ABBYY®

ABBYY® FineReader® Engine 10

USER'S GUIDE

Contents

Introducing ABBYY FineReader Engine 10

Welcome to ABBYY FineReader Engine 10!

Accuracy and speed, power and simplicity – are you expecting all these qualities from OCR SDK, but they seem to be contradictive? No more!

With new ABBYY FineReader Engine 10 you receive outstanding level of OCR quality and usability:

- Optimally balanced profiles with fine-tuned parameters for your particular tasks
- Worldwide recognized accuracy of technologies
- Amazingly improved speed
- Absolute world record – 198 recognition languages, including Chinese, Japanese, Korean and Arabic
- SDK Developer's Guide (Help), currently recognized for its unbeatable comprehensibility and usefulness, now becomes even better with its improved appearance and revised content

ABBYY FineReader Engine 10 – powerful and convenient OCR technology. Just try and appreciate!

Key Features

- Extreme Recognition Speed
- Improved Recognition Accuracy
- Powerful and Simple API
- Unique Layout Reconstruction Abilities
- Improved PDF Export
- Unrivaled Document Image and Photo Processing
- Flexible and Strong Protection System

Basic Usage Scenarios

Rich experience in use of ABBYY SDK in hundreds of products all over the world allows us to extract the most widespread scenarios of OCR SDK usage:

Document Conversion Scenarios	Scenarios for Data Capture	General Scenarios
• Document Conversion for Content Reuse • Document Archiving • Book Archiving	• Text Extraction • Field-Level Recognition • Barcode Recognition	• Image Preprocessing • Scanning

How to Use this Help

In this Developer's Help you can find all the necessary information about ABBYY FineReader Engine 10.

Guided Tour
See the Guided Tour section to learn about how to use ABBYY FineReader Engine:

- **Basic Usage Scenarios Implementation**
  You can find here the way to use ABBYY FineReader Engine for your task.
- **Advanced Techniques**
  The information for advanced users.

- **Best Practices**
  Offers you some advice on how to prepare images for recognition.

- **Samples**
  Short description of available samples. The detailed description of the samples you can find in the Code Samples Library provided with this distribution pack.

API Reference
The complete reference of the FineReader Engine API.

Licensing
Important information about ABBYY FineReader Engine licenses and activation.

Distribution
Information about distribution of applications which use the ABBYY FineReader Engine library.

Specifications
The list of supported image formats, predefined languages, text types, barcode types, export formats, system requirements, and:

- What's New in ABBYY FineReader Engine 10

- Compatibility with ABBYY FineReader Engine 9.0

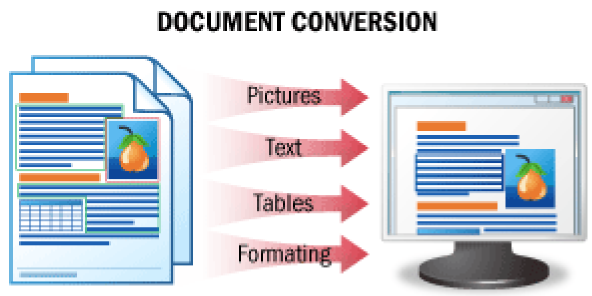
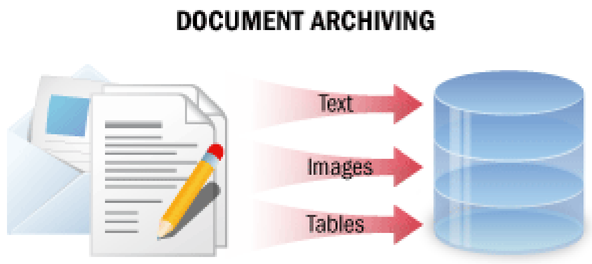
Frequently Asked Questions
The extracts of the most important information.

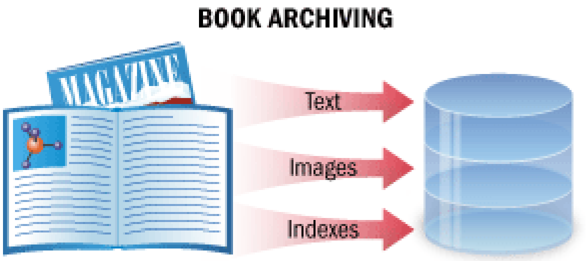
Contact ABBYY
You can find here the contacts of ABBYY offices and Technical support service.

You can visit our website at www.abbyy.com for the most up-to-date information about ABBYY FineReader Engine.

Basic Usage Scenarios Overview

This section describes the most common scenarios in which ABBYY FineReader Engine may be used. We recommend to start your work with ABBYY FineReader Engine with selecting the appropriate scenario. After you found the appropriate scenario, you can find a detailed description of the scenario, implementation advice, and suggestions on optimizing the code for specific tasks in the Guided Tour section.

Document Conversion	
 DOCUMENT CONVERSION Pictures Text Tables Formating	The result of this scenario is an **editable** version of a document. In this scenario, document images are recognized, retaining all the original formatting intact and the data are saved to an editable file format. As a result, you get editable versions of your documents, which can be easily checked for errors and modified. See for details **Document Conversion**.
	Under this processing scenario, paper documents are converted into **not editable** electronic copies containing all document information in searchable format. As a result of such processing, the resulting copies may be easily found in the electronic archive using full-text search, document text segments may be copied and the document may be sent by email or printed out. See for details **Document Archiving**.

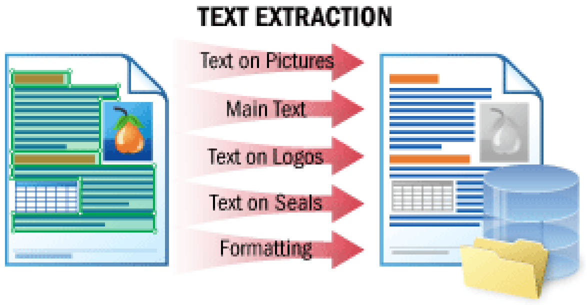


This scenario is used for processing books, magazines, newspapers to create an **electronic library**; for instance, when digitizing paper book collections for purposes of facilitating and expanding access to them and for their preservation.

Under this scenario, books, magazines, newspapers are converted into not editable electronic copies containing all information from the source in searchable format.

See for details **Book Archiving**.

Data Capture

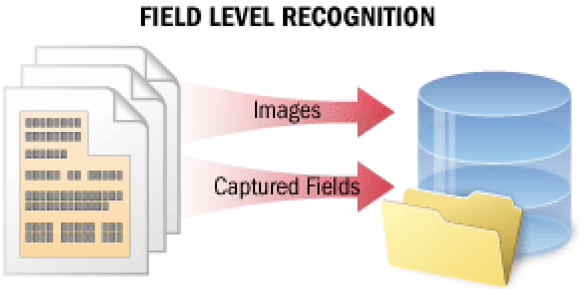


This scenario is used to recognize the entire document text in order to prepare the document for search and extraction of useful data.

Such a scenario may serve as a basis for implementing more complex scenarios to extract vital data from documents, especially for automated input of paper document data into information systems and databases as well as for automated classification and indexation of documents in document management systems (e.g., inputting invoices into accounting software, inputting questionnaires into the CRM system).

This scenario enables extraction of the main text of the document, which contains all necessary information about the document. When using this scenario, all text data including texts on logos, seals and elements other than the main text, are extracted from the text.

See for details **Text Extraction**.

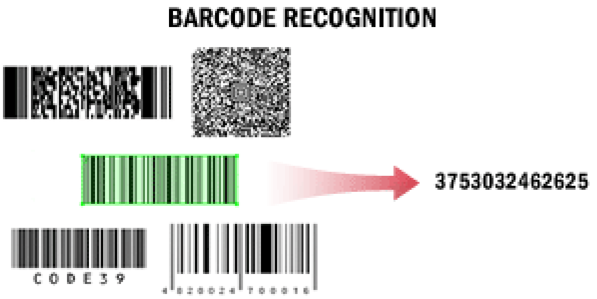


In the case of field-level recognition, short text fragments are recognized in order to capture data from certain fields. Recognition quality is crucial in this scenario.

This scenario may also be used as part of more complex scenarios where meaningful data are to be extracted from documents (for example, to capture data from paper documents into information systems and databases or to automatically classify and index documents in Document Management Systems).

In this scenario, the system recognizes either several lines of text in only some of the fields or the entire text on a small image. The system computes a certainty rating for each recognized character. The certainty ratings can then be used when checking the recognition results. Additionally, the system may store multiple recognition variants for words and characters in the text, which may then be used in voting algorithms to improve the quality of recognition.

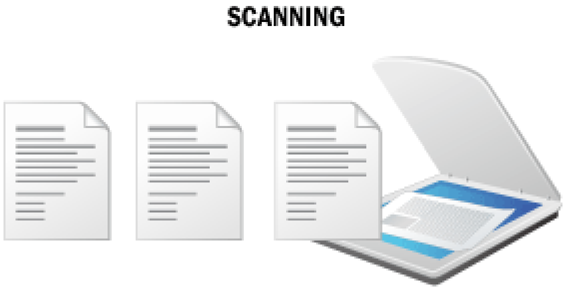
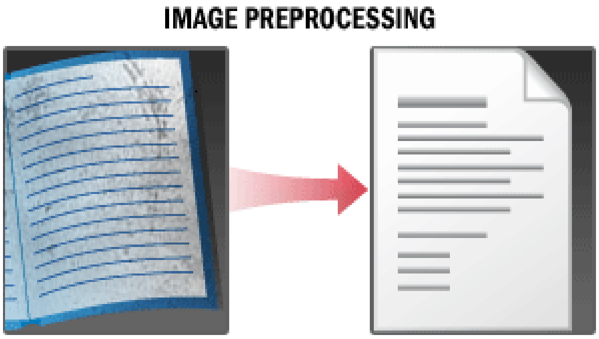
See for details **Field-Level Recognition**.



In this scenario, ABBYY FineReader Engine is used to read barcodes. Barcodes may need to be read, for example, for purposes of automatic document separation, for processing documents by a Document Management System, or for indexing and classifying documents.

This scenario may be used as part of other scenarios. For example, documents scanned with high-speed production scanners may be separated by means of barcodes, or documents prepared for long-term storage may be placed into archiving Document Management Systems based on the values of their barcodes.

When extracting barcodes from texts, the system may detect all barcodes or only barcodes of a certain type with a certain value. The system may get the value of a barcode and calculate its check

	sum. Recognized barcode values can be saved into formats most convenient for further processing, for example into TXT. See for details **Barcode Recognition**.
General	
	In this scenario, ABBYY FineReader Engine is used on a "scanning computer," which scans images and saves them as files. This scenario may be used as part of other scenarios in the preliminary stage of document processing, i.e. for obtaining electronic versions of documents for further processing. Usage examples include scanning documents for archiving purposes, getting editable versions of documents, and extracting meaningful data from documents. Paper documents are scanned and the images are saved in an electronic format, producing high-quality electronic versions of your printed documents. See for details **Scanning**.
	This scenario can be used to prepare images for further processing or to improve their visual quality (e.g. after scanning or prior to recognition). This scenario may be used as part of other scenarios in the first stage of document processing, i.e. to prepare documents for recognition. Usage examples include creating uneditable document copies for archiving, getting editable versions of documents, and extracting meaningful data from documents. See for details **Image Preprocessing**.

See also

Basic Usage Scenarios

Key Features

Extreme Recognition Speed

Tuned Fast Mode	Perfectly adjusted Fast mode provides absolutely amazing results – about 90-110% speed increasing[*] with more than 98,5% accuracy for most of European languages

[*] comparing to Fast mode of ABBYY FineReader Engine 9.0 (First release, 21 October 2008)

Note: ABBYY unrivaled multicore support architecture ensures close to linear performance growth with increasing number of cores for multipage documents. For 2 CPU cores it works almost 2 times faster, for 4 cores – almost 4 times!

Improved Recognition Accuracy

Accuracy tuning for European languages	ABBYY OCR technologies are worldwide famed for the recognition accuracy but now they show the really outstanding results! The accuracy increased 50%[**] on average for European languages and valued more than 99.3% of correctly recognized characters.
Improved classifier for CJK	The recognition accuracy for Chinese, Japanese and Korean languages went up 30-60% due to improved Asian characters classifier. Now ABBYY OCR SDK provides the top level of accuracy among international multilanguage OCR technologies.
New mode for low resolution scans	The special new recognition mode for low quality documents – old faxes, low resolution scans provides 20% higher accuracy for such documents than standard Normal mode.

[**] comparing to Normal mode of ABBYY FineReader Engine 9.0 (First release, 21 October 2008)

Powerful and Simple API

Special profiles for popular usage scenarios	A lot of developers mentioned that ABBYY FineReader Engine API is the most powerful and full-functional among OCR SDKs. Now it becomes simpler with new profiles for the most popular recognition tasks. They are predefined with optimal parameters for easy start and guaranteed OCR quality without long-time manual tuning. Manual parameters setting is also available for any custom solutions.
Document structure API	ABBYY FineReader Engine 10 provides unique feature-set for access to document structure elements like headings, chapters, page numbers, footnotes, headers, footers and so on.

Unique Layout Reconstruction Abilities

Document structure detection	ABBYY FineReader Engine 10 automatically detects headings in recognized document, determines their level in document structure, defines their text styles and reconstructs the whole structure as Document Map of resulting document.
TOC reconstruction	In final document the Table of Contents appears as a set of links to the headings. After document editing TOC could be updated automatically as a single object to add new headings and revise page numbers.
Charts and diagrams detection	Automatic charts and diagrams detection feature was improved in 10th version of ABBYY OCR SDK. Now it is possible to choose if recognize text on chart or stay it in origin image form.
Picture and table captions processing	ABBYY FineReader Engine 10 automatically detects picture's and table's captions and exports them to final document as a single frame including the picture and its title.
Document styles defining	ABBYY FineReader Engine 10 analyzes text font type, size, and its placement and defects the corresponding font style for every type of text. So for the headings of each level there are special styles, for ordinary text, for TOC and for picture captions there are also special styles.
"Glossy magazine" processing model	New ABBYY SDK can reconstruct complicated layouts consisted of many pictures and text blocks on a page or including very large pictures for the whole page.

Improved PDF Export

Superior quality-size ratio for PDF files	New PDF export together with improved MRC (Mixed Raster Content) compression technology allows achieving higher quality and less size of PDF documents.
PDF export profiles	There are more than 40 parameters for PDF export tuning. ABBYY FineReader Engine 10 provides predefined profiles with optimal values for popular export variants: • MaxQuality • Balanced • MinSize • MaxSpeed

Unrivaled Document Image and Photo Processing

New features of Camera OCR™	Camera OCR technology – the set of document photo adjustment features for better recognition results was improved with new unique features: • Automatic correction of 3D perspective distortions • Blurred image correction • ISO noise reduction
New binarization	Previous OCR SDK version provided very high quality of binarization, but in some the most difficult cases it could commit errors and losses of information. New binarization technology ensures the whole text retention and prevents information losses even in difficult cases.

Flexible and Strong Protection System

Improved protection	Protection system of ABBYY FineReader Engine 10 provides:

	• Delegate and control SDK usage rights in local network • Count and control the numbers of recognized characters, pages, usage time and computing power • Track and control SDK usage on terminal servers and virtual machines

Full functionality

- Document Scanning and Image Import
- Image Preprocessing
- Document Analysis
- OCR and Other Recognition Technologies
- PDF Conversion
- Advanced Development Tools
- Receiving and Exporting Recognized Text
- Multi-CPU Recognition Architecture

Document Scanning and Image Import

ABBYY FineReader Engine can receive images from three types of sources: document scanning, opening from files, or directly from memory.

Document Scanning APIs

- TWAIN interface (including ADF support and manual input feeding)
- FineReader document scanning UI

With its powerful document scanning software tools, ABBYY FineReader Engine 10 enables flexible management of scanning parameters, such as: brightness, colority, resolution, image size, duplex scanning, pause between pages setup and more. For OCR purposes, the best resolutions lie in the range of 200-400 dpi. The choice of resolution depends on the quality of the paper original, the size of the font and other factors. For more details, please see the description of the Scanning scenario.

Image file formats

The OCR SDK supports the majority of image formats, including multi-page TIFF and JPEG 2000 (part1), and works with black-and-white, grayscale and color images. It also opens PDF files by converting them into images with Adobe® PDF Library Technology.

- BMP
- DCX
- DjVu
- GIF
- JBIG2
- JPEG
- JPEG 2000
- PCX
- PNG
- PDF
- TIFF and multi-page TIFF
- WDP

See more in Supported Image Formats.

Memory image formats

- Raw
- Bitmap (HBITMAP)
- DIB

Additional features for PDF files

- Extracting text layer from PDF
- Image only PDF input
- Vectorized PDF
- Password protected PDF

See also

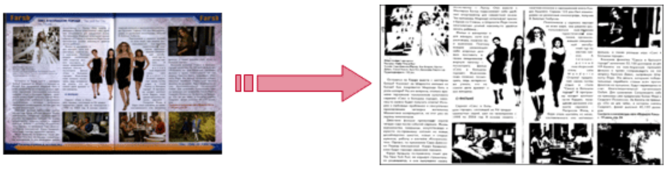
Key Features
Image Preprocessing
Basic Usage Scenarios Implementation: Scanning

Image Preprocessing

Why improve images?

The task of improving image quality is two-fold.

On the one hand, we need to improve the quality of the images to make them more suitable for OCR.



On the other hand, we need to improve the appearance of the images, which is necessary, for example, when we store document images in archives.

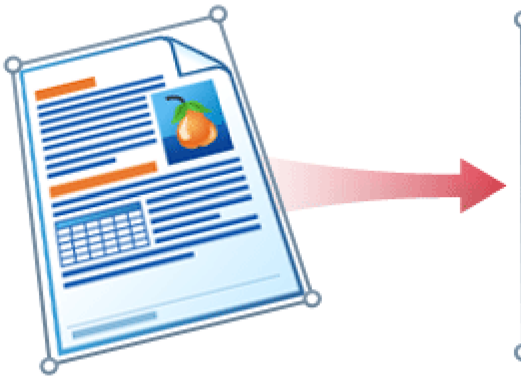
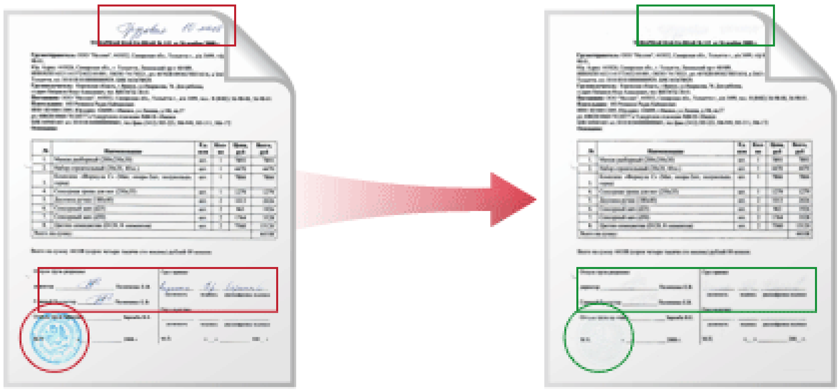
As ABBYY technologies are focused on document analysis and recognition, the system includes a set of powerful image preprocessing technologies: adaptive binarization, correction of distortions, straightening text lines, splitting facing pages, and others.

No third-party tools are needed to get accurate OCR results. ABBYY offers a complete set of preprocessing technologies geared towards OCR.

