](#).

3.6 smartid_result.h

```

00001
00011 #ifndef SMARTID_ENGINE_SMARTID_RESULT_H_INCLUDED_
00012 #define SMARTID_ENGINE_SMARTID_RESULT_H_INCLUDED_
00013
00014 #if defined _MSC_VER
00015 #pragma warning(push)
00016 #pragma warning(disable : 4290)
00017 #endif
00018
00019 #include "smartid_common.h"
00020
00021 #include <cstdint>
00022 #include <map>
00023 #include <string>
00024 #include <vector>
00025
00026 namespace se { namespace smartid {
00027
00031 class SMARTID_DLL_EXPORT OcrCharVariant {
00032 public:
00036     OcrCharVariant();
00037
00039     ~OcrCharVariant();
00040
00048     OcrCharVariant(uint16_t utf16_char, double confidence) throw(std::exception);
00049
00059     OcrCharVariant(const std::string& utf8_char,
00060                   double confidence) throw(std::exception);
00061
00063     uint16_t GetUtf16Character() const;
00065     std::string GetUtf8Character() const;
00067     double GetConfidence() const;
00068
00069 private:
00070     uint16_t character_;
00071     double confidence_;
00072 };
00073
00077 class SMARTID_DLL_EXPORT OcrChar {
00078 public:
00082     OcrChar();
00083
00090     OcrChar(const std::vector<OcrCharVariant>& ocr_char_variants,
00091            bool is_highlighted, bool is_corrected,
00092            const Rectangle& ocr_char_rect = {});
00093
00095     ~OcrChar();
00096
00098     const std::vector<OcrCharVariant>& GetOcrCharVariants() const;
00099
00101     bool IsHighlighted() const;
00103     bool IsCorrected() const;
00104
00110     uint16_t GetUtf16Character() const throw(std::exception);
00111
00118     std::string GetUtf8Character() const throw(std::exception);
00119
00125     const Rectangle& GetRectangle() const;
00126
00127 private:
00128     std::vector<OcrCharVariant> ocr_char_variants_;
00129     bool is_highlighted_;
00130     bool is_corrected_;
00131     Rectangle rect_;
00132 };
00133
00137 class SMARTID_DLL_EXPORT OcrString {
00138 public:
00140     OcrString();
00142     OcrString(const std::vector<OcrChar>& ocr_chars);
00146     OcrString(const std::string& utf8_string);
00148     ~OcrString();
00149
00151     const std::vector<OcrChar>& GetOcrChars() const;
00152
00154     std::string GetUtf8String() const;
00155
00157     std::vector<uint16_t> GetUtf16String() const;
00158
00159 private:
00160     std::vector<OcrChar> ocr_chars_;
00161 };
00162
00167 class SMARTID_DLL_EXPORT StringField {

```

```

00168 public:
00172     StringField();
00173
00184     StringField(const std::string& name, const OcrString& value,
00185                 bool is_accepted, double confidence,
00186                 const std::map<std::string, std::string>& attributes = {}) throw(std::exception);
00187
00199     StringField(const std::string& name, const std::string& value,
00200                 bool is_accepted, double confidence,
00201                 const std::map<std::string, std::string>& attributes = {}) throw(std::exception);
00202
00206     ~StringField();
00207
00209     const std::string& GetName() const;
00211     const OcrString& GetValue() const;
00213     std::string GetUtf8Value() const;
00215     bool IsAccepted() const;
00218     double GetConfidence() const;
00219
00221     std::vector<std::string> GetAttributeNames() const;
00222
00224     const std::map<std::string, std::string> &GetAttributes() const;
00225
00231     bool HasAttribute(const std::string &attribute_name) const;
00232
00238     const std::string &GetAttribute(const std::string &attribute_name) const
00239         throw(std::exception);
00240
00241 private:
00242     std::string name_;
00243     OcrString value_;
00244
00246     bool is_accepted_;
00248     double confidence_;
00249
00250     std::map<std::string, std::string> attributes_;
00251 };
00252
00256 class SMARTID_DLL_EXPORT ImageField {
00257 public:
00261     ImageField();
00262
00274     ImageField(const std::string& name, const Image& value, bool is_accepted,