Image Preprocessing

Upon receiving images, ABBYY FineReader Engine performs a range of image preprocessing functions to improve the quality of document images for further recognition or archiving:

Image preprocessing (straightening + filters)	
Auto-detection of page orientation (90, 180, and 270 degrees)	This document imaging feature is very important for bulk input of images, when the direction in which document pages are scanned is unknown and can be different. The system automatically detects the orientation of each page and corrects it if needed.
Splitting facing pages and dual pages	It is used for scanning books as broadsides – for both left and right pages. The recognition quality is higher if the page is split into two, with each page corresponding to a single book page. Recognition and layout analysis are then performed separately for each page, along with the de-skewing if required.
Automated image de-skewing	It is an essential document imaging function which is applied to scanned documents requiring the compensation for image skew. It does not require leading edge borders or lines. New ABBYY FineReader Engine 10 provides several methods for de-skewing images: with pairs of black squares, lines or lines of text.
Lines	When capturing text from scanned or photographed books, the text lines may be uneven and difficult to OCR.

straightening	ABBYY technologies offer special algorithms that correct skew and straighten text lines for accurate text recognition.
Image despeckling (or image clean-up)	When scanning poor to medium quality documents, you may get very noisy images with lots of dots or speckles on them. These speckles, when they appear close to the letters or numbers, may affect the quality of OCR. This feature removes such noise. The size of the speckles to be removed may be specified by the user. Can be applied to an image as well as to any individual block (or zone) of the image.
Adaptive processing of digital photos	This technology automatically identifies digital photos and corrects distortions typically introduced by digital cameras. The system is aware of the typical defects commonly found in digital images, such as distorted text lines, and trapezoid 3D distortions, poor focus, smudge, darkened areas on facing pages in thick books, glare, ISO noise, etc. These defects are corrected by the system automatically, so that the user does not need any third-party applications to correct the photos. 
Texture filtering	Texture filtering technology helps to filter out background "noise" such as color and texture, increasing accuracy for difficult-to-read documents such as newsprint, color documents, faxes, and copies. 
Binarization	
Adaptive binarization	This is the process of converting images to black-and-white, removing noise, removing the background, removing the textures, and obtaining sharp text. The process ensures the best OCR quality. The required parameters are identified for each fragment separately. In the case of thin newspaper, the text printed on the reverse side may be visible on the scans. Adaptive binarization removes this text. Innovative Adaptive Binarization technology dynamically adjusts threshold of brightness for each image fragment during the recognition. By applying individual recognition parameters, it produces accurate recognition results for documents with gray or color-variable contrast background and textures.
Dithering	This is binarization of grayscale images using very small dots. It improves the appearance of the document, as the document appears to have more shades.
Filters for binary images	
Image Scaling	For documents scanned at lower resolutions (less than 120 dpi) and documents with small fonts (less than 10 pt), the images may be digitally enlarged to achieve better OCR quality.

See also

Usage Scenarios Implementation: Image Preprocessing
Key Features

Document Analysis

Basic document analysis features

Document Analysis is a set of functions for automatic detection of the following objects on a page:

- Text blocks
- Pictures
- Tables and table cells
- Barcodes
- Separators

Additionally document analysis provides some special features to prepare image for OCR:

- process detection of page orientation – 90, 180, and 270 degrees
- split double pages
- process vertical text detection in table cells
- detect and mark the blocks of garbage on page

This preparation is significantly important to specify which fields on page should be recognized and what should be kept in initial form.

And also there is an ability to designate the field for recognition manually. In this case you have to set field's coordinates and type of data inside. It is used in Field-Level Recognition scenario mostly for data capture.

ABBYY FineReader Engine 10 provides 3 automatic and 1 manual types of document analysis:

- General document analysis
- Document analysis for invoices
- Document analysis for full-text indexing
- Manual blocks specification for field-level recognition

General document analysis

This is default document analysis type which searches all objects: text blocks, pictures, tables, barcodes and separators. The results of this analysis are used for document structure and layout retrieval in content reuse scenario. All pictures and diagrams are preserved in original form without recognizing text on them.

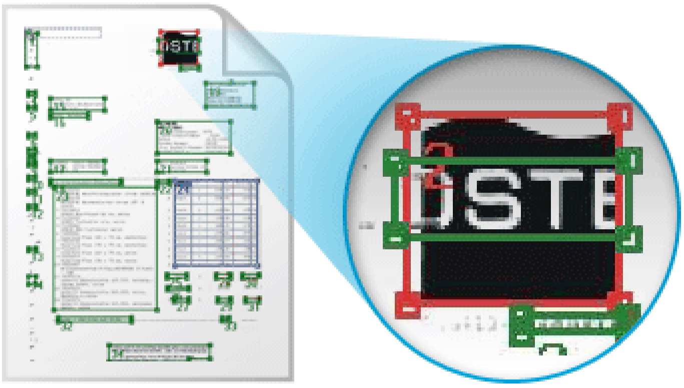
Document analysis for invoices

This is a preprocessing engine for converting semi-structured documents, such as invoices, payment drafts, bills, waybills, business cards, agreements, health claim forms, resumes, etc. It has been designed to accurately locate all the text on these documents, including characters and numbers — even if this information is located within stamps, pictures, logos or small-text areas.

Unlike the standard full-page document analysis, this one assumes that all printed information on documents is text. It also ensures that important text information is not identified as graphic elements and words or numerical values are not separated into multiple characters. As a result, maximum information about the text, including its coordinates, is available for analysis, field-by-field processing and parsing at subsequent processing stages by other systems.

Document analysis for full-text indexing

Automatically detects and recognizes all text on documents including text embedded in pictures, charts, and diagrams. Developers may choose to use this mode of document analysis to extract exhaustive full-text information on documents needed for document index building (as in DMS, CMS, Archiving systems).



Manual blocks specification for field-level recognition

This case does not need any analysis because the recognition field is directly defined by user or application. Recognizer receives the coordinates of field and type of text and process OCR in specified zone.

See also

Key Features

OCR and Other Recognition Technologies

Optical Character Recognition (OCR)

- **OCR** technology — printed text recognition is available for **198 languages**, including:
  - European languages (Latin, Cyrillic, Armenian, Greek alphabets)
  - **Chinese** (Simplified and Traditional), **Japanese**, and **Korean** (**CJK**)
  - **Thai**, **Vietnamese** and **Hebrew**
  - Arabic — technical preview version
  - **FineReader XIX** — an OCR module designed specifically for digitizing and archiving old documents, books and newspapers published in the XVII-XX centuries, many of which are rare and unique. Stored in the historical archives of libraries and government organizations, they are national heritage that must be preserved. FineReader XIX provides a unique capability to recognize texts published in the period from 1600 till 1937 in English, French, German, Italian and Spanish. It supports recognition of old fonts such as Fraktur, Schwabacher and the majority of Gothic fonts.

Beſchreibung der Venediger Le-
[illegible]

Der erſt Dialogus.

[illegible]

- 47 languages have **dictionary/morphology support** that is significantly improves OCR accuracy.

- **Multilingual documents** recognition feature provides recognition of several languages e.g. German and Chinese; English, Russian and Korean at the same document.

- **Dot-matrix documents** recognition — ABBYY FineReader Engine recognizes printed dot matrix texts of many types. It has been trained using several thousand samples produced by a variety of printers including dot matrix, daisy wheel, chain and band printers, as well as using draft and Near Letter Quality (NLQ) printing modes.

- **Typewritten documents** recognition.
- Recognition of **OCR-A**, **OCR-B**, **MICR (E13B)** and **CMC7** fonts.

See the full list of supported languages and text types.

Intelligent Character Recognition (ICR)

- **ICR** technology – hand-printing characters recognition for more than **110 languages**.
- About 30 languages (with Latin, Greek and Cyrillic alphabets) with **morphology/dictionary support** and 85 languages with Latin characters without dictionaries.
- **ICR for Indian digits** used in Arab states.
- **22 regional styles of hand-printing** used in different countries and regions of the world (for supported ICR languages).
- Recognition of hand-printed characters in fields and frames – underlined fields, boxes, comb-style fields, etc.
- **Multilingual ICR**. One of the main advantages of ABBYY ICR technology is that it delivers almost the same high accuracy on digits and digits combined with letters of one or several languages, even if the fields contain both upper and lower case letters.

Optical Mark Recognition (OMR)

The ABBYY's **OMR** technology recognizes simple checkmarks, grouped checkmarks, model checkmarks and checkmarks with "corrections" made by hand in different variations:

- char box series
- comb in frame
- gray boxes
- partitioned frame
- simple comb
- text in frame
- underlined text

OMR delivers **accuracy rate of 99.995 %**

Optical Barcode Recognition (OBR)

- **1D and 2D barcode types**. ABBYY **OCR SDK** supports recognition of popular types of 1D and 2D barcodes. See the list of supported types of barcodes.
- **Fast barcode extraction**. This feature enables automated detection and recognition of barcodes at any angle on a document. It works both for 1D and 2D barcodes

Recognition modes

With the Engine's pre-defined processing modes, developers have the ability to quickly set up and tune the processing speed and accuracy in a way which is the most appropriate for their needs. In addition to the default processing mode, both **OCR and ICR recognition** can be performed in normal, fast and balanced recognition modes:

- **Normal recognition mode**
  It is the most accurate mode for achieving the highest quality of recognition. This mode is highly recommended if you are planning to reuse recognized content and in other tasks when the accuracy is the critically important issue.
- **Fast recognition mode**
  It is designed for high-volume document processing and for the cases when speed is of primary importance. This mode increases processing speed by 200-250% making the technology ideal for using in content management (CMS), document management (DMS) and archiving systems.

- **Balanced recognition mode**
  It sets the intermediate values of recognition accuracy and speed between Normal and Fast modes. Generally it provides higher speed for almost the same accuracy level as Normal mode.

Full Text and Field-Level Recognition

There are two types of recognition which could be separated: full text and field-level recognition. The main difference is that full text recognition usually includes OCR technology and used for document conversion. Field-level recognition includes OCR, ICR and other technologies that are used in local area for recognizing and extraction particular data.

The following table shows specifications of these recognition types:

Specification	Full text recognition	Field-level recognition
Where is used	Document conversion, books archiving	Data capture
Document analysis	General document analysis, document analysis for invoices, document analysis for full-text indexing	Manual blocks specification for field-level recognition
Recognition	OCR with general accuracy about 96-99%	OCR, ICR, OMR, Barcodes recognition with predefined data types and values range. Accuracy is about 100%
Verification	Recommended for content reuse	Obligatory in most cases
Synthesis	Used for document retrieval	Not used
Export of recognition results	Document files (RTF, DOC, PDF, etc.)	Export to XML file or database

Full text recognition

Full text recognition is a basic recognition type for different tasks, like:

- Documents and books conversion for archiving
- Document conversion for content reuse
- Ground text extraction for fields detection and documents classification

All of them require the recognition (OCR) of whole text on document (page). Before recognition the document analysis usually processes for splitting and correct orientation of pages, detection of text blocks, pictures and other objects.

Then after OCR document synthesis rebuilds the structure and layout of document (for content reuse task) or just retrieves the correct text order for complex documents with several text columns and pictures (for archive scenario). Resulted text is exported depending on task as pure text or as a document of supported format.

The text could be manually verified for increasing its accuracy, especially for future reuse.

Field-level recognition

ABBYY FineReader Engine 10 delivers complete field-level recognition capabilities to support key business processes such as forms processing, keyword classification, and keyword indexing. Powerful image processing functions increase its ability to intelligently detect small zone areas of any quality, with any type of graphic specifics which may affect the recognition accuracy (i.e. underlined text, after-scanning garbage, spaces in the text, etc.)

Key functionality for field-level or zonal recognition includes multilingual OCR and ICR, OMR, barcode recognition and a range of specific functions, such as:

- Data extraction from fields with various borders and frames, including combo-box, underlined fields, boxes, and even fields where the data does not fit within the field border
- Definition of field content by setting alphabets, dictionaries, regular expressions, types of segmentations, handwriting styles, etc.
- Detection of in-field spacing, accurately recognizing fields where the spaces are allowed. ABBYY FineReader Engine 10 also allows use of dictionaries which contain word combinations with spaces
- Intelligent processing of blocks with intersecting parts and lines, provides recognition of text (words and symbols) located entirely within the block borders, saving time spent on non-relevant text block recognition
- Text block despeckle, with the ability to specify the size of white or black "garbage"

Field-level recognition is supported by the Engine’s special tools for developers such as Voting API and "On-the-Fly" Recognition Tuning. For details, please see Advanced Development Tools.

User Languages

ABBYY FineReader Engine provides an API for creating and editing recognition languages, creating copies of predefined recognition languages and adjusting them, and adding new words to user languages.

Below are two examples illustrating how user languages can help you to improve recognition quality:

- In documents filled out by hand, the values in the form fields usually belong to a specific set such as city names, countries, zip codes, product codes, sums, etc. To improve the quality of ICR recognition, you can use user languages to describe the information which may be entered in each field.
- If a document contains "structures" such as product codes, telephone numbers, passport numbers etc., recognition errors may occur. This happens because the program reads such structures letter by letter. To improve the recognition of product codes and the like, you can create a new recognition language which will help the program to read specific types of data correctly.

Pattern Training

In the vast majority of cases ABBYY FineReader Engine can successfully read texts without prior training. However, in such cases as recognition of decorative or outlined fonts or bulk input of low print quality documents, preliminary pattern training will prove useful.

The OCR SDK allows you to create and exploit user patterns directly via API or import them from the ABBYY FineReader desktop application (Professional or Corporate Edition). Since ABBYY FineReader Engine 10th version you can “teach patterns” by loading pictures and matching corresponding characters.

See also

Key Features
Advanced Development Tools

PDF Conversion

The PDF format is often used in electronic archives for data storage purposes. It is the format of choice because of its versatility and possibility to keep both images and text.

Technologies developed by ABBYY allow recognized texts to be saved in PDF and PDF/A formats. One of the main goals of archiving is to achieve the smallest file size possible without losing in data quality.

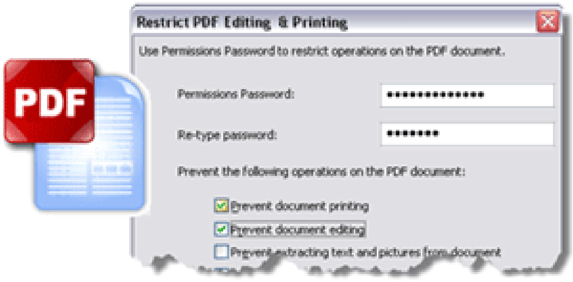
A special compression technology called MRC (Mixed Raster Content) is used to minimize the size of PDF and PDF/A files.

PDF Input

Intelligent PDF processing	ABBYY FineReader Engine analyses internal information within the source PDF files such as: • annotations, • metadata, • text objects, • font dictionaries • content stream SDK enhances PDF conversion performance and speed by efficient and accurate text selection. If text is embedded into the PDF file, the OCR engine examines the integrity of the text layer, and makes a decision as to whether or not to extract the text or apply OCR on a block by block basis.
Capture of internal PDF information	It extracts internal PDF links, hyperlinks and document properties such as: subject, author, title, and keywords.

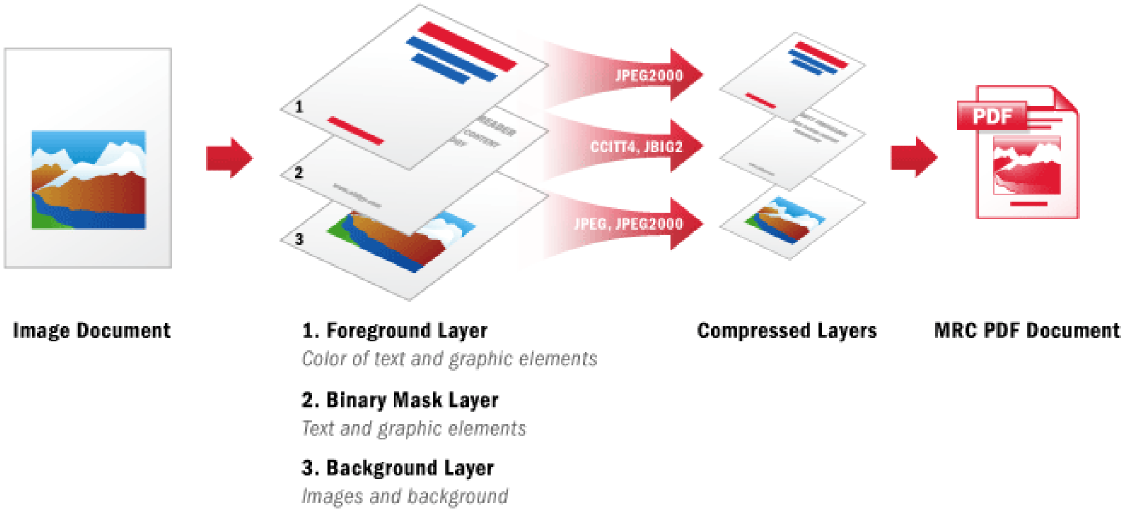
PDF Output

PDF security and encryption support	ABBYY FineReader Engine 10 supports a variety of PDF security settings, increasing its applicability for government agencies and other organizations demanding high security. • "Open File" password settings designed to prevent unauthorized access to a document. • Restriction of certain operations, such as printing, editing or extracting file content, by assigning permission passwords.

	- Support for the latest encryption standards.  Restrict PDF Editing & Printing Use Permissions Password to restrict operations on the PDF document. Permissions Password: ************** Re-type password: ******* Prevent the following operations on the PDF document: ☑ Prevent document printing ☑ Prevent document editing □ Prevent extracting text and pictures from document
Output in Tagged PDF format	Tagged PDF can be "reflowed" to fit different page or screen widths. Ideal for use with handheld devices (PDAs) or screen readers typically used by visually impaired users.
Page size	Ability to set the size for all pages of an output file during PDF conversion.
Metadata export	ABBYY FineReader Engine 10 enables metadata exporting (bookmarks, hyperlinks, cross-references, etc.).
Conversion to PDF/A format	Conversion to PDF/A format which is recommended as a standard for long-term preservation of page-oriented documents. ABBYY's technologies allow saving documents to PDF/A formats of different compliance levels: PDF/A-1a, PDF/A-1b. The PDF/A-1a format has the following features: best retention of document formatting, logical structure, and ordinary appearance as well as the possibility of retaining the document appearance when using displays of different sizes (the document content is organized in a specific way to achieve this). The PDF/A-1b format is used to reproduce the document appearance only. When processing by ABBYY technologies, documents are saved in PDF/A-1b format by default.
CJK to PDF export	Enables conversion of documents in Chinese (both simplified and traditional), Japanese and Korean into PDF format.

PDF (PDF/A) MRC compression

A special compression technology called MRC (Mixed Raster Content) is used to minimize the size of PDF and PDF/A files.



Document image files are usually very large due to the background, which is often makes up to 90% of the file size. The background may, however, be unnecessary in the resulting document. It is the text and pictures that are important.

The MRC compression technology allows locating the color background and deleting it or compressing to a high degree. This leaves text and pictures against a white background contributing to smaller file size.

Picture objects (diagrams, graphs, logos, photos, drawings, stamps, signatures, etc.) are also slightly compressed, but only to an extent that doesn't lower the quality.

The MRC technology analyzes the outlines of similar characters in the document, creates an average character template and uses it instead of a character itself. This leads to better readability, because some of the text defects are corrected, and the character outlines become more precise.

As a result, you get a smaller image which looks even better than before. The resulting document will have an unobtrusive bland background with fine text and pictures.

This “reconstruction” of the document can be useful when you have to deal with low quality images due to: bad lighting, out-of-focus photo, incorrect scanning/photo parameters, dark uncoated paper, or document dilapidation.

All this results in the image having a dark background with additional textures. The text appears blurred and difficult to read.
The MRC technology allows for better document appearance and up to 8-10 smaller file size than JPEG.

Clear and simple PDF Conversion

ABBYY FineReader Engine provides developers with special tools to achieve the optimal PDF conversion mode appropriate for their particular needs.

PDF Export Scenario	Description
MaxQuality	Optimize the PDF (PDF/A) export in order to receive the best quality of the resulting file.
Balanced	The PDF (PDF/A) export will be balanced between the quality of the resulting file, its size and the time of processing.
MinSize	Optimize the PDF (PDF/A) export in order to receive the minimum size of the resulting file.
MaxSpeed	Optimize the PDF (PDF/A) export in order to receive the highest speed of processing.

See also

Key Features

Advanced Development Tools

Useful tools that enhance the developer's ability to interact with ABBYY FineReader Engine and manipulate the recognition process on the core level:

Working with Profiles

ABBYY FineReader Engine 10 provides a set of predefined profiles which are already fine-tuned for the basic usage scenarios. The settings specified in these profiles provide the best results in the corresponding situations. Besides, most of the profiles come in two forms: with the settings optimized for the best quality of the resulting document or with the settings optimized for the highest speed of processing. Below is a list of available predefined profiles:

Scenario	Profile Name
Document archiving	• DocumentArchiving_Accuracy • DocumentArchiving_Speed
Book archiving	• BookArchiving_Accuracy • BookArchiving_Speed
Document conversion for content reuse	• DocumentConversion_Accuracy • DocumentConversion_Speed
Text extraction for fields detection and documents classification	• TextExtraction_Accuracy • TextExtraction_Speed
Field-level recognition	• FieldLevelRecognition

Barcode recognition	• BarcodeRecognition

📝**Note:** You can view the list of settings provided by these profiles in the description of corresponding scenarios.

The settings provided with these profiles can be loaded using the **LoadPredefinedProfile** method of the **Engine** object. After the profile is loaded, newly created objects will have the new default values specified in the profile.

Voting API support

When ABBYY FineReader Engine is used as one of the participating recognition engines in a third-party application, it supplies recognition alternatives (or hypotheses) with a relevant confidence level for characters, words and intercharacter separation. This information helps developers design an efficient and accurate voting algorithm for applications that require multiple recognition technologies. For example, when recognizing an "O", ABBYY FineReader Engine may return 3 hypotheses: "0" (zero), with 60% confidence; capital "O", with 80% confidence; and capital "C", with 10% confidence. Another example for intercharacter separation: the possible hypotheses for an "m" would be "m", "rn", and "in". See more in Using Voting API.

"On-the-fly" tuning of core recognition

ABBYY FineReader provides developers with the access and ability to manipulate the recognition engine during the OCR process on a core level. The FineReader recognition engine generates hypotheses (or recognition alternatives) and allows developers to influence or fine-tune the procedure of setting the confidence level for each hypothesis (or selecting the best hypothesis) using their own specific ranking criteria.

Sample Codes for common conversion tasks

The SDK is supplied with the set of Source Code Samples showing how to use the Engine in different scenarios. The Code Samples are provided for Visual Basic, Visual Basic .Net, Delphi, raw C++, C++ with the Native COM Support, C#, and script languages.

See also

Key Features

Receiving and Exporting Recognized Text

The FineReader Engine OCR API provides a wide range of options for export of recognition results on different levels of document reconstruction:

- Various levels of text format retention when exporting to external formats (from **simple text without formatting** to **complete page layout retention**, including columns, tables, frames, fonts, font size, paragraph styles, borders, etc.)
- Access to detailed information about each recognized character
- A set of functions for post-editing and post-formatting of the recognized text before prior to export
- Export of recognized text to various formats:
  - RTF
  - DOC/DOCX
  - XLS/XLSX
  - PPTX
  - PDF
  - PDF/A
  - HTML
  - TXT/CSV
  - XML

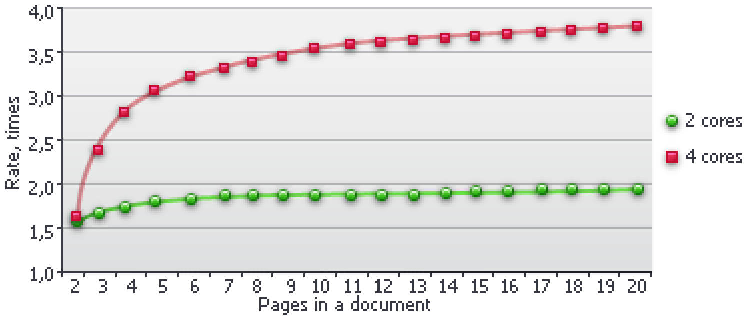
  See Export Formats.
- Replacing uncertain characters with the corresponding images when saving to PDF
- Retaining text color and pictures of original image into all export formats

See also

Key Features
Tuning Export Parameters

Multi-CPU Recognition Architecture

ABBYY FineReader Engine automatically combines and executes steps of distributing pages, and coordinating recognition and synthesis. That provides easy scalability and utilization of multi Core/CPU hardware and brings up to 90% of speed increase for each additional core comparing to one-core systems.



Note: This graphic does not take into account document export step because it could vary from scenario to scenario and can't be paralleled. Speed increase rate can also be different depending on a document complexity. For documents with complex layout it is higher, for simple – lower. The more time spent for analysis and recognition – the higher benefit from multi-processing. The graphic also shows that the more pages are in a document the more effective load balancing and performance rate.

Numbers quoted are based on internal ABBYY testing.

See also

Key Features

Benefits

- Choose ABBYY FineReader Engine 10 and get award-winning OCR SDK providing unrivaled accuracy, high recognition speed, outstanding functionality and 198 supported languages.
- Enjoy working with comprehensive, easily-integrated API supplied with clear documentation.
- Appreciate unique set of breakthrough technologies including improved ADRT™, Camera OCR™, new binarization and others.
- Expand your markets with ABBYY SDK's multiple OS support: Windows, Linux, Mac OS and variety of embedded platforms.
- Trust in ABBYY's proven partnerships with industry leaders worldwide who have been choosing ABBYY's technologies for decades.

See also

Key Features

Short Specifications

- OCR for 198 languages including:
  - European (Armenian, Cyrillic, Greek, Latin alphabets)
  - Asian (Chinese, Japan, Korean, Taiwanese, Vietnamese, Thai)
  - Arabic
  - Hebrew
  - Old fonts (English, French, German, Italian, Spanish)

- ICR for 110 languages (Cyrillic, Greek, Latin alphabets)
- OMR
- Barcodes 1D (15 types) and 2D (PDF417, Aztec, DataMatrix, QR Code)
- Recognition modes (Normal, Balanced, Fast)
- Text font types (Matrix, MICR E13B, MICR CMC7, Normal, OCR-A, OCR-B, Gothic, Typewriter)
- Import formats
    - Scanning (API, TWAIN UI, FineReader UI)
    - Image files (BMP, DCX, DjVu, GIF, JBIG2, JPEG, JPEG 2000, PCX, PNG, TIFF)
    - Memory Image Formats (Raw, Bitmap [HBITMAP], DIB)
    - PDF formats (Extracting text layer, Image only, Vectorized, Password protected)
- Export formats
    - HTML, RTF/DOC/DOCX, XLS/XLSX, PPTX, TXT/CSV, ABBYY XML
    - PDF formats (Image Only/Image on Text/Text and Images/Text on Image/Font embedding, PDF MRC), PDF/A-1a, PDF/A-1b
    - ODF (Open Office document format), EPUB, FB2, ALTO – available in Maintenance release
    - Image formats (BMP, DCX, JBIG2, JPEG, JPEG 2000, PCX, PNG, TIFF)

See also

Supported Image Formats
List of the Predefined Languages
Text Types
Barcode Types
Export Formats
Specifications

Getting Started

We recommend starting your work with ABBYY FineReader Engine with selecting the appropriate scenario. After you found the appropriate scenario, you can find a detailed description of the scenario, implementation advice, and suggestions on optimizing the code for specific tasks in the Basic Usage Scenarios section.

If your task is not compatible with any of the basic scenarios, you may find useful advices in the Advanced Techniques section. We recommend you to refer to the Programming Aspects section, where you can find useful information on using ABBYY FineReader Engine in different programming languages. Either you can view Sample codes provided with the ABBYY FineReader Engine developer package for quick start.

To start your work with FineReader Engine API, you should create the **Engine** object with the **GetEngineObject** function. The detailed API Reference you can also find in this Developer's Help.

Guided Tour

This section contains information which will help you in your work with ABBYY FineReader Engine 10:

- **Basic Usage Scenarios Implementation**
  Describes the main scenarios in which ABBYY FineReader Engine can be used. We recommend that you begin work with ABBYY FineReader Engine by selecting the scenario most suitable for your task.

- **Advanced Techniques**
  Provides advanced information about working with the ABBYY FineReader Engine API, including information on tuning the parameters of document processing, working with images, languages, recognized texts, special recognition cases such as recognition of hieroglyphic languages, checkmarks, handprinted texts, and recognition with training.

- **Best Practices**
  Offers you some advice on how to prepare images for recognition.

- **Samples**
  Provides a short description of the samples. A detailed description of the samples is available in the **Code Samples Library** provided with this distribution pack.

Basic Usage Scenarios Implementation

This section describes the most common scenarios in which ABBYY FineReader Engine may be used. Each article contains a detailed description of the scenario, implementation advice, and suggestions on optimizing the code for specific tasks.

Select the scenario appropriate for your task:

- **Document Conversion**
  Suitable for converting documents into an editable format.

- **Document Archiving**
  Suitable for processing paper documents for electronic archives.

- **Book Archiving**
  Suitable for processing books, magazines, and newspapers for electronic libraries.

- **Text Extraction**
  Suitable for extracting entire text from documents to make them searchable and to extract useful data.

- **Field-Level Recognition**
  Suitable for recognition of small text fragments to capture data from document fields.

- **Barcode Recognition**
  Suitable for reading barcodes.

- **Image Preprocessing**
  Suitable for preparing images for further processing or for improving their visual quality.

- **Scanning**
  Suitable for getting images from a scanner and their subsequent processing.

Document Conversion

The result of this scenario is an editable version of a document.

In this scenario, document images are recognized, retaining all the original formatting intact, and the data are saved to an editable file format. As a result, you get editable versions of your documents, which can be easily checked for errors and modified. You will also be able to copy all or some of the text for re-use.

A document goes through several processing steps, which are in some ways slightly different from the other common scenarios:

1. **Image preprocessing**
   Images you get by means of a scanner or a digital camera may need some tweaking before they can be optically recognized.

For example, noisy images or images with distorted text lines will need some correction for optical recognition to be successful.

2. **Recognition**
   When recognizing a document, various layout elements (text, tables, images, separators, etc.) of the document are identified. In the course of the document synthesis, the logical structure of the document is restored, while the page synthesis enables one to fully restore the document formatting (fonts, styles, etc.)

3. **Export**
   The recognized document is saved to an editable format, such as RTF, DOC, DOCX.

Scenario implementation

Below is the detailed description of a recommended method of using ABBYY FineReader Engine 10 for the implementation of the above scenario. The proposed method employs the processing settings that are most suitable for the above scenario.

Step 1. Loading ABBYY FineReader Engine

To start your work with ABBYY FineReader Engine you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only ABBYY FineReader Engine externally creatable object.

To create the **Engine** object use the **GetEngineObject** exported function.

Sample code for the procedure of ABBYY FineReader Engine loading and initialization in C++ and Visual Basic:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;

void LoadFREngine()
{
        if( Engine != 0 ) {
                // Already loaded
                return;
        }
        // First step: load FREngine.dll
        if( libraryHandle == 0 ) {
                libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
                if( libraryHandle == 0 ) {
                        throw L"Error while loading ABBYY FineReader Engine";
                }
        }

        // Second step: obtain the Engine object
        typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR,
FREngine::IEngine** );
        GetEngineObjectFunc pGetEngineObject =
                ( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject"
);

        if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0,
&Engine ) != S_OK ) {
                UnloadFREngine();
                throw L"Error while loading ABBYY FineReader Engine";
        }
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
```

```
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
             ByVal DeveloperSN As String, _
             ByVal Reserved1 As String, _
             ByVal Reserved2 As String, _
             EngineObj As FREngine.Engine) As Long

Sub Engine_Load(ByVal DeveloperSN As String)
       ' Visual Basic may load libraries from the current path only
       ChDir "Path to the folder with FREngine.dll"

       ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
       Dim DeveloperSN_WideChar As String
       DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)

       If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
             MsgBox "Error loading ABBYY FineReader Engine"
       End If
End Sub
```

Step 2. Loading setting for the above scenario

ABBYY FineReader Engine enables loading of all processing settings that are most suitable for this scenario using the **LoadPredefinedProfile** method of the **Engine** object. This method uses the name of a used settings profile as an input parameter. Please see Working with Profiles for more information.

ABBYY FineReader Engine supports 2 options of settings for this scenario. Both these profiles enable font style detection and full document synthesis:

- *DocumentConversion_Accuracy*
  This profile optimizes the document conversion process in order to ensure that the resulting document is of the highest quality possible.

- *DocumentConversion_Speed*
  This profile optimizes the processing speed of the document conversion process: the processes of document analysis and recognition are sped up.
  ⚠**Important!** This profile requires the Fast Mode module available in the license.

Sample code for the procedure of profile loading in C++ and Visual Basic:

Visual C++ (COM) code

```
// Load a predefined profile
Engine->LoadPredefinedProfile( L"DocumentConversion_Speed" );
```

Visual Basic code

```
' Load a predefined profile
Engine.LoadPredefinedProfile "DocumentConversion_Speed"
```

If you wish to change processing settings, use appropriate parameter objects. Please see Additional optimization for specific tasks below for further information.

Step 3. Loading and preprocessing of images

ABBYY FineReader Engine provides the **FRDocument** object which allows processing multi-page documents. Use of this object allows you to preserve the logical organization of the document, retaining the original text and columns, fonts, styles, etc.

To load images of a single document and preprocess them, you should create the **FRDocument** object and add images into it. You may do one of the following:

- Create the **FRDocument** object using the **CreateFRDocumentFromImage** method of the **Engine** object. This method creates the **FRDocument** object and loads images from the specified file.

- Create the FRDocument object with the help of the **CreateFRDocument** method of the Engine object, then add images to the created FRDocument object from file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method of the FRDocument object).

Sample code for the procedure of image loading and preprocessing in C++ and Visual Basic:

Visual C++ (COM) code

```
// Open image file and create the FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage(
L"C:\\MyImage.tif", 0 );
```

Visual Basic code

```
' Open image file and create the FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif")
```

Note: Image preprocessing with the loaded *DocumentConversion_Accuracy* or *DocumentConversion_Speed* profile does not include the orientation detection. If you want orientation to be automatically detected, you will need to tune additional parameters and pass corresponding object to the preprocessing function. Please refer Additional optimization for specific tasks below for further information.

Step 4. Document recognition

To recognize a document, we suggest that the analysis and recognition methods of the **FRDocument** object be used. This object provides a whole array of methods for document analysis, recognition and synthesis. The most convenient method allowing document analysis, recognition and synthesis using just one method is the **Process** method. It also uses simultaneous processing features of multiprocessor and multicore systems in the most efficient manner. However, you may also carry out consecutive analysis, recognition and synthesis using **Analyze**, **Recognize** (or **AnalyzeAndRecognize**) and **Synthesize** methods.

Sample code for the procedure of document recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
// Analyze, recognize, and synthesize the document.
// While the profile is loaded, you do not need to pass any additional parameters to
the processing method.
pFRDocument->Process( 0, 0, 0 );
```

Visual Basic code

```
' Analyze, recognize, and synthesize the document.
' While the profile is loaded, you do not need to pass any additional parameters to the
processing method.
frDocument.Process
```

Step 5. Document export

To save a recognized document, you may use the **Export** method of the **FRDocument** object by assigning the **FileExportFormatEnum** constant as one of the parameters. You may change the default parameters of export using the corresponding export object. Please see Additional optimization for specific tasks below for further information.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object. Use the **IFRDocument::Close** method.

Sample code for the procedure of document export to RTF in C++ and Visual Basic:

Visual C++ (COM) code

```
// Save a recognized document to an editable format (e.g. RTF)
frDocument->Export(L"C:\\MyText.rtf", FREngine::FEF_RTF, 0);
// Release the FRDocument object
frDocument->Close();
```

Visual Basic code

```
' Save a recognized document to an editable format (e.g. RTF)
frDocument.Export "C:\MyText.rtf", FEF_RTF, Nothing
' Release the FRDocument object
frDocument.Close
```

Step 6. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine you need to unload the **Engine** object. To do this use the **DeinitializeEngine** exported function.

Sample code for the procedure of ABBYY FineReader Engine unloading and deinitialization in C++ and Visual Basic:

Visual C++ (COM) code

```
void UnloadFREngine()
{
        if( libraryHandle == 0 ) {
                return;
        }
        // Release Engine object
        Engine = 0;
        // Deinitialize FineReader Engine
        typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
        DeinitializeEngineFunc pDeinitializeEngine =
                ( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
        if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
                throw L"Error while unloading ABBYY FineReader Engine";
        }
        // Now it's safe to free the FREngine.dll library
        FreeLibrary( libraryHandle );
        libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization for specific tasks

Below is the overview of the Help topics containing additional information regarding customization of settings at different stages of the document conversion to an editable format:

- **Scanning**
  - **Scanning**
    Description of the ABBYY FineReader Engine scenario for document scanning.
- **Opening and preprocessing**
  - **Image Preprocessing**
    Description of the ABBYY FineReader Engine scenario for preliminary preparation of images or enhancement of their visual quality.
- **Recognition**
  - **Tuning Analysis, Recognition, and Synthesis Parameters**
    Customization of document processing using objects of analysis, recognition and synthesis parameters.
  - **PageProcessingParams Object**
    This object enables customization of analysis and recognition parameters. Using this object, you can indicate which image and text characteristics must be detected (inverted image, orientation, barcodes, recognition language, recognition error margin).
  - **SynthesisParamsForPage Object**
    This object includes parameters responsible for restoration of a page formatting during synthesis.
  - **SynthesisParamsForDocument Object**
    This object enables customization of the document synthesis: restoration of its structure and formatting.

  - **MultiProcessingParams Object**
    Simultaneous processing of documents may be useful when processing a large number of documents. In this case the document load will be spread over the processor cores during the analysis and recognition, which makes it possible to speed up processing. Reading modes (simultaneous or consecutive) are set using the **MultiProcessingMode** property. The **RecognitionProcessesCount** property controls the number of processes, which may be started.

- **Export**

  - **Tuning Export Parameters**
    Customization of the document export using objects of export parameters.

  - **RTFExportParams Object**
    This object enables customization of the RTF/DOC/DOCX saving format parameters.

  - **HTMLExportParams Object**
    This object allows customization of export to the HTML format.

  - **PPTExportParams Object**
    Object for customization of the PPTX saving format parameters.

See also

Basic Usage Scenarios Implementation

Document Archiving

This scenario is used for processing paper documents to save them to an electronic archive, especially when creating an archive of agreements, project documentation, invoices, certificates, etc.

Under this processing scenario, paper documents are converted into uneditable electronic copies containing all document information in searchable format. As a result of such processing, the resulting copies may be easily found in the electronic archive using full-text search, document text segments may be copied and the document may be sent by email or printed out.

To create an electronic copy, the document first needs to go through several processing stages, each of which has its own peculiarities in this scenario:

1. **Scanning**
   Scanning may be done manually for each separate document as well as automatically by scanning a whole batch of documents. In the latter case, a batch of images may have to be separated additionally into documents after scanning.

2. **Image preprocessing**
   Scanned images may require some preprocessing prior to recognition, for example, if scanned documents contain background noise, skewed text, inverted colors, black margins, wrong orientation or resolution.

3. **Recognition**
   To extract text data from a document, the document recognition is required. When processing a large volume of documents, simultaneous document processing may be come in useful. In this case, in the course of analysis and recognition the document load will be spread over the processor cores, which makes it possible to speed up processing.

4. **Export**
   The recognized document is saved to a suitable storage format. The most convenient formats for storing documents are PDF, PDF/A, PDF and PDF/A with MRC. When saving to these formats, one may use a mode, under which the text is placed underneath the document image — this enables full preservation of the document formatting and provides a full-text search. The MRC settings allow significant reduction of a file size without loss of visual quality. Also when saving to the PDF format, one may customize security settings of the document protecting it from unauthorized viewing and printing.

Scenario implementation

Below is the detailed description of the recommended method of using ABBYY FineReader Engine 10 for implementation of the above scenario. The proposed method uses processing settings that are most suitable for this scenario. Under the proposed implementation of the scenario, the document scanning phase is omitted. Please see Additional optimization for specific tasks below for the tips on scanning implementation.

Step 1. Loading ABBYY FineReader Engine

To start your work with ABBYY FineReader Engine you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only ABBYY FineReader Engine externally creatable object.

To create the **Engine** object use the **GetEngineObject** exported function.

Sample code for the procedure of ABBYY FineReader Engine loading and initialization in C++ and Visual Basic:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;

void LoadFREngine()
{
        if( Engine != 0 ) {
                // Already loaded
                return;
        }
        // First step: load FREngine.dll
        if( libraryHandle == 0 ) {
                libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
                if( libraryHandle == 0 ) {
                        throw L"Error while loading ABBYY FineReader Engine";
                }
        }

        // Second step: obtain the Engine object
        typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR,
FREngine::IEngine** );
        GetEngineObjectFunc pGetEngineObject =
                ( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject"
);

        if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0,
&Engine ) != S_OK ) {
                UnloadFREngine();
                throw L"Error while loading ABBYY FineReader Engine";
        }
}
```

Visual Basic code

```
Public Engine As FREngine.Engine

Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                ByVal DeveloperSN As String, _
                ByVal Reserved1 As String, _
                ByVal Reserved2 As String, _
                EngineObj As FREngine.Engine) As Long

Sub Engine_Load(ByVal DeveloperSN As String)
        ' Visual Basic may load libraries from the current path only
        ChDir "Path to the folder with FREngine.dll"

        ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
        Dim DeveloperSN_WideChar As String
        DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)

        If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
```

```
            MsgBox "Error loading ABBYY FineReader Engine"
      End If
End Sub
```

Step 2. Loading settings for the above scenario

ABBYY FineReader Engine enables loading of all processing settings that are most suitable for this scenario using the **LoadPredefinedProfile** method of the **Engine** object. This method uses the name of a used settings profile as an input parameter. Please see Working with Profiles for more information.