00275                 double confidence) throw(std::exception);
00276
00278     ~ImageField();
00279
00281     const std::string& GetName() const;
00283     const Image& GetValue() const;
00285     bool IsAccepted() const;
00287     double GetConfidence() const;
00288
00289 private:
00290     std::string name_;
00291     Image value_;
00292
00293     bool is_accepted_;
00294     double confidence_;
00295 };
00296
00300 class SMARTID_DLL_EXPORT MatchResult {
00301 public:
00305     MatchResult();
00306
00313     MatchResult(const std::string& tpl_type,
00314                 const Quadrangle& quadrangle,
00315                 bool accepted = false,
00316                 double confidence = 0.0,
00317                 int standard_width = 0,
00318                 int standard_height = 0);
00319
00323     ~MatchResult();
00324
00326     const std::string& GetTemplateType() const;
00328     const Quadrangle& GetQuadrangle() const;
00330     int GetStandardWidth() const;
00332     int GetStandardHeight() const;
00334     bool GetAccepted() const;
00336     double GetConfidence() const;
00337
00338 private:
00339     std::string template_type_;
00340     Quadrangle quadrangle_;
00341     int standard_width_;
00342     int standard_height_;
00343     bool accepted_;
00344     double confidence_;

```

```

00345 };
00346
00351 class SMARTID_DLL_EXPORT ForensicField {
00352 public:
00353     ForensicField();
00354
00355     ForensicField(const std::string& name,
00356                  const std::string& value,
00357                  bool is_accepted,
00358                  double confidence,
00359                  const std::map<std::string, std::string>& attributes = {}) throw(std::exception);
00360
00361     ~ForensicField();
00362
00363     const std::string& GetName() const;
00364     const std::string& GetValue() const;
00365     bool IsAccepted() const;
00366     double GetConfidence() const;
00367
00368     std::vector<std::string> GetAttributeNames() const;
00369
00370     const std::map<std::string, std::string> &GetAttributes() const;
00371
00372     bool HasAttribute(const std::string &attribute_name) const;
00373
00374     const std::string &GetAttribute(const std::string &attribute_name) const
00375         throw(std::exception);
00376
00377 private:
00378     std::string name_;
00379     std::string value_;
00380
00381     bool is_accepted_;
00382     double confidence_;
00383
00384     std::map<std::string, std::string> attributes_;
00385 };
00386
00387 class SMARTID_DLL_EXPORT SegmentationResult {
00388 public:
00389     SegmentationResult();
00390
00391     SegmentationResult(
00392         const std::map<std::string, Quadrangle>& raw_fields_quadrangles,
00393         const std::map<std::string, Quadrangle>& raw_fields_template_quadrangles,
00394         bool accepted = false);
00395
00396     ~SegmentationResult();
00397
00398     std::vector<std::string> GetRawFieldsNames() const;
00399
00400     bool HasRawFieldQuadrangle(const std::string &raw_field_name) const;
00401
00402     const Quadrangle& GetRawFieldQuadrangle(const std::string &raw_field_name) const throw (
00403         std::exception);
00404
00405     const std::map<std::string, Quadrangle>& GetRawFieldQuadrangles() const;
00406
00407     const Quadrangle& GetRawFieldTemplateQuadrangle(const std::string &raw_field_name) const throw (
00408         std::exception);
00409
00410     const std::map<std::string, Quadrangle>& GetRawFieldTemplateQuadrangles() const;
00411
00412     bool GetAccepted() const;
00413
00414 private:
00415     std::map<std::string, Quadrangle> raw_field_quadrangles_;
00416     std::map<std::string, Quadrangle> raw_field_template_quadrangles_;
00417     bool accepted_;
00418 };
00419
00420 class SMARTID_DLL_EXPORT RecognitionResult {
00421 public:
00422     RecognitionResult();
00423
00424     RecognitionResult(const std::map<std::string, StringField>& string_fields,
00425                      const std::map<std::string, ImageField>& image_fields,
00426                      const std::map<std::string, ForensicField>& forensic_fields,
00427                      const std::map<std::string, StringField>& raw_string_fields,
00428                      const std::map<std::string, ImageField>& raw_image_fields,
00429                      const std::string& document_type,
00430                      const std::vector<MatchResult>& match_results,
00431                      const std::vector<SegmentationResult>& segmentation_results,
00432                      bool is_terminal);
00433
00434     ~RecognitionResult();
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508

```

```
00510     std::vector<std::string> GetStringFieldNames() const;
00512     bool HasStringField(const std::string& name) const;
00513
00520     const StringField& GetStringField(
00521         const std::string& name) const throw(std::exception);
00522
00527     const std::map<std::string, StringField>& GetStringFields() const;
00528
00533     std::map<std::string, StringField>& GetStringFields();
00534
00539     void SetStringFields(const std::map<std::string, StringField>& string_fields);
00540
00542
00544     std::vector<std::string> GetImageFieldNames() const;
00546     bool HasImageField(const std::string& name) const;
00547
00554     const ImageField& GetImageField(
00555         const std::string& name) const throw(std::exception);
00556
00561     const std::map<std::string, ImageField>& GetImageFields() const;
00562
00567     std::map<std::string, ImageField>& GetImageFields();
00568
00573     void SetImageFields(const std::map<std::string, ImageField>& image_fields);
00574
00576
00578     std::vector<std::string> GetForensicFieldNames() const;
00580     bool HasForensicField(const std::string& name) const;
00581
00588     const ForensicField& GetForensicField(
00589         const std::string& name) const throw(std::exception);
00590
00595     const std::map<std::string, ForensicField>& GetForensicFields() const;
00596
00601     std::map<std::string, ForensicField>& GetForensicFields();
00602
00607     void SetForensicFields(
00608         const std::map<std::string, ForensicField>& forensic_fields);
00609
00611
00613     std::vector<std::string> GetRawStringFieldNames() const;
00615     bool HasRawStringField(const std::string& name) const;
00616
00623     const StringField& GetRawStringField(
00624         const std::string& name) const throw(std::exception);
00625
00630     const std::map<std::string, StringField>& GetRawStringFields() const;
00631
00636     std::map<std::string, StringField>& GetRawStringFields();
00637
00642     void SetRawStringFields(const std::map<std::string, StringField>& raw_string_fields);
00643
00645
00647     std::vector<std::string> GetRawImageFieldNames() const;
00649     bool HasRawImageField(const std::string& name) const;
00650
00657     const ImageField& GetRawImageField(
00658         const std::string& name) const throw(std::exception);
00659
00664     const std::map<std::string, ImageField>& GetRawImageFields() const;
00665
00670     std::map<std::string, ImageField>& GetRawImageFields();
00671
00676     void SetRawImageFields(const std::map<std::string, ImageField>& raw_image_fields);
00677
00679
00682     const std::string& GetDocumentType() const;
00683
00685     void SetDocumentType(const std::string& doctype);
00686
00688
00691     const std::vector<MatchResult>& GetMatchResults() const;
00693     void SetMatchResults(const std::vector<MatchResult>& match_results);
00694
00696
00699     const std::vector<SegmentationResult>& GetSegmentationResults() const;
00701     void SetSegmentationResults(const std::vector<SegmentationResult>& segmentation_results);
00702
00704
00709     bool IsTerminal() const;
00711     void SetIsTerminal(bool is_terminal);
00712