ABBYY FineReader Engine supports 2 options of settings for this scenario. Both these profiles enable detection of all text on an image, including text embedded into the image, while skew correction is not performed, fonts and styles are not detected, and full document synthesis is not performed:

- *DocumentArchiving_Accuracy*
  This profile optimizes the document archiving process in order to ensure that the resulting document is of the highest quality possible.

- *DocumentArchiving_Speed*
  This profile optimizes the processing speed of the document archiving process: the processes of document analysis and recognition are sped up.

⚠**Important!** These profiles require the DA for Full-Text Indexing module available in the license. The *DocumentArchiving_Speed* profile requires additionally the Fast Mode module.

Note: The settings provided by these predefined profiles are not intended for converting a document into an editable format. Use the document conversion profiles for such purpose.

Sample code for the procedure of profile loading in C++ and Visual Basic:

Visual C++ (COM) code

```
// Load a predefined profile
Engine->LoadPredefinedProfile( L"DocumentArchiving_Accuracy" );
```

Visual Basic code

```
' Load a predefined profile
Engine.LoadPredefinedProfile "DocumentArchiving_Accuracy"
```

If you wish to change processing settings, use appropriate parameter objects. Please see Additional optimization for specific tasks for further information.

Step 3. Loading and preprocessing of images

ABBYY FineReader Engine provides the **FRDocument** object which allows processing multi-page documents. Use of this object allows you to preserve the logical organization of the document.

To load images of a single document and preprocess them, you should create the **FRDocument** object and add images into it. You may do one of the following:

- Create the **FRDocument** object using the **CreateFRDocumentFromImage** method of the **Engine** object. This method creates the **FRDocument** object and loads images from the specified file.

- Create the FRDocument object with the help of the **CreateFRDocument** method of the Engine object, then add images to the created FRDocument object from file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method of the FRDocument object).

Sample code for the procedure of image loading and preprocessing in C++ and Visual Basic:

Visual C++ (COM) code

```
// Open image file and create the FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage(
L"C:\\MyImage.tif", 0 );
```

Visual Basic code

```
' Open image file and create the FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif")
```

Step 4. Document recognition

To recognize a document, we suggest that the methods of the **FRDocument** object analysis and recognition be used. This object provides a whole array of methods for document analysis, recognition and synthesis. The most convenient method allowing document analysis, recognition and synthesis using just one method is the **Process** method. It also uses simultaneous processing features of multiprocessor and multicore systems in the most efficient manner. However, you may also carry out consecutive analysis, recognition and synthesis using the **Analyze**, **Recognize** (or **AnalyzeAndRecognize**) and **Synthesize** methods.

Sample code for the procedure of document recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
// Analyze, recognize, and synthesize the document.
// While the profile is loaded, you do not need to pass any additional parameters to
the processing method.
pFRDocument->Process( 0, 0, 0 );
```

Visual Basic code

```
' Analyze, recognize, and synthesize the document.
' While the profile is loaded, you do not need to pass any additional parameters to the
processing method.
frDocument.Process
```

Step 5. Document export

To save a recognized document, you may use the **Export** method of the **FRDocument** object by assigning the **FileExportFormatEnum** constant as one of the parameters. In this scenario you can save the document, for example, to the PDF format using MRC in the export mode PEM_ImageOnText (property **TextExportMode** of the **PDFExportParams** object). You may change the default parameters of export using the corresponding export object. Please see Additional optimization for specific tasks below for further information.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object. Use the **IFRDocument::Close** method.

Sample code for the procedure of document export to PDF in C++ and Visual Basic:

Visual C++ (COM) code

```
// Save a recognized document to an archive format (e.g. PDF)

 // Create a PDFExportParams object
 FREngine::IPDFExportParamsPtr params = Engine->CreatePDFExportParams();
 // Set necessary parameters
 params->MRCMode = FREngine::MRC_Auto;
 params->TextExportMode = FREngine::PEM_ImageOnText;

 // Use the parameters during export
 frDocument->Export(L"C:\\MyText.pdf", FREngine::FEF_PDF, params);

 // Release the FRDocument object
 frDocument->Close();
```

Visual Basic code

```
' Save a recognized document to an archive format (e.g. PDF)

' Create a PDFExportParams object
Dim params As FREngine.PDFExportParams
Set params = Engine.CreatePDFExportParams
' Set necessary parameters
params.MRCMode = MRC_Auto
params.TextExportMode = PEM_ImageOnText

' Use the parameters during export
frDocument.Export "C:\MyText.pdf", FEF_PDF, params
' Release the FRDocument object
frDocument.Close
```

Step 6. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine you need to unload the **Engine** object. To do this use the **DeinitializeEngine** exported function.

Sample code for the procedure of ABBYY FineReader Engine unloading and deinitialization in C++ and Visual Basic:

Visual C++ (COM) code

```
void UnloadFREngine()
{
        if( libraryHandle == 0 ) {
                return;
        }
        // Release Engine object
        Engine = 0;
        // Deinitialize FineReader Engine
        typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
        DeinitializeEngineFunc pDeinitializeEngine =
                ( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
        if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
                throw L"Error while unloading ABBYY FineReader Engine";
        }
        // Now it's safe to free the FREngine.dll library
        FreeLibrary( libraryHandle );
        libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization for specific tasks

Below is the overview of the Help topics containing additional information regarding customization of settings at different stages of document processing:

- **Scanning**
  - **Scanning**
    Description of the ABBYY FineReader Engine scenario for document scanning.
  - **Tips for Document Scanning**
    Getting quality images from scanning paper documents.
  - **Setting up Scanning Options**
    Implementing scanning using ABBYY FineReader Engine scanning interfaces.
- **Opening and preprocessing**
  - **Image Preprocessing**
    Description of the ABBYY FineReader Engine scenario for preliminary preparation of images and enhancement of their visual quality.
- **Recognition**
  - **Tuning Analysis, Recognition, and Synthesis Parameters**
    Customization of document processing using objects of analysis, recognition and synthesis parameters.
  - **PageProcessingParams Object**
    This object enables customization of analysis and recognition parameters. Using this object, you can indicate which image and text characteristics must be detected (inverted image, orientation, bar codes, recognition language, recognition error margin).

  - **SynthesisParamsForPage Object**
    This object includes parameters responsible for restoration of a page formatting during synthesis.

  - **SynthesisParamsForDocument Object**
    This object enables customization of the document synthesis: restoration of its structure and formatting.

  - **MultiProcessingParams Object**
    Simultaneous processing of documents may be useful when processing a large number of documents. In this case the document load will be spread over the processor cores during the analysis and recognition, which makes it possible to speed up processing. Reading modes (simultaneous or consecutive) are set using the **MultiProcessingMode** property, the **RecognitionProcessesCount** property controls the number of processes, which may be started.

- **Export**

  - **Tuning Export Parameters**
    Customization of document export using objects of export parameters.

  - **PDFExportParams Object**
    This object allows you to tune PDF (PDF/A) export with only several parameters.

  - To customize the PDF (PDF/A) format export mode, use the **TextExportMode** property of the **PDFExportParams** object, and to customize MRC settings, use the **MRCMode** property.

  - In addition, you can customize image export settings to ensure faster processing, additional reduction of a file size, etc. For example, you can save a colored image as a grayscale or black and white image, if this fits your scenario (use the **Colority** property of the **PDFExportParams** object).

  - You can change the image resolution in such a way that the resulting electronic copy may subsequently be printed out on a printer, viewed on a computer screen or you can select low resolution allowing only for reading of text and providing very poor quality of graphics (use the **Resolution** and **ResolutionType** property of the **PDFExportParams** object).

- **Separation into documents**

  - Under this scenario, the batch of images may have to be separated into documents. ABBYY FineReader Engine 10 does not support automatic document separation. However, you can use ABBYY FlexiCapture Engine to implement automatic separation. The documents may be separated, for instance, based on the number of pages in a document or based on pages having separating barcodes. When implementing barcode separation, you can use the scenario for extraction of barcode values only from the document.

See also

Basic Usage Scenarios Implementation

Book Archiving

This scenario is used for processing books, magazines, newspapers to create an electronic library; for instance, when digitizing paper book collections for purposes of facilitating and expanding access to them and for their preservation.

Under this scenario, books, magazines, newspapers are converted into uneditable electronic copies containing all information from the source in searchable format. As a result of such processing, the resulting copies may be easily found in the electronic library using full-text search. During processing a special emphasis is placed on preserving the quality of the recognized text and restoring the structural elements of the document, especially the content.

To create an electronic copy, image files obtained by scanning or saved in electronic format first need to go through several processing stages, each of which has its own peculiarities for this scenario:

1. **Image preprocessing**
   Images obtained by scanning may require some preprocessing prior to recognition. For instance, the image of a scanned book may require straightening out of the lines skewed near the fold line, removal of the fold line darks, splitting of the image of a double-page spread into two separate pages.

2. **Recognition**
   To extract text data from a document, the document needs to be recognized. When recognizing books and newspapers, restoring logical structure of a document is of special importance. When processing a large volume of documents, simultaneous document processing may come in useful. In this case, during analysis and recognition the document load will be spread over processor cores, which makes it possible to speed up processing.

3. **Export**
   The recognized document is saved to a format used for storing data. The most convenient formats for storing documents in an electronic library are PDF, PDF/A, PDF and PDF/A with MRC. When saving to these formats, one may use a mode, under which the text is placed underneath a document image — this enables one to fully preserve the document formatting and provides a full-text search. The MRC settings allow significant reduction of a file size without loss of visual quality. Also when saving to the PDF format, one may customize security settings of the document protecting it from unauthorized viewing and printing.

Scenario implementation

Below is the detailed description of a recommended method of using ABBYY FineReader Engine 10 for the implementation of the above scenario. The proposed method employs the processing settings that are most suitable for the above scenario.

Step 1. Loading ABBYY FineReader Engine

To start your work with ABBYY FineReader Engine you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only ABBYY FineReader Engine externally creatable object.

To create the **Engine** object use the **GetEngineObject** exported function.

Sample code for the procedure of ABBYY FineReader Engine loading and initialization in C++ and Visual Basic:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;

void LoadFREngine()
{
        if( Engine != 0 ) {
                // Already loaded
                return;
        }
        // First step: load FREngine.dll
        if( libraryHandle == 0 ) {
                libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
                if( libraryHandle == 0 ) {
                        throw L"Error while loading ABBYY FineReader Engine";
                }
        }

        // Second step: obtain the Engine object
        typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR,
FREngine::IEngine** );
        GetEngineObjectFunc pGetEngineObject =
                ( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject"
);

        if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0,
&Engine ) != S_OK ) {
                UnloadFREngine();
                throw L"Error while loading ABBYY FineReader Engine";
        }
}
```

Visual Basic code

```
Public Engine As FREngine.Engine

Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                ByVal DeveloperSN As String, _
```

```
            ByVal Reserved1 As String, _
            ByVal Reserved2 As String, _
            EngineObj As FREngine.Engine) As Long

Sub Engine_Load(ByVal DeveloperSN As String)
      ' Visual Basic may load libraries from the current path only
      ChDir "Path to the folder with FREngine.dll"

      ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
      Dim DeveloperSN_WideChar As String
      DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)

      If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
            MsgBox "Error loading ABBYY FineReader Engine"
      End If
End Sub
```

Step 2. Loading settings for the above scenario

ABBYY FineReader Engine enables one to load all processing settings that are most suitable for this scenario using the **LoadPredefinedProfile** method of the **Engine** object. This method uses the name of a used settings profile as an input parameter. Please see Working with Profiles for more information.

ABBYY FineReader Engine supports 2 options of settings for this scenario. Both these profiles enable font style detection and full document synthesis:

- *BookArchiving_Accuracy*
  This profile optimizes document processing in order to ensure that the resulting document is of the highest quality possible.

- *BookArchiving_Speed*
  This profile optimizes the processing speed of the document creation process.
  ⚠**Important!** This profile requires the Fast Mode module available in the license.

Sample code for the procedure of profile loading in C++ and Visual Basic:

Visual C++ (COM) code

```
// Load a predefined profile
Engine->LoadPredefinedProfile( L"BookArchiving_Speed" );
```

Visual Basic code

```
' Load a predefined profile
Engine.LoadPredefinedProfile "BookArchiving_Speed"
```

If you wish to change processing settings, use appropriate parameter objects. Please see Additional optimization for specific tasks below for further information.

Step 3. Loading and preprocessing of images

ABBYY FineReader Engine provides the **FRDocument** object which allows processing multi-page documents. Use of this object allows you to preserve the logical organization of the document, retaining the original text and columns, fonts, styles, etc.

To load images of a single document and preprocess them, you should create the **FRDocument** object and add images into it. You may do one of the following:

- Create the **FRDocument** object using the **CreateFRDocumentFromImage** method of the **Engine** object. This method creates the **FRDocument** object and loads images from the specified file.

- Create the FRDocument object with the help of the **CreateFRDocument** method of the Engine object, then add images to the created FRDocument object from file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method of the FRDocument object).

Sample code for the procedure of image loading and preprocessing in C++ and Visual Basic:

Visual C++ (COM) code

```
// Open image file and create the FRDocument object
```

```
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage(
L"C:\\MyImage.tif", 0 );
```

Visual Basic code

```
' Open image file and create the FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif")
```

Step 4. Document recognition

To recognize a document, we suggest that the methods of the **FRDocument** object analysis and recognition be used. This object provides a whole array of methods for document analysis, recognition and synthesis. The most convenient method allowing document analysis, recognition and synthesis using just one method is the **Process** method. It also takes advantage simultaneous processing features of multiprocessor and multicore systems in the most efficient manner. However, you may also perform consecutive analysis, recognition and synthesis using **Analyze**, **Recognize** (or **AnalyzeAndRecognize**) and **Synthesize** methods.

Sample code for the procedure of document recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
// Analyze, recognize, and synthesize the document
// While the profile is loaded, you do not need to pass any additional parameters to
the processing method.
pFRDocument->Process( 0, 0, 0 );
```

Visual Basic code

```
' Analyze, recognize, and synthesize the document
' While the profile is loaded, you do not need to pass any additional parameters to the
processing method.
frDocument.Process
```

Step 5. Document export

To save a recognized document, you may use the **Export** method of the **FRDocument** object by assigning the **FileExportFormatEnum** constant as one of the parameters. In this scenario you can save the document, for example, to the PDF/A format with MRC in the PEM_ImageOnText export mode (the **TextExportMode** property of the **PDFExportParams** object). You may change the default parameters of export using the corresponding export object. Please see Additional optimization for specific tasks below for further information.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object. Use the **IFRDocument::Close** method.

Sample code for the procedure of document export to PDF/A in C++ and Visual Basic:

Visual C++ (COM) code

```
// Save a recognized document to an archive format (e.g. PDF/A)

 // Create a PDFExportParams object
 FREngine::IPDFExportParamsPtr params = Engine->CreatePDFExportParams();
 // Set necessary parameters
 params->PDFAComplianceMode = FREngine::PCM_Pdfa_1a;
 params->MRCMode = FREngine::MRC_Always;
 params->TextExportMode = FREngine::PEM_ImageOnText;

 // Use the parameters during export
 frDocument->Export(L"C:\\MyText.pdf", FREngine::FEF_PDFA, params);

 // Release the FRDocument object
 frDocument->Close();
```

Visual Basic code

```
' Save a recognized document to an archive format (e.g. PDF/A)

 ' Create a PDFExportParams object
 Dim params As FREngine.PDFExportParams
 Set params = Engine.CreatePDFExportParams
 ' Set necessary parameters
 params.PDFAComplianceMode = PCM_Pdfa_1a
 params.MRCMode = MRC_Always
 params.TextExportMode = PEM_ImageOnText
 ' Use the parameters during export
 frDocument.Export "C:\MyText.pdf", FEF_PDFA, params

 ' Release the FRDocument object
 frDocument.Close
```

Step 6. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine you need to unload the **Engine** object. To do this use the **DeinitializeEngine** exported function.

Sample code for the procedure of ABBYY FineReader Engine unloading and deinitialization in C++ and Visual Basic:

Visual C++ (COM) code

```
void UnloadFREngine()
{
        if( libraryHandle == 0 ) {
                return;
        }
        // Release Engine object
        Engine = 0;
        // Deinitialize FineReader Engine
        typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
        DeinitializeEngineFunc pDeinitializeEngine =
                ( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
        if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
                throw L"Error while unloading ABBYY FineReader Engine";
        }
        // Now it's safe to free the FREngine.dll library
        FreeLibrary( libraryHandle );
        libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization for specific tasks

Below is the overview of the Help topics containing additional information regarding customization of settings at different stages of document processing:

- **Scanning**
  - **Scanning**
    Description of the ABBYY FineReader Engine scenario for document scanning.
- **Opening and preprocessing**
  - **Image Preprocessing**
    Description of the ABBYY FineReader Engine scenario for preliminary preparation of images or enhancement of their visual quality.
- **Recognition**
  - **Tuning Analysis, Recognition, and Synthesis Parameters**
    Customization of document processing using objects of analysis, recognition and synthesis parameters.

- **PageProcessingParams Object**
  This object enables customization of analysis and recognition parameters. Using this object, you can indicate which image and text characteristics must be detected (inverted image, orientation, barcodes, recognition language, recognition error margin).

- **SynthesisParamsForPage Object**
  This object includes parameters responsible for restoration of a page formatting during synthesis.

- **SynthesisParamsForDocument Object**
  This object enables customization of document synthesis: restoration of its structure and formatting.

- **MultiProcessingParams Object**
  Simultaneous processing of documents may be useful when processing a large number of documents. In this case the document load will be spread over the processor cores during the analysis and recognition, which makes it possible to speed up processing. Reading modes (simultaneous or consecutive) are set using the **MultiProcessingMode** property. The **RecognitionProcessesCount** property controls the number of processes, which may be started.

- **Export**

  - **Tuning Export Parameters**
    Customization of the document export using objects of export parameters.

  - **PDFExportParams Object**
    This object enables customization of the PDF saving format parameters.

  - To customize the PDF (PDF/A) format export mode, use the **TextExportMode** property of the **PDFExportParams** object, and to customize MRC settings, use the **MRCMode** property.

See also

Basic Usage Scenarios Implementation

Text Extraction

This scenario is used to recognize the entire document text in order to prepare the document for search and extraction of useful data.

Such a scenario may serve as a basis for implementing more complex scenarios to extract vital data from documents, especially for automated input of paper document data into information systems and databases as well as for automated classification and indexation of documents in document management systems (e.g., inputting invoices into accounting software, inputting questionnaires into the CRM system).

This scenario enables extraction of the main text of the document, which contains all necessary information about the document. When using this scenario, main text data including texts on logos, seals and elements other than the main text, are extracted from the text.

To extract the main text of the document, image files obtained by scanning or saved in electronic format typically go through several processing stages, each of which has its own peculiarities in the content of this scenario:

1. **Image preprocessing**
   Scanned images may require some preprocessing prior to recognition, for example, if scanned documents contain background noise, skewed text, inverted colors, black margins, wrong orientation or resolution.

2. **Recognition**
   Recognition of images is performed using settings, which ensure that the maximum amount of text is extracted from a document image.

The text obtained as a result of processing may be used for searching vital data (however, information regarding the search for vital data lies outside the scope of this scenario). A certain algorithm is used to look up key words, e.g. names of form margins, tables, lines and table columns, signature and stamp fields, etc. Field containing important data are highlighted based on key words. These fields may be re-read using special recognition parameters depending on the type of data. The data found may be checked for consistency with the type and restrictions specified.

The data found may be saved to a database and an electronic uneditable copy of the paper document may be placed in the archive.

Scenario implementation

Below is the detailed description of the recommended method of using ABBYY FineReader Engine 10 for implementation of the above scenario. The proposed method uses processing settings that are most suitable for this scenario.

Step 1. Loading ABBYY FineReader Engine

To start your work with ABBYY FineReader Engine you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only ABBYY FineReader Engine externally creatable object.

To create the **Engine** object use the **GetEngineObject** exported function.

Sample code for the procedure of ABBYY FineReader Engine loading and initialization in C++ and Visual Basic:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
void LoadFREngine()
{
        if( Engine != 0 ) {
                // Already loaded
                return;
        }
        // First step: load FREngine.dll
        if( libraryHandle == 0 ) {
                libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
                if( libraryHandle == 0 ) {
                        throw L"Error while loading ABBYY FineReader Engine";
                }
        }
        // Second step: obtain the Engine object
        typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR,
FREngine::IEngine** );
        GetEngineObjectFunc pGetEngineObject =
                ( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject"
);
        if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0,
&Engine ) != S_OK ) {
                UnloadFREngine();
                throw L"Error while loading ABBYY FineReader Engine";
        }
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                 ByVal DeveloperSN As String, _
                 ByVal Reserved1 As String, _
                 ByVal Reserved2 As String, _
                 EngineObj As FREngine.Engine) As Long
Sub Engine_Load(ByVal DeveloperSN As String)
     ' Visual Basic may load libraries from the current path only
     ChDir "Path to the folder with FREngine.dll"
     ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
     Dim DeveloperSN_WideChar As String
     DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)
     If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
         MsgBox "Error loading ABBYY FineReader Engine"
     End If
End Sub
```

Step 2. Loading settings for the above scenario

The most suitable settings for this scenario may be selected in ABBYY FineReader Engine using the **LoadPredefinedProfile** method of the **Engine** object. This method uses the name of a used settings profile as an input parameter. Please see Working with Profiles for more information.

ABBYY FineReader Engine supports 2 options of settings for this scenario. Both these profiles enable detection of all text on an image, including small text areas of low quality (pictures and tables are not detected), while fonts and styles are not detected, and full document synthesis is not performed:

- *TextExtraction_Accuracy*
  This profile optimizes the text extraction process in order to ensure that the resulting document is of the highest quality possible.

- *TextExtraction_Speed*
  This profile optimizes the processing speed of the text extraction process: the processes of document analysis and recognition are sped up.

⚠**Important!** These profiles require the DA for Invoices module available in the license. The *TextExtraction_Speed* profile requires additionally the Fast Mode module.

Sample code for the procedure of profile loading in C++ and Visual Basic:

Visual C++ (COM) code

```
// Load a predefined profile
Engine->LoadPredefinedProfile( L"TextExtraction_Accuracy" );
```

Visual Basic code

```
' Load a predefined profile
Engine.LoadPredefinedProfile "TextExtraction_Accuracy"
```

If you wish to change processing settings, use appropriate parameter objects. Please see Additional optimization for specific tasks below for further information.

Step 3. Loading and preprocessing of images

ABBYY FineReader Engine provides the **FRDocument** object which allows processing multi-page documents.

To load images of a single document and preprocess them, you should create the **FRDocument** object and add images into it. You may do one of the following:

- Create the **FRDocument** object using the **CreateFRDocumentFromImage** method of the **Engine** object. This method creates the **FRDocument** object and loads images from the specified file.

- Create the FRDocument object with the help of the **CreateFRDocument** method of the Engine object, then add images to the created FRDocument object from file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method of the FRDocument object).

Sample code for the procedure of image loading and preprocessing in C++ and Visual Basic:

Visual C++ (COM) code

```
// Open image file and create the FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage(
L"C:\\MyImage.tif", 0 );
```

Visual Basic code

```
' Open image file and create the FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif")
```

Step 4. Document recognition

To recognize the document you should use analysis and recognition methods of the **FRDocument** object. This object provides a whole array of methods for document analysis and recognition. The most convenient method allowing document analysis, recognition and synthesis using just one method is the **Process** method. It also uses simultaneous processing features of multiprocessor and multicore systems in the most efficient manner. However, you may also carry out consecutive analysis, recognition and synthesis using **Analyze**, **Recognize** (or **AnalyzeAndRecognize**) methods.

Sample code for the procedure of document recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
// Analyze, recognize, and synthesize the document.
// While the profile is loaded, you do not need to pass any additional parameters to
the processing method.
pFRDocument->Process( 0, 0, 0 );
```

Visual Basic code

```
' Analyze, recognize, and synthesize the document.
' While the profile is loaded, you do not need to pass any additional parameters to the
processing method.
frDocument.Process
```

Step 5. Searching for vital information

During analysis ABBYY FineReader Engine selects image blocks containing text, tables, pictures, etc. In the course of recognition the blocks containing text data get filled with the recognized text.

In ABBYY FineReader Engine the **Layout** object serves as a storage for blocks and recognized text. The main scenario of document processing works with layout within the **FRDocument** object which represents processing document. To access a layout of a document page, use the **IFRPage::Layout** property.

To search for key words, you may view the recognized text using the **Text** object, which is accessible via the properties of the text, table or barcode blocks.

The vital data you have found may be saved or processed as required. Please see Additional optimization for specific tasks below for more detailed information.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object. Use the **IFRDocument::Close** method.

Step 6. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine you need to unload the **Engine** object. To do this use the **DeinitializeEngine** exported function.

Sample code for the procedure of ABBYY FineReader Engine unloading and deinitialization in C++ and Visual Basic:

Visual C++ (COM) code

```
void UnloadFREngine()
{
	if( libraryHandle == 0 ) {
		return;
	}
	// Release Engine object
	Engine = 0;
	// Deinitialize FineReader Engine
	typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
	DeinitializeEngineFunc pDeinitializeEngine =
		( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
	if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
		throw L"Error while unloading ABBYY FineReader Engine";
	}
	// Now it's safe to free the FREngine.dll library
	FreeLibrary( libraryHandle );
	libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
```

```
End Sub
```

Additional optimization for specific tasks

- **Scanning**
  - **Scanning**
    Description of the ABBYY FineReader Engine scenario for document scanning.
- **Opening and preprocessing**
  - **Image Preprocessing**
    Description of the ABBYY FineReader Engine scenario for preliminary preparation of images.
- **Recognition**
  - **Tuning Analysis, Recognition, and Synthesis Parameters**
    Customization of document processing using objects of analysis, recognition and synthesis parameters.
  - **PageProcessingParams Object**
    This object enables customization of analysis and recognition parameters. Using this object, you can indicate which image and text characteristics must be detected (inverted image, orientation, barcodes, recognition language, recognition error margin).
  - **SynthesisParamsForPage Object**
    This object includes parameters responsible for restoration of a page formatting during synthesis.
  - **SynthesisParamsForDocument Object**
    This object enables customization of document synthesis: restoration of its structure and formatting.
  - **MultiProcessingParams Object**
    Reading modes (simultaneous or consecutive) are set using the **MultiProcessingMode** property. The **RecognitionProcessesCount** property controls the number of processes, which may be started.
- **Searching for vital information**
  - **Working with Layout and Blocks**
    About page layout, block types, and working with them.
  - **Layout Object**
    This object's parameters provide access to the page layout and the recognized text after document recognition.
  - **Working with Text**
    Working with recognized text, paragraphs, words and symbols.
- **Re-reading of document using special parameters for specified data type**
  - **Field-Level Recognition**
    Description of scenario for recognizing short text segments.
- **Saving data**
  - To save recognized data, you may use the **Export** or **ExportPages** methods of the **FRDocument** object by assigning the **FileExportFormatEnum** constant as one of the parameters.
  - **Document Archiving**
    Description of the scenario for saving an electronic copy of document.

See also

Basic Usage Scenarios Implementation

Field-Level Recognition

In the case of field-level recognition, short text fragments are recognized in order to capture data from certain fields. Recognition quality is crucial in this scenario.

This scenario may also be used as part of more complex scenarios where meaningful data are to be extracted from documents (for example, to capture data from paper documents into information systems and databases or to automatically classify and index documents in Document Management Systems).

In this scenario, the system recognizes either several lines of text in only some of the fields or the entire text on a small image. The system computes a certainty rating for each recognized character. The certainty ratings can then be used when checking the recognition results. Additionally, the system may store multiple recognition variants for words and characters in the text, which may then be used in voting algorithms to improve the quality of recognition.

The processing of small text fragments in this scenario is in some ways different from the same steps in other scenarios:

1. **Image preprocessing**
   The images to be recognized may include markup and background noise, both of which may hamper recognition. For this reason, any unwanted markup and background noise are removed at this stage.

2. **Recognition**
   When recognizing small text fragments, the type of the data to be recognized is known in advance. Therefore, the quality of recognition may be improved through the use of external dictionaries, regular expressions, custom recognition languages and alphabets, and by imposing restrictions on the number of characters in a string. Text fields may contain both printed and handprinted text.

3. **Working with the recognized data**
   This scenario requires maximum recognition accuracy in order to keep data verification work to a minimum. The system may compute a certainty rating for each recognized word or character and provide multiple recognition variants from which several Engines may then choose the best candidate by applying voting algorithms.

Implementing the scenario

Below follows a detailed description of the recommended method of using of ABBYY FineReader Engine 10 in this scenario. The suggested method uses processing settings deemed most appropriate for this scenario.

Step 1. Loading ABBYY FineReader Engine

To start working with ABBYY FineReader Engine, you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only externally creatable object in ABBYY FineReader Engine.

To create the **Engine** object, use the **GetEngineObject** exported function. Sample C++ and Visual Basic code for loading and initializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
void LoadFREngine()
{
	if( Engine != 0 ) {
		// Already loaded
		return;
	}
	// First step: load FREngine.dll
	if( libraryHandle == 0 ) {
		libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0, 
LOAD_WITH_ALTERED_SEARCH_PATH );
		if( libraryHandle == 0 ) {
			throw L"Error while loading ABBYY FineReader Engine";
		}
	}
	// Second step: obtain the Engine object
	typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR, 
FREngine::IEngine** );
	GetEngineObjectFunc pGetEngineObject =
		( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject" 
);
```

```
	if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0, 
&Engine ) != S_OK ) {
		UnloadFREngine();
		throw L"Error while loading ABBYY FineReader Engine";
	}
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                ByVal DeveloperSN As String, _
                ByVal Reserved1 As String, _
                ByVal Reserved2 As String, _
                EngineObj As FREngine.Engine) As Long
Sub Engine_Load(ByVal DeveloperSN As String)
     ' Visual Basic may load libraries from the current path only
     ChDir "Path to the folder with FREngine.dll"
     ' this conversion is needed to pass a Unicode string as a DLL function parameter 
correctly
     Dim DeveloperSN_WideChar As String
     DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)
     If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
         MsgBox "Error loading ABBYY FineReader Engine"
     End If
End Sub
```

Step 2. Loading settings for the scenario

The most suitable settings can be selected by using the **LoadPredefinedProfile** method of the **Engine** object. This method accepts the name of the settings profile being used as an input parameter. The most suitable settings can be loaded by using the pre-defined profile named *FieldLevelRecognition*. For more about profiles, see Working with Profiles.

Sample code for the procedure of profile loading in C++ and Visual Basic:

Visual C++ (COM) code

```
// Load a predefined profile
Engine->LoadPredefinedProfile( L"FieldLevelRecognition" );
```

Visual Basic code

```
' Load a predefined profile
Engine.LoadPredefinedProfile "FieldLevelRecognition"
```

If you wish to change the settings used for processing, use the corresponding parameter objects. See the Additional optimization section below for more information.

Step 3. Loading and preprocessing images

ABBYY FineReader Engine provides a **FRDocument** object for processing multi-page documents. To load the images of a document and preprocess them, you should create the **FRDocument** object and add images into it. You can do one of the following:

- Create an **FRDocument** object using the **CreateFRDocumentFromImage** method of the **Engine** object. This method creates an **FRDocument** object and loads images from a specified file.
- Create an FRDocument object with the help of the **CreateFRDocument** method of the Engine object, then add the images into the created FRDocument object from a file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method of the FRDocument object).

Sample code for the procedure of image loading and preprocessing in C++ and Visual Basic:

Visual C++ (COM) code

```
// Open image file and create the FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage( 
L"C:\\MyImage.tif", 0 );
```

Visual Basic code

```
' Open image file and create the FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif")
```

Step 4. Recognition

In this scenario, methods which include synthesis have to be used for recognition. Only this approach will make the character attributes available for further operations after recognition. The most convenient method allowing document analysis, recognition and synthesis using just one method is the **Process** method. However, you may also perform consecutive analysis, recognition and synthesis using **Analyze**, **Recognize** (or **AnalyzeAndRecognize**) and **Synthesize** methods.

Also you may use user dictionaries and special languages during recognition. See the Additional optimization section below for more information.

Sample code for the procedure of document recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
// Analyze, recognize, and synthesize the document
// While the profile is loaded, you do not need to pass any additional parameters to
the processing method.
pFRDocument->Process( 0, 0, 0 );
```

Visual Basic code

```
' Analyze, recognize, and synthesize the document
' While the profile is loaded, you do not need to pass any additional parameters to the
processing method.
frDocument.Process
```

Step 5. Working with the recognized data

Use the **Text** object to access the recognized text fragment (you can get this object for a text block via the **ITextBlock::Text** property). Use the **Paragraphs** property to get the collection of paragraphs in the fragment and the **IParagraphs::Item** method to access the individual paragraphs. The **IParagraph::Text** property provides access to the recognized text of a paragraph.

You can use the **IParagraph::Words** to get the collection of words in a paragraph. Use the **IWords::Item** method to access individual words in the collection. The **IWord::Text** property returns the line that contains the recognized word. Use the **GetRecognitionVariants** method of the **Word** object or the **GetWordRecognitionVariants** method of the **Paragraph** object to get the recognition variants for a word.

The attributes of individual characters can be accessed via the **GetCharParams** method of the **Paragraphs** object. This method provides access to the **CharParams** object, which contains the parameters of the recognized character. The recognition variants for a character are accessible via the **ICharParams::CharacterRecognitionVariants** property.

For detailed information on working with text, see Working with Text. For information on using the Engine in voting algorithms, see Using Voting API.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object. Use the **IFRDocument::Close** method.

Step 6. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine, you need to unload the **Engine** object. To do this, use the **DeinitializeEngine** exported function.

Sample C++ and Visual Basic code for unloading and deinitializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
void UnloadFREngine()
{
	if( libraryHandle == 0 ) {
		return;
	}
	// Release Engine object
	Engine = 0;
	// Deinitialize FineReader Engine
	typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
	DeinitializeEngineFunc pDeinitializeEngine =
		( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
```

```
       if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
              throw L"Error while unloading ABBYY FineReader Engine";
       }
       // Now it's safe to free the FREngine.dll library
       FreeLibrary( libraryHandle );
       libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization

These are the sections of the help file where you can find additional information about setting up the parameters for the various processing stages:

- **Opening and preprocessing images**
    - **Image Preprocessing**
      Describes a scenario of using ABBYY FineReader Engine to preprocess images.
- **Recognition**
    - **Working with Languages**
      Using built-in and custom recognition languages.
    - **Working with Dictionaries**
      Using dictionaries to improve recognition quality.
    - **Recognizing Words with Spaces**
      Using dictionaries to recognize words with spaces (such as New York, etc.)
    - **Recognizing Handprinted Texts**
      Using ICR (Intelligent Character Recognition).
    - **Recognizing Checkmarks**
      Setting up recognition of checkmarks and groups of checkmarks.
- **Working with the recognized data**
    - **Working with Text**
      Working with the recognized text, paragraphs, words, and characters.
    - **Using Voting API**
      Working with words and character recognition alternatives.

See also

Basic Usage Scenarios Implementation

Barcode Recognition

In this scenario, ABBYY FineReader Engine is used to read barcodes. Barcodes may need to be read, for example, for purposes of automatic document separation, for processing documents by a Document Management System, or for indexing and classifying documents.

This scenario may be used as part of other scenarios. For example, documents scanned with high-speed production scanners may be separated by means of barcodes, or documents prepared for long-term storage may be placed into archiving Document Management Systems based on the values of their barcodes.

When extracting barcodes from texts, the system may detect all barcodes or only barcodes of a certain type with a certain value. The system may get the value of a barcode and calculate its check sum.

Recognized barcode values can be saved into formats most convenient for further processing, for example into TXT.

Implementing the scenario

Below follows a detailed description of the recommended method of using of ABBYY FineReader Engine 10 in this scenario. The suggested method uses processing settings deemed most appropriate for this scenario.

Step 1. Loading ABBYY FineReader Engine

To start working with ABBYY FineReader Engine, you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only externally creatable object in ABBYY FineReader Engine.

To create the **Engine** object, use the **GetEngineObject** exported function. Sample C++ and Visual Basic code for loading and initializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
void LoadFREngine()
{
        if( Engine != 0 ) {
                // Already loaded
                return;
        }
        // First step: load FREngine.dll
        if( libraryHandle == 0 ) {
                libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
                if( libraryHandle == 0 ) {
                        throw L"Error while loading ABBYY FineReader Engine";
                }
        }
        // Second step: obtain the Engine object
        typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR,
FREngine::IEngine** );
        GetEngineObjectFunc pGetEngineObject =
                ( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject"
);
        if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0,
&Engine ) != S_OK ) {
                UnloadFREngine();
                throw L"Error while loading ABBYY FineReader Engine";
        }
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                 ByVal DeveloperSN As String, _
                 ByVal Reserved1 As String, _
                 ByVal Reserved2 As String, _
                 EngineObj As FREngine.Engine) As Long
Sub Engine_Load(ByVal DeveloperSN As String)
     ' Visual Basic may load libraries from the current path only
     ChDir "Path to the folder with FREngine.dll"
     ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
```

```
     Dim DeveloperSN_WideChar As String
     DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)
     If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
         MsgBox "Error loading ABBYY FineReader Engine"
     End If
End Sub
```

Step 2. Loading settings for the scenario

The most suitable settings can be selected by using the **LoadPredefinedProfile** method of the **Engine** object. This method accepts the name of the settings profile being used as an input parameter. For more about profiles, see Working with Profiles.