00716     const std::string& GetJpegCompression() const;
00717
00719     void SetJpegCompression(const std::string& jpeg_compression);
00720
00721 private:
```



```

00722     std::map<std::string, StringField> string_fields_;
00723     std::map<std::string, ImageField> image_fields_;
00724     std::map<std::string, ForensicField> forensic_fields_;
00725
00726     std::map<std::string, StringField> raw_string_fields_;
00727     std::map<std::string, ImageField> raw_image_fields_;
00728     std::string document_type_;
00729     std::vector<MatchResult> match_results_;
00730     std::vector<SegmentationResult> segmentation_results_;
00731     bool is_terminal_;
00732     std::string jpeg_compression_;
00733 };
00734
00740 class SMARTID_DLL_EXPORT ProcessingFeedback {
00741 public:
00742     ProcessingFeedback();
00743
00744     ProcessingFeedback(const std::map<std::string, Quadrangle> &quadrangles);
00745
00746     ~ProcessingFeedback();
00747
00748     const std::map<std::string, Quadrangle>& GetQuadrangles() const;
00749 private:
00750     std::map<std::string, Quadrangle> quadrangles_;
00751 };
00752
00765 class SMARTID_DLL_EXPORT ResultReporterInterface {
00766 public:
00767
00768     virtual void SnapshotRejected();
00769
00770     virtual void FeedbackReceived(const ProcessingFeedback& processing_feedback);
00771
00772     virtual void DocumentMatched(const std::vector<MatchResult>& match_results);
00773
00774     virtual void DocumentSegmented(const std::vector<SegmentationResult>& segmentation_results);
00775
00776     virtual void SnapshotProcessed(const RecognitionResult& recog_result) = 0;
00777
00778     virtual void SessionEnded();
00779
00780     virtual ~ResultReporterInterface();
00781 };
00782
00815 class SMARTID_DLL_EXPORT IntegratedFieldState {
00816 public:
00817     explicit IntegratedFieldState(bool is_terminal = false);
00818
00819     bool IsTerminal() const;
00820     void SetIsTerminal(bool is_terminal);
00821 private:
00822     bool is_terminal_;
00823 };
00824
00836 class SMARTID_DLL_EXPORT SessionState {
00837 public:
00838     virtual ~SessionState();
00839
00840     std::vector<std::string> GetIntegratedFieldStateNames() const;
00841     bool HasIntegratedFieldState(const std::string& name) const;
00842
00843     const IntegratedFieldState& GetStringFieldState(
00844         const std::string& name) const throw(std::exception);
00845
00846     const std::map<std::string, IntegratedFieldState>& GetIntegratedFieldStates() const;
00847
00848     std::map<std::string, IntegratedFieldState>& GetIntegratedFieldStates();
00849
00850     void SetIntegratedFieldStates(const std::map<std::string, IntegratedFieldState>& integrated_field_states)
00851 ;
00852
00853     int GetSnapshotsProcessed() const;
00854
00855 protected:
00856     std::map<std::string, IntegratedFieldState> integrated_field_states_;
00857     int snapshots_processed_;
00858
00859     SessionState(int snapshots_processed);
00860 };
00861
00862 } // namespace se::smartid
00863
00864 #if defined _MSC_VER
00865 #pragma warning(pop)
00866 #endif

```

```
00887
00888 #endif // SMARTID_ENGINE_SMARTID_RESULT_H_INCLUDED
```


Index

- AddEnabledDocumentTypes
 - se::smartid::SessionSettings, [42](#)
- Clone
 - se::smartid::SessionSettings, [42](#)
- CopyBase64ToBuffer
 - se::smartid::Image, [6](#)
- CopyToBuffer
 - se::smartid::Image, [7](#)
- CreateSessionSettings
 - se::smartid::RecognitionEngine, [22](#)
- Crop
 - se::smartid::Image, [7](#)
- DisableField
 - se::smartid::SessionSettings, [42](#)
- DisableForensicField
 - se::smartid::SessionSettings, [42](#)
- DocumentMatched
 - se::smartid::ResultReporterInterface, [38](#)
- DocumentSegmented
 - se::smartid::ResultReporterInterface, [38](#)
- EnableField
 - se::smartid::SessionSettings, [43](#)
- EnableForensicField
 - se::smartid::SessionSettings, [43](#)
- EstimateFocusScore
 - se::smartid::Image, [8](#)
- FeedbackReceived