The most suitable settings for the scenario can be loaded by using the predefined profile named *BarcodeRecognition*. This profile enables extracting only barcodes (texts, pictures, or tables are not detected).

⚠**Important!** This profile requires the Barcode Autolocation module available in the license.

Sample code for the procedure of profile loading in C++ and Visual Basic:

Visual C++ (COM) code

```
// Load a predefined profile
Engine->LoadPredefinedProfile( L"BarcodeRecognition" );
```

Visual Basic code

```
' Load a predefined profile
Engine.LoadPredefinedProfile "BarcodeRecognition"
```

If you wish to change the settings used for processing, use the corresponding parameter objects. See the Additional optimization section below for more information.

Step 3. Loading and preprocessing images

ABBYY FineReader Engine provides a **FRDocument** object for processing multi-page documents. To load the images of a document and preprocess them, you should create the **FRDocument** object and add images into it. You can do one of the following:

- Create an **FRDocument** object using the **CreateFRDocumentFromImage** method of the **Engine** object. This method creates an **FRDocument** object and loads images from a specified file.

- Create an FRDocument object with the help of the **CreateFRDocument** method of the Engine object, then add the images into the created FRDocument object from a file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method of the FRDocument object).

Sample code for the procedure of image loading and preprocessing in C++ and Visual Basic:

Visual C++ (COM) code

```
// Open image file and create the FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage(
L"C:\\MyImage.tif", 0 );
```

Visual Basic code

```
' Open image file and create the FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif")
```

Step 4. Extracting barcodes

If the *BarcodeRecognition* profile is loaded, you may use the **Process** method of the **FRDocument** object to extract only barcodes. In this case ABBYY FineReader Engine detects only blocks with barcodes. No other blocks are detected. The recognized barcode blocks can be accessed via the **Layout** object obtained by the above methods.

To read barcodes of a specific type, specify the appropriate parameters of the **BarcodeParams** object and pass the **BarcodeParams** object as a parameter of one of the above functions.

Sample code for the procedure of extracting barcodes in C++ and Visual Basic:

Visual C++ (COM) code

```
// Extract barcodes
// While the BarcodeRecognition profile is loaded, you do not need to pass any
additional parameters to the processing method.
pFRDocument->Process( 0, 0, 0 );
```

Visual Basic code

```
' Extract barcodes
' While the BarcodeRecognition profile is loaded, you do not need to pass any
additional parameters to the processing method.
frDocument.Process
```

Step 5. Exporting the recognized data

To save the values of the recognized barcodes to a file, you may use the **Export** method of the **FRDocument** object by assigning the **FileExportFormatEnum** constant as one of the parameters. This scenario can export, for example, to TXT. You may change the default parameters of export using the corresponding export object. Please see Additional optimization for specific tasks below for further information.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object. Use the **IFRDocument::Close** method.

Sample code for the procedure of document export to text format in C++ and Visual Basic:

Visual C++ (COM) code

```
// Save recognized barcodes to some format (e.g. TXT)
frDocument->Export( L"C:\\sample.txt", FREngine::FEF_TXT, 0 );
// Release the FRDocument object
frDocument->Close();
```

Visual Basic code

```
' Save recognized barcodes to some format (e.g. TXT)
frDocument.Export "C:\sample.txt", FEF_TXT, Nothing
' Release the FRDocument object
frDocument.Close
```

Step 6. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine, you need to unload the **Engine** object. To do this, use the **DeinitializeEngine** exported function.

Sample C++ and Visual Basic code for unloading and deinitializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
void UnloadFREngine()
{
        if( libraryHandle == 0 ) {
                return;
        }
        // Release Engine object
        Engine = 0;
        // Deinitialize FineReader Engine
        typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
        DeinitializeEngineFunc pDeinitializeEngine =
                ( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
        if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
                throw L"Error while unloading ABBYY FineReader Engine";
        }
        // Now it's safe to free the FREngine.dll library
        FreeLibrary( libraryHandle );
        libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization

These are the sections of the help file where you can find additional information about setting up the parameters for the various processing stages:

- **Opening and preprocessing images**
    - **Image Preprocessing**
      Describes a scenario of using ABBYY FineReader Engine to preprocess images.
- **Extracting and reading barcodes**
    - **BarcodeParams Object**
      This object allows you to set up the barcode analysis and reading parameters.
    - **Barcode Types**
      The list of barcodes supported in ABBYY FineReader Engine 10 and their brief descriptions.
    - **FRDocument Object**
      Apart from barcode values, you may need to extract other information contained in document. In this case you may wish to use the methods of the **FRDocument** object.
    - **PageProcessingParams Object**
      This object allows you to set up analysis and recognition parameters for the entire document. Using this object, you may specify whether barcode values should be recognized. To detect barcodes, set the value of the **DetectBarcodes** property to TRUE. Otherwise, barcodes will be identified as pictures. The barcode reading parameters are accessible via the **BarcodeParams** property.
    - **Working with Layout and Blocks**
      You can also mark barcode blocks manually and specify their analysis and reading parameters. This section provides detailed information on working with blocks.
- **Working with the recognized barcode values**
    - **BarcodeBlock Object**
      The **Text** and **BarcodeText** properties of this object contain the value of the barcode obtained through recognition. The other properties of this object can be used to get the type of the barcode, its orientation, and other parameters.
- **Export**
    - **Tuning Export Parameters**
      Setting up export with the objects of export parameters.
    - **TextExportParams Object**
      This object allows you to set up the saving of recognition results to TXT.

See also

Basic Usage Scenarios Implementation

Image Preprocessing

This scenario can be used to prepare images for further processing or to improve their visual quality (e.g. after scanning or prior to recognition).

This scenario may be used as part of other scenarios in the first stage of document processing, i.e. to prepare documents for recognition. Usage examples include creating uneditable document copies for archiving, getting editable versions of documents, and extracting meaningful data from documents.

In this scenario, image files are subjected to additional processing, such as:

- **Auto-detection of page orientation**
  Is very important for bulk input of images, when the direction in which document pages are scanned is unknown and can be different.

- **Automated image de-skewing**
  It is applied to scanned documents requiring the compensation for image skew. ABBYY FineReader Engine provides several methods for de-skewing images: with pairs of black squares, lines or lines of text.

- **Image despeckling**
  When scanning poor to medium quality documents, you may get very noisy images with lots of dots or speckles on them. These speckles, when they appear close to the letters or numbers, may affect the quality of OCR. The size of the speckles to be removed may be specified by the user. Despeckling can be applied to an image as well as to any individual zone of the image.

- **Splitting facing pages of scanned books into two separate images**
  It is used for scanning books as broadsides – for both left and right pages. The recognition quality is higher if the page is split into two, with each page corresponding to a single book page.

- **Lines straightening**
  When capturing text from scanned or photographed books, the text lines may be uneven and difficult to OCR. For accurate text recognition skew correction and straightening text lines should be performed.

- **Texture filtering**
  Texture filtering technology helps to filter out background "noise" such as color and texture, increasing accuracy for difficult-to-read documents such as newsprint, color documents, faxes, and copies.

- **Removing motion blur and ISO noise from digital photos**
  The system automatically identifies the typical defects commonly found in digital images, such as glare, ISO noise.

- **Clipping page margins**
  When need to improve the appearance of the images, you may want to clip some image areas, e.g. excess margins on digital photos.

Once preprocessed, the images are saved in user-defined formats or forwarded to further processing.

Implementing the scenario

Below follows a detailed description of the recommended method of using of ABBYY FineReader Engine 10 in this scenario.

Step 1. Loading ABBYY FineReader Engine

To start working with ABBYY FineReader Engine, you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only externally creatable object in ABBYY FineReader Engine.

To create the **Engine** object, use the **GetEngineObject** exported function. Sample C++ and Visual Basic code for loading and initializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
void LoadFREngine()
{
	if( Engine != 0 ) {
		// Already loaded
		return;
	}
	// First step: load FREngine.dll
	if( libraryHandle == 0 ) {
		libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
		if( libraryHandle == 0 ) {
			throw L"Error while loading ABBYY FineReader Engine";
		}
	}
	// Second step: obtain the Engine object
	typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR, 
FREngine::IEngine** );
	GetEngineObjectFunc pGetEngineObject = 
		( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject" 
);
	if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0, 
&Engine ) != S_OK ) {
		UnloadFREngine();
		throw L"Error while loading ABBYY FineReader Engine";
	}
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                 ByVal DeveloperSN As String, _
                 ByVal Reserved1 As String, _
                 ByVal Reserved2 As String, _
                 EngineObj As FREngine.Engine) As Long
Sub Engine_Load(ByVal DeveloperSN As String)
     ' Visual Basic may load libraries from the current path only
     ChDir "Path to the folder with FREngine.dll"
     ' this conversion is needed to pass a Unicode string as a DLL function parameter 
correctly
     Dim DeveloperSN_WideChar As String
     DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)
     If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
         MsgBox "Error loading ABBYY FineReader Engine"
     End If
End Sub
```

Step 2. Image preprocessing

The basic scenarios of image processing work with images within the **FRDocument** object which represents processing document.

To load images to the document, you may do one of the following:

- When creating the **FRDocument** object, use the **CreateFRDocumentFromImage** method of the **Engine** object.
- Add images to the created **FRDocument** object from file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method).

All these methods use as a parameter **PrepareImageMode** object which allows you to specify different parameters of image preprocessing. Create this object by calling the **IEngine::CreatePrepareImageMode** function, then change its properties as necessary and then pass it to a function that requires it.

Also you can modify loaded images. See the Additional optimization section below for more information.

See sample C++ and Visual Basic code for preprocessing images:

Visual C++ (COM) code

```
// Preprocess image
// Create a PrepareImageMode object
FREngine::IPrepareImageModePtr preParams = Engine->CreatePrepareImageMode();
// Set necessary parameters, e.g. CorrectSkewMode property
preParams->CorrectSkewMode = FREngine::CSM_CorrectSkewByBlackSquaresHorizontally;
// Open image file, preprocess it with the specified parameters, and create the 
FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocumentFromImage( 
L"C:\\MyImage.tif", preParams );
```

Visual Basic code

```
' Preprocess image
' Create a PrepareImageMode object
Dim preParams As FREngine.PrepareImageMode
Set preParams = Engine.CreatePrepareImageMode
' Set necessary parameters, e.g. CorrectSkewMode property
params.CorrectSkewMode = CSM_CorrectSkewByBlackSquaresHorizontally
' Open image file, preprocess it with the specified parameters, and create the
FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocumentFromImage("C:\MyImage.tif", preParams)
```

Step 3. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine, you need to unload the **Engine** object. To do this, use the **DeinitializeEngine** exported function.

Sample C++ and Visual Basic code for unloading and deinitializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
void UnloadFREngine()
{
       if( libraryHandle == 0 ) {
              return;
       }
       // Release Engine object
       Engine = 0;
       // Deinitialize FineReader Engine
       typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
       DeinitializeEngineFunc pDeinitializeEngine =
              ( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
       if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
              throw L"Error while unloading ABBYY FineReader Engine";
       }
       // Now it's safe to free the FREngine.dll library
       FreeLibrary( libraryHandle );
       libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization

These are the sections of the help file where you can find additional information about setting up the parameters for the various processing stages:

- **Image preprocessing**
  - **Working with Images**
    Working with images in ABBYY FineReader Engine and setting up image opening and preprocessing parameters.
  - **ImageDocument Object**
    The main object which provides access to images.
  - **PrepareImageMode Object**
    The parameters of this object affect image opening and preprocessing: skew correction, image inversion, mirroring, prepared image compression, resolution, rotation.

  - **ImageModification Object**
    Use this object for additional processing of source images (cropping, despeckling).

  - **DetectOrientation Method of the FRPage Object**
    This method detects text orientation on the image.

  - **CorrectSkew Method of the ImageDocument Object**
    Use this method to correct skew of the already opened image.

  - **RemoveGarbage Method of the ImageDocument Object**
    This method removes garbage (excess dots that are smaller than a certain size) from the image.

  - **FindPageSplitPosition Method of the FRPage Object**
    This method detects the direction of text on the image and finds the position where it can be split.

  - To straighten out distorted lines on an image, use the **IFRPage::RemoveGeometricalDistortions** or **IDocumentAnalyzer::RemoveGeometricalDistortions** method.

  - **SmoothImage Method of the ImageDocument Object**
    Allows you to smooth the image.

  - **RemoveColorObjects Method of the ImageDocument Object**
    With this method you can remove color objects from the whole image, or only from some parts of the image: specified region, its background, or only stamps and signatures in this region.

  - **SubtractColor Method of the ImageDocument Object**
    Removes the color with the specified hue and saturation from the image. The method is primary designed for filtering color on images of passports and certificates.

  - To preprocess digital photos, you may use the **IImageDocument::RemoveCameraBlur** and **IImageDocument::RemoveCameraNoise** methods.

- **Saving images**

  - **WriteToFile Method of the Image Object**
    Use this method to save images to a file in a format of your choice.

See also

Basic Usage Scenarios Implementation

Scanning

In this scenario, ABBYY FineReader Engine is used on a "scanning computer," which scans images and saves them as files.

This scenario may be used as part of other scenarios in the preliminary stage of document processing, i.e. for obtaining electronic versions of documents for further processing. Usage examples include scanning documents for archiving purposes, getting editable versions of documents, and extracting meaningful data from documents.

Paper documents are scanned and the images are saved in an electronic format, producing high-quality electronic versions of your printed documents.

Documents may go through the following processing stages:

1. **Scanning**
   Documents may be scanned via one of the two available scanning interfaces provided by scanners (TWAIN or WIA), by using ABBYY's own scanning interface, or without a scanning interface.

2. **Image preprocessing**
   Once scanned, the images may be preprocessed. Preprocessing includes despeckling, correction of distorted text lines, color inversion, removal of black margins, and correction of image orientation or resolution. Facing pages may be split into two separate images. Processed images may be saved in various image formats such as JPEG, TIFF, BMP.

Implementing the scenario

Below follows a detailed description of the recommended method of using of ABBYY FineReader Engine 10 in this scenario. Under the proposed implementation of the scenario, the image preparation phase is omitted. Please see Additional optimization for specific tasks below for the tips on image preparation implementation.

Step 1. Loading ABBYY FineReader Engine

To start working with ABBYY FineReader Engine, you need to create the **Engine** object. The **Engine** object is the top object in the hierarchy of the ABBYY FineReader Engine objects and is the only externally creatable object in ABBYY FineReader Engine.

To create the **Engine** object, use the **GetEngineObject** exported function. Sample C++ and Visual Basic code for loading and initializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
// HANDLE to FREngine.dll
static HMODULE libraryHandle = 0;
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
void LoadFREngine()
{
        if( Engine != 0 ) {
                // Already loaded
                return;
        }
        // First step: load FREngine.dll
        if( libraryHandle == 0 ) {
                libraryHandle = LoadLibraryEx( ::GetFreDllPathU(), 0,
LOAD_WITH_ALTERED_SEARCH_PATH );
                if( libraryHandle == 0 ) {
                        throw L"Error while loading ABBYY FineReader Engine";
                }
        }
        // Second step: obtain the Engine object
        typedef HRESULT ( STDAPICALLTYPE* GetEngineObjectFunc )( BSTR, BSTR, BSTR,
FREngine::IEngine** );
        GetEngineObjectFunc pGetEngineObject =
                ( GetEngineObjectFunc )GetProcAddress( libraryHandle, "GetEngineObject"
);
        if( pGetEngineObject == 0 || pGetEngineObject( ::GetFreDeveloperSN(), 0, 0,
&Engine ) != S_OK ) {
                UnloadFREngine();
                throw L"Error while loading ABBYY FineReader Engine";
        }
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                 ByVal DeveloperSN As String, _
                 ByVal Reserved1 As String, _
                 ByVal Reserved2 As String, _
                 EngineObj As FREngine.Engine) As Long
Sub Engine_Load(ByVal DeveloperSN As String)
     ' Visual Basic may load libraries from the current path only
     ChDir "Path to the folder with FREngine.dll"
     ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
     Dim DeveloperSN_WideChar As String
     DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)
     If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
         MsgBox "Error loading ABBYY FineReader Engine"
     End If
End Sub
```

Step 2. Scanning

ABBYY FineReader Engine offers a **ScanManager** object for scanning. Scanning and saving to file may be implemented with the **Scan** method of the **ScanManager** object.

See sample C++ and Visual Basic code for scanning:

Visual C++ (COM) code

```
// Create ScanManager object
 FREngine::IScanManagerPtr scanManager = Engine->CreateScanManager();


 // Specify the scan source
 FREngine::IStringsCollectionPtr sources = scanManager->ScanSources;
 _bstr_t scanner = sources->Item( 0 );

 // Create the ScanSourceSettings object
 FREngine::IScanSourceSettingsPtr scanSettings = scanManager-
>GetScanSourceSettings(scanner);

 // Set an interface type
 scanManager->ScanOptionsInterfaceType = FREngine::SOIT_None;

 // Tune the scanning options
 scanSettings->Resolution = 300;
 scanSettings->PictureMode = FREngine::SPM_Grayscale;

 // Set up the scanning options
 scanManager->PutScanSourceSettings(scanner, scanSettings);

 // The name of the folder in which scanned pages will be stored
 char scanFolder[MAX_PATH + 1];

 // Scan and save images into scanFolder folder

 FREngine::IStringsCollectionPtr scannedImages =
  scanManager->Scan( scanner, scanFolder, VARIANT_FALSE );
```

Visual Basic code

```
' Create the ScanManager object
 Dim ScanManager As FREngine.ScanManager
 Set ScanManager = Engine.CreateScanManager

 ' Specify the scan source
 Dim Scanner As String
 Scanner = ScanManager.ScanSources(0)

 ' Create the ScanSourceSettings object
 Dim ScanSettings As FREngine.ScanSourceSettings
 Set ScanSettings = ScanManager.ScanSourceSettings(Scanner)

 ' Set an interface type
 ScanManager.ScanOptionsInterfaceType = SOIT_None

 ' Tune the scanning options
 ScanSettings.Resolution = 300
 ScanSettings.PictureMode = SPM_Grayscale

 ' Set up the scanning options
 ScanManager.ScanSourceSettings(Scanner) = ScanSettings

 ' The name of the folder in which scanned pages will be stored
 Dim ScanFolder As String

 ' Scan and save images into scanFolder folder
 Dim ScannedImages As FREngine.StringsCollection
 Set ScannedImages = ScanManager.Scan(Scanner, ScanFolder, False)
```

Step 3. Unloading ABBYY FineReader Engine

After finishing your work with ABBYY FineReader Engine, you need to unload the **Engine** object. To do this, use the **DeinitializeEngine** exported function.

Sample C++ and Visual Basic code for unloading and deinitializing ABBYY FineReader Engine:

Visual C++ (COM) code

```
void UnloadFREngine()
{
        if( libraryHandle == 0 ) {
                return;
        }
        // Release Engine object
        Engine = 0;
        // Deinitialize FineReader Engine
        typedef HRESULT ( STDAPICALLTYPE* DeinitializeEngineFunc )();
        DeinitializeEngineFunc pDeinitializeEngine =
                ( DeinitializeEngineFunc )GetProcAddress( libraryHandle,
"DeinitializeEngine" );
        if( pDeinitializeEngine == 0 || pDeinitializeEngine() != S_OK ) {
                throw L"Error while unloading ABBYY FineReader Engine";
        }
        // Now it's safe to free the FREngine.dll library
        FreeLibrary( libraryHandle );
        libraryHandle = 0;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

Additional optimization

These are the sections of the help file where you can find additional information about setting up the parameters for the various processing stages:

- **Scanning**

    - **Tips for Document Scanning**
      Some tips on how to get good-quality scans of printed documents.

    - **Setting up Scanning Options**
      Using the ABBYY FineReader Engine scanning interfaces for scanning.

    - **ScanSourceSettings Object**
      Use this object to set up the scanning parameters.

- **Image preprocessing**

    - **Image Preprocessing**
      Describes a scenario of using ABBYY FineReader Engine to preprocess images or to improve their visual quality.

- **Document separation**

    - In this scenario, you may need to separate images into documents. ABBYY FineReader Engine 10 provides no means for automatic document separation. However, you can use ABBYY FlexiCapture Engine for the purpose. Documents may be separated based on the number of pages in each document or by using separator pages with barcodes. To implement barcode document separation, you can use the barcode recognition scenario.

See also

Basic Usage Scenarios Implementation

Advanced Techniques

The main programming aspects are presented in the following sections:

- Programming Aspects
  - Error Handling
  - Working with Properties
  - Working with Connectable Objects
  - Working with COM Interfaces from a Script Language
  - Using ABBYY FineReader Engine in Delphi

For tuning document processing parameters, see the following sections:

- Working with Profiles
- Tuning Analysis, Recognition, and Synthesis Parameters
- Tuning Export Parameters

For information on working with images, languages, layout, and recognized texts, see:

- Working with Images
- Working with Languages
- Working with Layout and Blocks
- Working with Text
- Working with Logical Structure of a Document
- Using Voting API
- Using Text Type Autodetection

For details on special cases, such as recognition of hieroglyphic languages, checkmarks, handprinted texts, and recognition with training, see the following sections:

- Recognizing Checkmarks
- Recognizing Handprinted Texts
- Recognizing Hieroglyphic Languages
- Recognizing with Training
- Training User Patterns

Information on working with dictionaries can be found here:

- Working with Dictionaries
- Working with Regular Expressions
- Recognizing Words with Spaces

Finally, scanning with ABBYY FineReader Engine is described in:

- Setting up Scanning Options

Programming Aspects

The ABBYY FineReader Engine application programming interface conforms to the COM standard and can be easily used in C/C++, Visual Basic, .Net, Delphi, or any other development tools supporting COM components. The Engine can be adapted for use in scripting languages like VBS, JS, Perl.

This section describes the main aspects of using ABBYY FineReader Engine in applications written in different programming languages and provides references to articles which discuss related topics.

Loading, initialization, and deinitialization

The **Engine** object is a singleton, so only one object of this type may be created in a single instance of the application that uses ABBYY FineReader Engine. Repeated attempts to create the **Engine** object will return the same object.

It is prohibited to initialize and deinitialize ABBYY FineReader Engine at the entry points of other DLLs, and also in constructors and destructors of static and global objects implemented in DLLs, because they are called at the DLL entry points. This restriction is due to the fact that the Win32 **LoadLibrary** and **FreeLibrary** functions are not re-entrant. A user should initialize and deinitialize ABBYY FineReader Engine elsewhere, for example, in **WinMain** function of an EXE module.

During initialization, ABBYY FineReader Engine will reset the LC_CTYPE setting of msvcrt.dll to the operating system defaults. This fact should be taken into account if your application depends upon the msvcrt.dll locale-dependent services.

For details see the description of the **GetEngineObject** function.

Note: .NET developers must make sure to specify [STAThread] (single-thread apartment model) as an attribute on the application main function, otherwise an error may occur:

```
[STAThread]
public static void Main()
{
  ...
}
```

The "Engine deinitialization failed" exception can be thrown during the deinitialization of the Engine object if not all of the objects which were created and used by the application have been deleted before the **Engine** object deinitialization. If you work with programming languages which do not have garbage collections (for example, C++), you must either use smart pointer classes (see the samples in C++ (COM)) or release objects that were created by creation methods when they are no longer needed. If all the objects have been deleted, the exception may be caused by the scavenger operation. If the application is developed in Visual Basic .Net, all objects with the *Nothing* value are not deleted, they are only marked for deletion. The exact moment when the garbage collector deletes the object is not known. Therefore, you should call the following methods before the deinitialization of the **Engine** object so that the garbage collector deletes the object:

```
GC.Collect()
GC.WaitForPendingFinalizers()
```

You can use the **StartLogging** method of the **Engine** object to get the list of objects that have not been deleted.

Also in this section

- **Error Handling**
  Information about error handling.

- **Working with Properties**
  The interfaces of ABBYY FineReader Engine objects have various properties and methods. The way the properties are handled in different languages is discussed in this article.

- **Working with Connectable Objects**
  Some of the objects in ABBYY FineReader Engine are so-called "connectable objects." Here you can find useful recommendations on working with such objects.

- **Working with COM Interfaces from a Scripting Language**
  The detailed description of how to use FineReader Engine in a scripting language.

- **Using ABBYY FineReader Engine in Delphi**
  The description of the initialization and deinitialization procedure in Delphi.

Error Handling

All ABBYY FineReader Engine interface methods and properties return a value of the HRESULT type. The HRESULT (for result handle) is a way of returning success, warning, and error values. HRESULTs are really not handles to anything; they are only 32-bit values with several fields encoded in the value. A zero result indicates success and a non-zero result indicates failure.

Note: Please do not handle exceptions that may be thrown during the work of the ABBYY FineReader Engine interface methods, because these exceptions are handled within ABBYY FineReader Engine.

If a method or property call was not successful, this method or property returns an HRESULT code that indicates the failure. Besides, it initializes the **IErrorInfo** object with a more detailed description of the error. Visual Basic users may access the HRESULT code through the *Number* property of the *Err* object. Other attributes of the *Err* object are initialized with the information from the **IErrorInfo**. Please refer to the documentation on COM for detailed description of error handling. The most general tips for it are as follows:

- **Visual Basic**. Error handling here is performed with the use of the **On Error** statement. If you do not use the **On Error Resume Next** statement anywhere in your code, any run-time error that occurs can cause an error message from the **IErrorInfo** object to be displayed and code execution stopped.

- **Raw C++**. ABBYY FineReader Engine interface methods and properties cannot throw exceptions but return HRESULTs. The most important means for handling these return codes are the SUCCEEDED and FAILED macros. They test the HRESULT value and deduce from it what was the result of the operation — success or failure. To get a pointer to the **IErrorInfo** object's interface, use the **GetErrorInfo** API function.

- **C++ with the Native COM support**. The Native COM support technology translates the HRESULT codes of interface functions into exceptions of a special type (**_com_error**) and automatically uses information from the **IErrorInfo** object. Thus, a sequence of ABBYY FineReader Engine methods may be enclosed by the statements:

```
try {
    ...
    } catch (_com_error e) {
    ...
    }
```

If any method or property that was called from inside the **try-catch** block returns an error code, this leads to throwing an exception, the code after the erroneous statement is not executed, and control is transferred to the code after the **catch** statement. Generally, error handling with the Native COM support may be performed in a way standard for any C++ code using functions that may throw exceptions.

See also

Standard Return Codes

Working with Properties

The interfaces of ABBYY FineReader Engine objects have various properties and methods. As Visual Basic users are familiar with the notion of property, we will discuss the way the properties are handled in C++.

For a C++ user, a property is a couple of methods (get and put for read-write properties) or a single get method (for read-only properties). However, the "Native COM support" featured by Microsoft Visual C++ makes the way the properties are handled more like the one used in Visual Basic. This is the way implied by the sense on the noun "property."

The ABBYY FineReader Engine properties may be of the following types:

Visual Basic type	**C++ type**
Boolean (with two values, True and False)	VARIANT_BOOL (with two values VARIANT_TRUE and VARIANT_FALSE)
Long	long
Double	double
String	BSTR, a pointer to Unicode string. Zero value specifies an empty srting.
Object	IUnknown-derived interface
Enumeration	

See the details of working with different types of properties below:

Working with simple properties

We will use the Boolean property to describe how simple properties are used. This property is described in the type library as follows:

```
interface IMyObject : IUnknown {
...
[propget]
HRESULT MyProperty([out, retval]VARIANT_BOOL* pVal);
 [propput]
HRESULT MyProperty([in]VARIANT_BOOL newVal);
...
};
```

A Visual Basic user handles this property as follows:

```
If MyObject.MyProperty <> True Then
   MyObject.MyProperty = True
End If
```

A C++ user, on the other hand, uses two methods to work with this property. These methods have get_ and put_ prefixes. The respective C++ code should look as follows:

```
IMyObject* pMyObject;
...
VARIANT_BOOL res;
pMyObject->get_MyProperty(&res);
if( res != VARIANT_TRUE )
   pMyObject->put_MyProperty(VARIANT_TRUE);
```

However, the Native COM support makes the procedure simpler, and the respective code should look as follows:

```
IMyObjectPtr pMyObject;
...
if(pMyObject->MyProperty != VARIANT_TRUE)
   pMyObject->MyProperty = VARIANT_TRUE
```

If the type library only defines the "get" method for a property, this property is called read-only. Its value cannot be changed by the user, it may only be accessed for "reading."

Working with string properties

Working with string properties is very similar to working with simple properties, but has its own specifics. A C++ user working with string properties must free the strings that are passed to set-methods, and also those that are returned by get-methods. However, this is done automatically in Visual Basic and in C++ with the Native COM support. Suppose **MyObject** also supports a string property called **Name**. This property is described in the type library as follows:

```
interface IMyObject : IUnknown {
...
[propget]
HRESULT Name([out, retval]BSTR* pVal);
 [propput]
HRESULT Name([in]BSTR newVal);
...
};
```

A C++ user works with this property like this:

```
IMyObject* pMyObject;
...
// "get" method
BSTR res;
pMyObject->get_Name(&res);
...
// Now free the string allocated in ABBYY FineReader Engine
::SysFreeString(res);
// "put" method
BSTR str = ::SysAllocString(L"New Name");
```

```
pMyObject->put_Name(str);
// Now free the string that we allocated
::SysFreeString(str);
```

For Visual Basic this may be rewritten as follows:

```
Dim obj As MyObject
Dim res As String
res = obj.Name
Dim str As String
str = "New Name"
obj.Name = str
```

Working with object properties

A C++ user will say that the parameters of "get" methods of "Object" properties are pointers to an object's interface pointer. As the interfaces of the objects are derived from IUnknown, they may be passed as IUnknown pointers to the properties or methods, which use objects of several types as input or output parameters (you may, however, get the interface you need by calling the QueryInterface method).

There are two different types of "put" methods for object properties — clear put, described by the **propput** keyword in the type library (the object is copied in this case); and put by reference, described by the **propputref** keyword in the type library (only a pointer to an existing object's interface is stored in the property in this case). A property may support only one of these put methods; most of ABBYY FineReader Engine object properties support clear put, while the **IRecognizerParams::TextLanguage** property supports put by reference. In Visual Basic, put by reference is performed using the Set statement, while clear put is performed without this keyword.

Suppose again the **MyObject** object supports **MyObjectProperty** property that refers to an object of **MyChildObject** type.

```
interface IMyObject : IUnknown {
...
[propget]
HRESULT MyObjectProperty([out, retval]IMyChildObject** pVal);
 [propputref]
HRESULT MyObjectProperty([in]IMyChildObject* newVal);
...
};
```

The same property is accessed as follows in Visual Basic:

```
Dim ChildObj As MyChildObject
Set ChildObj=MyObject.MyObjectProperty
' Do something with the object
...
' Clear put (If it were put by reference, we would write
' Set MyObject.MyObjectProperty=ChildObj)
MyObject.MyObjectProperty=ChildObj
```

A C++ user will writes this code as follows:

```
IMyObject* pMyObject;
...
IMyChildObject* pChildObj=0;
// get_ method may return 0 in certain cases
pMyObject->get_MyObjectProperty(&pChildObj);
// Do something with the object
...
pMyObject->put_MyObjectProperty(pChildObj);
...
pChildObj->Release();
```

Note that in C++ you should call the **Release** method for an object got via a property. The Native COM support calls **AddRef** and **Release** methods automatically using auto pointers.

⚠**Important!** If an object property refers to a child object of the object that exposes this property, a pointer to the child object's interface is valid until its parent object exists. An attempt to access a child object after its parent object is destroyed may result in an error.

Working with read-only object properties in raw C++

Certain ABBYY FineReader Engine objects (for example, **ILayout::Blocks**) have read-only object properties. Such properties cannot be changed directly in raw C++. If you want to change such a property, you need to pass a reference to the property object to a new variable, and then use this variable to change it. Below you can see a C++ sample for the **ILayout::Blocks** property which is represented by a read-only collection:

```
ILayout* pLayout = 0;
ILayoutBlocks* pLayoutBlocks = 0;
long blockIndex;
...
// The pLayoutBlocks variable receives a reference to the blocks collection from Layout
pLayout->get_Blocks( &pLayoutBlocks );
// Remove an element from the blocks collection
pLayoutBlocks->Remove( blockIndex );
```

Working with Connectable Objects

Some of the objects in ABBYY FineReader Engine are so-called "connectable objects". This means that they implement the **IConnectionPointContainer** interface. Connectable objects support communication between ABBYY FineReader Engine and its clients. Connectable objects in ABBYY FineReader Engine are:

- **DocumentAnalyzer**
- **Exporter**
- **ImageDocument**
- **ScanManager**
- **FRDocument**
- **FRPages**
- **FRPage**

Each of the ABBYY FineReader Engine connectable objects provides connection points of two types — one that uses a **dispatch** interface and one that uses the interface derived from *IUnknown*. The **dispatch** interface is designed for automatic use in Visual Basic and similar environments, while the vtbl-based interface is suitable for use in C++.

An ABBYY FineReader Engine client application that wants to receive notifications of certain events in ABBYY FineReader Engine should implement interfaces of a specific type and "advise" the objects implementing these interfaces to the corresponding connectable objects.

In Visual Basic, this is done by simply declaring the object *WithEvents* and implementing the corresponding methods of the callback interface. The procedure for Visual Basic is described in detail in the "ABBYY FineReader Engine API Reference" section for each connectable object.

Here is how you can connect an object on the client side to a notification source. We will use **DocumentAnalyzer** as an example.

```
class CDocumentAnalyzerEventsListener : public IDocumentAnalyzerEvents {
public:
...
    // Provide IUnknown methods simple implementation. They may also be
    // implemented by inheritance from some standard class with COM support
    ULONG AddRef();
    ULONG Release();
    HRESULT QueryInterface(REFIID iid, void** ppvObject)
    {
        if( ppvObject == 0 )
            return E_INVALIDARG;
        if( riid == __uuidof(IDocumentAnalyzerEvents) ) {
            *ppvObject = static_cast<IDocumentAnalyzerEvents*>( this );
        } else if( riid == IID_IUnknown ) {
            *ppvObject = static_cast<IUnknown*>( this );
        } else {
            *ppvObject = 0;
            return E_NOINTERFACE;
        }
        AddRef();
        return S_OK;
    }
    // Provide IDocumentAnalyzerEvents methods implementation
    HRESULT OnRegionProcessed(long, IRegion*, VARIANT_BOOL*);
    HRESULT OnProgress(long, VARIANT_BOOL*);
};
```

Thus we have the **CDocumentAnalyzerEventListener** class which may be used to receive notifications from the **DocumentAnalyzer** object. The following section of code advises this object to the notification source (error handling is omitted):

```
// Suppose that we have already created the DocumentAnalyzer object
IDocumentAnalyzer* da;
IConnectionPointContainer* pContainer=0;
da->QueryInterface(IID_IConnectionPointContainer, (void**)&pContainer);
IConnectionPoint* pPoint=0;
pContainer->FindConnectionPoint(__uuidof(IDocumentAnalyzerEvents),
                                &pPoint);
CDocumentAnalyzerEventsListener listener;
IUnknown* listenerUnknown=0;
listener.QueryInterface(IID_IUnknown, (void**)&listenerUnknown);
// A variable to store the cookie returned from the IConnectionPoint::Advise method
DWORD cookie;
pPoint->Advise(listenerUnknown, &cookie);
...
// After notification, the listener is no longer needed and should be unadvised
pPoint->Unadvise(cookie);
```

Refer to the documentation on COM for a more detailed description of connectable objects.

See also

See sample: EventsHandling

Working with COM Interfaces from a Scripting Language

ABBYY FineReader Engine 10 supports dynamic binding in COM interfaces:

- Almost all ABBYY FineReader Engine 10 interfaces are derived from **IDispatch** (the exceptions are some of the callback interfaces implemented on the client side).
- Scripting languages (for example, VBScript and JScript) support only dynamic binding. Therefore, due to inheriting such interfaces from IDispatch the use of ABBYY FineReader Engine API from these languages requires minimal additional effort: only the **Engine** object cannot be created by using the COM method **CoCreateObject**. So you will have to create an additional object for creating the **Engine** object. See the method for creating this additional object in the FRECOMWrapper sample.
- The other objects created by the methods of the **Engine** object named "Create..." or "Load..." can now be created and used directly from the scripting language.

The **FRECOMWrapper** sample code produces FREngineWrap.dll that can be used for getting the ABBYY FineReader Engine object from a scripting language such as VBScript or JavaScript. The FREngineWrap.dll library has the **FRELoader** class with the **Load** method which loads the ABBYY FineReader Engine library and with the **EngineObject** property containing a pointer to the **Engine** object.

Note: In order to create the FREngineWrap.dll library, compile the **FRECOMWrapper** sample located in **\Samples\Visual C++ (COM)\FRECOMWrapper**.

For example, you can create the **Engine** object by using the JavaScript **ActiveXObject** function.