 - se::smartid::ResultReporterInterface, [38](#)
- ForensicField
 - se::smartid::ForensicField, [2](#)
- GetAttribute
 - se::smartid::ForensicField, [3](#)
 - se::smartid::StringField, [50](#)
- GetAvailableModes
 - se::smartid::SessionSettings, [43](#)
- GetBase64String
 - se::smartid::Image, [8](#)
- GetBoundingBox
 - se::smartid::Quadrangle, [19](#)
- GetChannels
 - se::smartid::Image, [8](#)
- GetCurrentMode
 - se::smartid::SessionSettings, [43](#)
- GetEnabledDocumentTypes
 - se::smartid::SessionSettings, [43](#)
- GetForensicField
 - se::smartid::RecognitionResult, [26](#)
- GetForensicFields
 - se::smartid::RecognitionResult, [26](#)
- GetHeight
 - se::smartid::Image, [8](#)
- GetImageField
 - se::smartid::RecognitionResult, [26](#)
- GetImageFields
 - se::smartid::RecognitionResult, [26](#), [27](#)
- GetIntegratedFieldStates
 - se::smartid::SessionState, [48](#)
- GetOption
 - se::smartid::SessionSettings, [43](#)
- GetOptionNames
 - se::smartid::SessionSettings, [44](#)
- GetOptions
 - se::smartid::SessionSettings, [44](#)
- GetPoint
 - se::smartid::Quadrangle, [20](#)
- GetQuadrangles
 - se::smartid::ProcessingFeedback, [18](#)
- GetRawFieldQuadrangle
 - se::smartid::SegmentationResult, [39](#)
- GetRawFieldTemplateQuadrangle
 - se::smartid::SegmentationResult, [40](#)
- GetRawImageField
 - se::smartid::RecognitionResult, [27](#)
- GetRawImageFields
 - se::smartid::RecognitionResult, [27](#)
- GetRawStringField
 - se::smartid::RecognitionResult, [27](#)
- GetRawStringFields
 - se::smartid::RecognitionResult, [28](#)
- GetRequiredBase64BufferLength
 - se::smartid::Image, [8](#)
- GetRequiredBufferLength
 - se::smartid::Image, [9](#)
- GetSessionState
 - se::smartid::RecognitionSession, [31](#)
- GetStride
 - se::smartid::Image, [9](#)
- GetStringField
 - se::smartid::RecognitionResult, [28](#)
- GetStringFieldState
 - se::smartid::SessionState, [48](#)
- GetStringFields
 - se::smartid::RecognitionResult, [28](#)
- GetSupportedDocumentTypes
 - se::smartid::SessionSettings, [44](#)
- GetSupportedFieldNames
 - se::smartid::SessionSettings, [44](#)
- GetSupportedForensicFieldNames
 - se::smartid::SessionSettings, [45](#)
- GetUtf16Character
 - se::smartid::OcrChar, [14](#)
- GetUtf8Character
 - se::smartid::OcrChar, [15](#)
- GetVersion
 - se::smartid::RecognitionEngine, [23](#)
- GetWidth

- se::smartid::Image, 9
- HasAttribute
 - se::smartid::ForensicField, 3
 - se::smartid::StringField, 51
- HasOption
 - se::smartid::SessionSettings, 45
- Image
 - se::smartid::Image, 5, 6
- ImageField
 - se::smartid::ImageField, 11
- IsMemoryOwner
 - se::smartid::Image, 9
- MaskImageRegionQuadrangle
 - se::smartid::Image, 9
- MaskImageRegionRectangle
 - se::smartid::Image, 9
- MatchResult
 - se::smartid::MatchResult, 13
- OcrChar
 - se::smartid::OcrChar, 14
- OcrCharVariant
 - se::smartid::OcrCharVariant, 15, 16
- operator=
 - se::smartid::Image, 10
- operator[]
 - se::smartid::Quadrangle, 20
- Point
 - se::smartid::Point, 17
- ProcessImage
 - se::smartid::RecognitionSession, 31
- ProcessImageData
 - se::smartid::RecognitionSession, 32
- ProcessImageDataBase64
 - se::smartid::RecognitionSession, 33
- ProcessImageFile
 - se::smartid::RecognitionSession, 33, 34
- ProcessSnapshot
 - se::smartid::RecognitionSession, 34, 35
- ProcessYUVSnapshot
 - se::smartid::RecognitionSession, 35, 36
- Quadrangle
 - se::smartid::Quadrangle, 19
- RecognitionEngine
 - se::smartid::RecognitionEngine, 22
- Rectangle
 - se::smartid::Rectangle, 37
- RemoveEnabledDocumentTypes
 - se::smartid::SessionSettings, 45
- RemoveOption
 - se::smartid::SessionSettings, 46
- Resize
 - se::smartid::Image, 10
- Save
 - se::smartid::Image, 10
- se::smartid::ForensicField, 1
 - ForensicField, 2
 - GetAttribute, 3
 - HasAttribute, 3
- se::smartid::Image, 3
 - CopyBase64ToBuffer, 6
 - CopyToBuffer, 7
 - Crop, 7
 - EstimateFocusScore, 8
 - GetBase64String, 8
 - GetChannels, 8
 - GetHeight, 8
 - GetRequiredBase64BufferLength, 8
 - GetRequiredBufferLength, 9
 - GetStride, 9
 - GetWidth, 9
 - Image, 5, 6
 - IsMemoryOwner, 9
 - MaskImageRegionQuadrangle, 9
 - MaskImageRegionRectangle, 9
 - operator=, 10
 - Resize, 10
 - Save, 10
- se::smartid::ImageField, 10
 - ImageField, 11
- se::smartid::IntegratedFieldState, 12
- se::smartid::MatchResult, 12
 - MatchResult, 13
- se::smartid::OcrChar, 13
 - GetUtf16Character, 14
 - GetUtf8Character, 15
 - OcrChar, 14
- se::smartid::OcrCharVariant, 15
 - OcrCharVariant, 15, 16
- se::smartid::OcrString, 16
- se::smartid::Point, 17
 - Point, 17
- se::smartid::ProcessingFeedback, 18
 - GetQuadrangles, 18
- se::smartid::Quadrangle, 18
 - GetBoundingRectangle, 19
 - GetPoint, 20
 - operator[], 20
 - Quadrangle, 19
 - SetPoint, 20
- se::smartid::RecognitionEngine, 21
 - CreateSessionSettings, 22
 - GetVersion, 23
 - RecognitionEngine, 22
 - SpawnSession, 23
- se::smartid::RecognitionResult, 23
 - GetForensicField, 26
 - GetForensicFields, 26
 - GetImageField, 26
 - GetImageFields, 26, 27
 - GetRawImageField, 27

- GetRawImageFields, [27](#)
- GetRawStringField, [27](#)
- GetRawStringFields, [28](#)
- GetStringField, [28](#)
- GetStringFields, [28](#)
- SetForensicFields, [28](#)
- SetImageFields, [29](#)
- SetRawImageFields, [29](#)
- SetRawStringFields, [29](#)
- SetStringFields, [29](#)
- se::smartid::RecognitionSession, [29](#)
 - GetSessionState, [31](#)
 - ProcessImage, [31](#)
 - ProcessImageData, [32](#)
 - ProcessImageDataBase64, [33](#)
 - ProcessImageFile, [33](#), [34](#)
 - ProcessSnapshot, [34](#), [35](#)
 - ProcessYUVSnapshot, [35](#), [36](#)
- se::smartid::Rectangle, [36](#)
 - Rectangle, [37](#)
- se::smartid::ResultReporterInterface, [37](#)
 - DocumentMatched, [38](#)
 - DocumentSegmented, [38](#)
 - FeedbackReceived, [38](#)
 - SnapshotProcessed, [38](#)
- se::smartid::SegmentationResult, [38](#)
 - GetRawFieldQuadrangle, [39](#)
 - GetRawFieldTemplateQuadrangle, [40](#)
- se::smartid::SessionSettings, [40](#)
 - AddEnabledDocumentTypes, [42](#)
 - Clone, [42](#)
 - DisableField, [42](#)
 - DisableForensicField, [42](#)
 - EnableField, [43](#)
 - EnableForensicField, [43](#)
 - GetAvailableModes, [43](#)
 - GetCurrentMode, [43](#)
 - GetEnabledDocumentTypes, [43](#)
 - GetOption, [43](#)
 - GetOptionNames, [44](#)
 - GetOptions, [44](#)
 - GetSupportedDocumentTypes, [44](#)
 - GetSupportedFieldNames, [44](#)
 - GetSupportedForensicFieldNames, [45](#)
 - HasOption, [45](#)
 - RemoveEnabledDocumentTypes, [45](#)
 - RemoveOption, [46](#)
 - SetCurrentMode, [46](#)
 - SetEnabledDocumentTypes, [46](#)
 - SetEnabledFields, [46](#)
 - SetEnabledForensicFields, [46](#)
 - SetOption, [47](#)
- se::smartid::SessionState, [47](#)
 - GetIntegratedFieldStates, [48](#)
 - GetStringFieldState, [48](#)
 - SetIntegratedFieldStates, [48](#)
- se::smartid::StringField, [49](#)
 - GetAttribute, [50](#)
 - HasAttribute, [51](#)
 - StringField, [50](#)
- SetCurrentMode
 - se::smartid::SessionSettings, [46](#)
- SetEnabledDocumentTypes
 - se::smartid::SessionSettings, [46](#)
- SetEnabledFields
 - se::smartid::SessionSettings, [46](#)
- SetEnabledForensicFields
 - se::smartid::SessionSettings, [46](#)
- SetForensicFields
 - se::smartid::RecognitionResult, [28](#)
- SetImageFields
 - se::smartid::RecognitionResult, [29](#)
- SetIntegratedFieldStates
 - se::smartid::SessionState, [48](#)
- SetOption
 - se::smartid::SessionSettings, [47](#)
- SetPoint
 - se::smartid::Quadrangle, [20](#)
- SetRawImageFields
 - se::smartid::RecognitionResult, [29](#)
- SetRawStringFields
 - se::smartid::RecognitionResult, [29](#)
- SetStringFields
 - se::smartid::RecognitionResult, [29](#)
- smartid_common.h, [51](#)
- smartid_engine.h, [54](#)
- smartid_result.h, [57](#)
- SnapshotProcessed
 - se::smartid::ResultReporterInterface, [38](#)
- SpawnSession
 - se::smartid::RecognitionEngine, [23](#)
- StringField
 - se::smartid::StringField, [50](#)