```
// create the FRELoader object
FRELoader = new ActiveXObject( "FREngineWrap.FRELoader" );
...
// load the library
FRELoader.Load( );
...
// get the Engine object
var Engine = FRELoader.EngineObject;
```

In VBScript, use the **CreateObject** method:

```
' create FRELoader object
Set FRELoader = CreateObject( "FREngineWrap.FRELoader" )
...
' load the library
FRELoader.Load
...
' get the Engine object
Set Engine = FRELoader.EngineObject
```

In Perl, use the **CreateObject** method:

```
# create the FRELoader object
$FRELoader = $WScript->CreateObject( 'FREngineWrap.FRELoader' );
...
# load the library
$FRELoader->Load( );
...
# get the Engine object
my $Engine = $FRELoader->{EngineObject};
```

An example illustrating the use of the FREngineWrap.dll library can be found in **\Samples\Visual C++ (COM)\FRECOMWrapper\TestScripts**.

See also

Description of ABBYY FineReader Engine Samples

Using ABBYY FineReader Engine in Delphi

This section deals with certain peculiarities of using ABBYY FineReader Engine 10 in Delphi.

Creating the Object Pascal Wrapper Unit

In order to use ABBYY FineReader Engine 10 in Delphi, it is necessary to create the Object Pascal wrapper unit for the type library (the FREngine_TLB.pas file). Do the following:

1. Run command prompt (cmd.exe) and go to the folder where the ABBYY FineReader Engine 10 type library (FREngine.tlb) is located.

2. Run the tlibimp utility with the following parameters:
   Delphi 5.0:
   tlibimp -O- -E- -C- -P+ -T+ FREngine.tlb

   Delphi 6.0 and 7.0:
   tlibimp -O- -Cd- -C- -P+ -Pt+ FREngine.tlb

   Delphi 2010:
   tlibimp -O- -Cd- -C- -P+ -Pt+ FREngine.tlb

   This will generate the FREngine_TLB.pas file.

3. Add FREngine_TLB.pas to your project.

You need to re-generate FREngine_TLB.pas each time you receive an updated version of ABBYY FineReader Engine 10.

Deinitialization of the Engine Object

If not all the objects which were created and used by the application have been deleted before the deinitialization of the **Engine** object, the "Engine deinitialization failed" exception is thrown. If all the objects have been deleted, the exception may be caused by the scavenger operation. In Delphi all objects with the *nil* value are deleted only after exiting the procedure in which the objects were declared. Therefore, the entire ABBYY FineReader Engine code must be inserted into a separate procedure, and this procedure must be called before the deinitialization of the **Engine** object.

Note: You can use the StartLogging method of the Engine object to get the list of objects that have not been deleted.

Working with Profiles

ABBYY FineReader Engine supports numerous parameters which allow the user to fine-tune the Engine. The user can specify parameters for image preprocessing, analysis, recognition, synthesis, and export to receive the optimal speed and quality of processing. For example, if the application will export recognition results to TXT, then page layout is not relevant and many layout-related properties can be disabled.

When new objects are created, either directly with the help of the creation methods of the **Engine** object or indirectly, the properties of newly created objects are usually set to reasonable defaults (for more information about the default value of a property, see the description of the corresponding property). But default values are not always optimal for all usage scenarios. You may need to change these properties in some cases. This can be done either via the API or with the help of a profile. A profile contains a list of new default values for object properties.

Predefined profiles

ABBYY FineReader Engine provides a set of predefined profiles which are designed for the main usage scenarios. The settings provided in these profiles are most suitable in the corresponding situations. Besides, most of the profiles come in two forms: with the settings optimized for the best quality of the resulting document or with the settings optimized for the highest speed of processing. Below is a list of available predefined profiles:

- *DocumentConversion_Accuracy* — for converting documents into editable formats, optimized for accuracy
- *DocumentConversion_Speed* — for converting documents into editable formats, optimized for speed
- *DocumentArchiving_Accuracy* — for creating an electronic archive, optimized for accuracy
- *DocumentArchiving_Speed* — for creating an electronic archive, optimized for speed
- *BookArchiving_Accuracy* — for creating an electronic library, optimized for accuracy
- *BookArchiving_Speed* — for creating an electronic library, optimized for speed
- *TextExtraction_Accuracy* — for extracting text from documents, optimized for accuracy
- *TextExtraction_Speed* — for extracting text from documents, optimized for speed
- *FieldLevelRecognition* — for recognizing short text fragments
- *BarcodeRecognition* — for extracting barcodes
- *Version9Compatibility* — provided for compatibility, sets the processing parameters to the default values of ABBYY FineReader Engine 9.0.

Note: You can view the list of settings provided by these profiles in the descriptions of the corresponding scenarios.

Important! The profiles may require additional modules available in the license. See details in the descriptions of corresponding scenarios.

The settings provided with these profiles can be loaded using the **LoadPredefinedProfile** method of the **Engine** object. After the profile is loaded, newly created objects will have the new default values specified in the profile.

User profiles

You can also create your own profile file. The syntax of a profile file is similar to that of *.ini files. The sections contain the names of the objects whose properties are to be re-specified, and the keys contain the properties with their new values. The special section called

UserData can contain any user-defined keys. The values of Boolean properties are represented by the strings "true" or "false," while enumeration properties are represented by corresponding constants, for example:

```
[PrepareImageMode]
DiscardColorImage = true
[PDFExportParams]
TextExportMode = PEM_ImageOnText
[RecognizerParams]
TextLanguage = English,Russian
```

The **LoadProfile** method of the **Engine** object allows you to load a user profile file. After this file is loaded, newly created objects will have the new default values specified in the file. Loading parameters from a profile is similar to specifying the corresponding properties in the program code, but it simplifies the logic and data in the application.

A profile file can be used to re-specify all the properties of the following objects:

- **PrepareImageMode**
- **ImageProcessingParams**
- **PageProcessingParams**
- **PageAnalysisParams**
- **TableAnalysisParams**
- **BarcodeParams**
- **ObjectsExtractionParams**
- **OrientationDetectionParams**
- **RecognizerParams**, except the **PossibleTextTypes** property
- **SynthesisParamsForPage**
- **SynthesisParamsForDocument**
- **DocumentStructureDetectionParams**
- **FontFormattingDetectionParams**
- **RTFExportParams**
- **HTMLExportParams**
- **XLExportParams**
- **TextExportParams**
- **PPTExportParams**
- **XMLExportParams**
- **PDFExportParams**
- **PDFExportParamsOld**, except the **EncryptionInfo** property
- **PDFAExportParamsOld**
- **PDFMRCParams**

If an empty string is passed to **IEngine::LoadProfile**, the standard default values will be used.

The correctness of the new values of the properties and their conformity to the license are checked when a corresponding object is created.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
Tuning Export Parameters

Tuning Analysis, Recognition, and Synthesis Parameters

Document processing in ABBYY FineReader Engine consists of several steps: analysis, recognition, synthesis, and export. This section concerns with the parameters of analysis, recognition, and synthesis. For details about export parameters, see Tuning Export Parameters.

During analysis ABBYY FineReader Engine finds certain areas on the document pages. These areas are called "blocks." Each block has its type. Then the parts of the image that lie inside the blocks are recognized in the way defined by the block type. Finally, the text and background colors, fonts and other formatting elements are detected (this process is called "synthesis").

Before processing, you can set the parameters of analysis, recognition, and synthesis with the help of the parameters objects. Pointers to these objects can be passed to the processing methods as input parameters, and thus affect the results of processing. The following ABBYY FineReader Engine objects provide analysis, recognition, and synthesis methods: **FRDocument**, **FRPage**, **Engine**, **DocumentAnalyzer**.

The processes of analysis, recognition, and synthesis can also be tuned using profiles. See Working with Profiles for details.

Parameters of analysis and recognition

To set the parameters of analysis and recognition, you need to tune the properties of the **PageProcessingParams** object. The **PageProcessingParams** object is the parent for a group of ABBYY FineReader Engine objects which set up the page processing parameters. For analysis and recognition, the following child objects of the **PageProcessingParams** object are used:

- **PageAnalysisParams** — affects the page layout analysis
- **RecognizerParams** — contains the general page recognition parameters
- **BarcodeParams** — contains a set of properties specific to barcode recognition
- **ObjectsExtractionParams** — contains the parameters used for detecting additional objects (e.g. garbage, texture, small text areas of low quality, etc.) on an image before recognition
- **OrientationDetectionParams** — affects the page orientation detection

Parameters of synthesis

The process of synthesis may be divided into two stages: page synthesis and document synthesis. During page synthesis, only hyperlinks and text and background colors are detected. Font styles and formatting is detected during document synthesis. A set of Engine API objects become meaningful only after document synthesis — these are so-called document synthesis objects, which provide access to the logical structure of the document and formatting attributes, including headers, footers, page numbers, fonts, styles, and more.

The parameters of synthesis can be set with the help of the following objects:

- **SynthesisParamsForPage**. This object is used for setting up the parameters of page synthesis.
- **SynthesisParamsForDocument**. This object is used for setting up the parameters of document synthesis.

Tuning document processing

A step-by-step procedure that uses the parameter objects mentioned above should look like this:

1. Create a **PageProcessingParams** object with the help of the **CreatePageProcessingParams** method of the **Engine** object.
2. Set up the necessary properties of the sub-objects of the **PageProcessingParams** object. You do not need to set up *all* the properties of all the sub-objects, as on creation they are initialized with reasonable defaults. You only have to tune up those of the properties that you want to have values other than default ones.
   When you are setting up the parameters to be used by the layout analysis functions, do not forget to set the correct values of the properties of the sub-objects of the **PageProcessingParams** that affect recognition. This is recommended, because all these parameters are copied into the blocks that are created during the layout analysis and are then used for recognition, and also because analysis of certain parts of the image may involve recognition.
3. Create **SynthesisParamsForPage** and/or **SynthesisParamsForDocument** objects.
4. Set up the necessary properties of these objects. You do not need to set up *all* the properties of all the objects and sub-objects, as on creation they are initialized with reasonable defaults. You only have to tune up those of the properties that you want to have values other than the default ones.
5. You can pass these parameters to one of the processing methods of the **FRDocument**, **FRPage**, **Engine**, and **DocumentAnalyzer** objects.
   To recognize a document, we suggest that the processing methods of the **FRDocument** object be used. This object provides a whole array of methods for document analysis, recognition, and synthesis. The most convenient method allowing document analysis, recognition, and synthesis using just one method is the **Process** method. It also uses simultaneous processing features of multiprocessor and multicore systems in the most efficient manner. However, you can also carry out consecutive analysis, recognition, and synthesis using **Analyze**, **Recognize** (or **AnalyzeAndRecognize**), and **Synthesize** methods.

Sample code for setting the parameters of analysis, recognition, and synthesis.

Visual C++ (COM) code

```
FREngine::IEnginePtr Engine;
FREngine::IFRDocumentPtr frDocument;
// Create a PageProcessingParams object
```

```
FREngine::IPageProcessingParamsPtr processingParams = Engine-
>CreatePageProcessingParams();
// Set necessary parameters (do not forget to set the right recognition language)
processingParams->RecognizerParams->SetPredefinedTextLanguage( L"Russian,English");
processingParams->DetectOrientation = VARIANT_TRUE;
// Create a SynthesisParamsForDocument object
FREngine::ISynthesisParamsForDocumentPtr synthesisParams = Engine-
>CreateSynthesisParamsForDocument();
// Set necessary parameters
synthesisParams->SaveRecognitionInfo = VARIANT_FALSE;
// Recognize document with the specified parameters
frDocument->Process( processingParams, 0, synthesisParams );
```

Visual Basic code

```
Dim Engine As FREngine.Engine
Dim frDocument As FREngine.frDocument
' Create a PageProcessingParams object
Dim processingParams As FREngine.PageProcessingParams
Set processingParams = Engine.CreatePageProcessingParams
' Set necessary parameters (do not forget to set the right recognition language)
processingParams.RecognizerParams.SetPredefinedTextLanguage("Russian,English")
processingParams.DetectOrientation = True
' Create a SynthesisParamsForDocument object
Dim synthesisParams As FREngine.SynthesisParamsForDocument
Set synthesisParams = Engine.CreateSynthesisParamsForDocument
' Set necessary parameters
synthesisParams.SaveRecognitionInfo = False
' Recognize document with the specified parameters
frDocument.Process processingParams, Nothing, synthesisParams
```

See also

Working with Profiles
Tuning Export Parameters

Tuning Export Parameters

During export, recognized documents are saved in files in suitable formats. ABBYY FineReader Engine has a group of objects which provide tools for tuning different export parameters. Pointers to these objects can be passed to the export methods as input parameters, and thus affect the results of export. The following ABBYY FineReader Engine objects provide export methods: **FRDocument**, **FRPage**, **Engine**, **Exporter**.

For each supported external format, there is a corresponding export parameter object. These are:

- **RTFExportParams** for RTF, DOC, and DOCX formats
- **TextExportParams** for TXT and CSV formats
- **XLExportParams** for XLS and XLSX formats
- **HTMLExportParams** for HTML format
- **PDFExportParams** for PDF and PDF/A format
- **XMLExportParams** for XML format
- **PPTExportParams** for PPTX format

Export processes can also be tuned using profiles. See Working with Profiles for details.

The export procedure

A step-by-step procedure that uses objects of this group should look like this:

1. Create an export parameter object that corresponds to the external format in which you are going to save your text. Use the corresponding creation method of the **Engine** object.

2. Set up the necessary properties of the object you created. You do not need to set up *all* the properties of the export parameter object, as on creation they are initialized with reasonable defaults. You only have to tune up those of the properties that you want to have values other than default ones.

3. Pass it to one of the export methods of the **FRDocument**, **FRPage**, **Engine**, **Exporter** objects.

Sample code that uses the **RTFExportParams** object in C++ and Visual Basic:

Visual C++ (COM) code

```
FREngine::IEnginePtr Engine;
FREngine::IFRDocumentPtr frDocument;
// Create export parameter object
FREngine::IRTFExportParamsPtr params = Engine->CreateRTFExportParams();
// Tune export parameters
params->KeepLines = VARIANT_TRUE;
// Now export text into a file
frDocument->Export( L"C:\\myFile.rtf", FREngine::FEF_RTF, params );
```

Visual Basic code

```
Dim Engine As FREngine.Engine
Dim FRDocument As FREngine.FRDocument
' Create export parameter object
Dim Params As FREngine.RTFExportParams
Set Params = Engine.CreateRTFExportParams
' Tune export parameters
Params.KeepLines = True
' Now export text into a file
FRDocument.Export "C:\myFile.rtf", FEF_RTF, Params
```

Export to PDF and PDF/A formats

ABBYY FineReader Engine allows you to tune export to PDF and PDF/A formats in an even more convenient way. It provides the **PDFExportParams** object, which allows you to tune export with only a few parameters. You do not need to set all the parameters of the obsolete **PDFExportParamsOld** or **PDFAExportParamsOld** objects, but simply set the parameters of the **PDFExportParams** object for your task. For example, using only one **IPDFExportParams::Scenario** property you can optimize your PDF for quality and size.

The procedure which uses the **PDFExportParams** object is as follows:

1. Create a **PDFExportParams** object using the **CreatePDFExportParams** method of the **Engine** object.
2. Set the necessary parameters of the **PDFExportParams** object:
    - the scenario of export, which optimizes export for some parameters: quality, size of the file, or/and speed of export (the **Scenario** property)
    - the format of export: PDF, PDF/A-1a, or PDF/A-1b (the **PDFAComplianceMode** property)
    - the mode of recognized text export: text and pictures only, text over the page image, text under the page image, page image only (the **TextExportMode** property)
    - set other parameters, if necessary
3. Pass the object of export parameters to one of the export methods of the **FRDocument**, **FRPage**, **Engine**, **Exporter** objects.

Sample code in C++ and Visual Basic:

Visual C++ (COM) code

```
FREngine::IEnginePtr Engine;
 FREngine::IFRDocumentPtr frDocument;

 // Create a PDFExportParams object
 FREngine::IPDFExportParamsPtr params = Engine->CreatePDFExportParams();
 // Set necessary parameters
 params->Scenario = FREngine::PES_MaxSpeed;
 params->TextExportMode = FREngine::PEM_ImageOnText;

 // Use the parameters during export
 frDocument->Export(L"C:\\MyText.pdf", FREngine::FEF_PDF, params);
```

Visual Basic code

```
Dim Engine As FREngine.Engine
 Dim FRDocument As FREngine.FRDocument

 ' Create a PDFExportParams object
 Dim params As FREngine.PDFExportParams
 Set params = Engine.CreatePDFExportParams
 ' Set necessary parameters
 params.Scenario = PES_MaxSpeed
 params.TextExportMode = PEM_ImageOnText

 ' Use the parameters during export
 frDocument.Export "C:\MyText.pdf", FEF_PDF, params
```

See also

Export Formats
Working with Profiles
Tuning Analysis, Recognition, and Synthesis Parameters

Working with Images

The basic scenarios of image processing work with images within the **FRDocument** object, which represents the document being processed.

Image opening

To load images into the document, do one of the following:

- When creating the **FRDocument** object, use the **CreateFRDocumentFromImage** method of the **Engine** object.
- Add images to the created **FRDocument** object from a file (use the **AddImageFile**, **AddImageFileWithPassword**, or **AddImageFileWithPasswordCallback** method).

All these methods use the **PrepareImageMode** object as a parameter, which allows you to specify different parameters of image preprocessing. Create this object by calling the **IEngine::CreatePrepareImageMode** function, then change its properties as necessary, and then pass it to a function that requires it.

ImageDocument structure

Pages of the document provide access to the images via the **IFRPage::ImageDocument** property. Each open image in ABBYY FineReader Engine, each image in the so-called "internal format," is represented by the **ImageDocument** object, which includes three image planes. One image plane corresponds to one **Image** object:

- *Black-and-white* plane. It is the black-and-white copy of the source image. The copy is deskewed or non-deskewed, depending on the internal file preparation mode (see the description of the **IPrepareImageMode::CorrectSkewMode** property).
- *Color* plane. This is the color or gray copy of the source image. The copy is deskewed or non-deskewed, depending on the internal file preparation mode (see the description of the **IPrepareImageMode::CorrectSkewMode** object). If the source image was black-and-white, this page is the same as the "black-and-white" plane.
- *Preview*. A small color image used for displaying a preview image in the user interface. It may be or may not be available in the file in the internal format. The availability of this preview image depends on the internal file preparation mode (see the description of the **IPrepareImageMode::CreatePreview** property).

Each image plane of the above-mentioned set is characterized by its own size and resolution. The size and resolution of black-and-white and color images are the same. Since image documents may consist of deskewed images, the **ImageDocument** object has a set of coordinate conversion functions. Use the **IImageDocument::ConvertCoordinates** function to convert pixel coordinates from one image plane to another. The coordinates of pixels on the black-and-white image plane and the color image plane are the same. Remember that the recognition functions use the page received after image preparation (therefore, the page may be deskewed). So, it is this page size and resolution that should be used when you create your **Layout** objects, otherwise the ABBYY FineReader Engine export functions may not work correctly.

You can add an already created **ImageDocument** object to a document using the **AddImage** method of the **FRDocument** object.

Image modification

ABBYY FineReader Engine provides functionality for image editing (inversion, stretch, etc.) via the **ImageModification** object. To perform modification, do the following:

1. Create an **ImageModification** object with the help of the **CreateImageModification** method of the **Engine** object.
2. Specify the necessary parameters.
3. Call the **IImageDocument::Modify** method that takes the **ImageModification** object as an input parameter. The actual change takes place only when you call this method.
4. Save the changes using the **IImageDocument::SaveModified** method.
   ⚠**Important!** Modifications to the image are not saved until the **IImageDocument::SaveModified** method is called. If the **ImageDocument** object is released before a call to this method, the modifications are not saved.

Image saving

You can save the current image plane into an image file in a specified format using the **WriteToFile** method of the **Image** object.

Note that though the **ImageDocument** object provides a set of saving methods (**SaveTo**, **SaveToFile**, **SaveToMemory**), these methods cannot be used for saving an image in an external format. These methods save the contents of the **ImageDocument** object in the ABBYY FineReader Engine internal format, which cannot be viewed in any external program.

ABBYY FineReader Engine also provides functionality for saving several images into a single image file. To save multi-page image file, use the **MultipageImageWriter** object:

1. Create a **MultipageImageWriter** object using the **CreateMultipageImageWriter** method of the **Engine** object.
2. Add images to the end of the multi-page image file using the **AddPage** method of the **MultipageImageWriter** object. Each image is added as a single page.
3. Before the newly created image file can be used, all the references to the **MultipageImageWriter** object must be released.

See also

Supported Image Formats
Tips for Document Scanning
Tips for Taking Photos

Working with Languages

One of the main recognition parameters is the language which is used during recognition. It is important to set the right language before analysis and recognition. Recognition language can be easily specified with the help of the **IRecognizerParams::SetPredefinedTextLanguage** method. This method effects the **IRecognizerParams::TextLanguage** property. By default, this parameter is initialized with the English recognition language.

Below you can find useful information about the languages supported in ABBYY FineReader Engine by default and objects that provide advanced functionality for working with recognition languages.

Predefined languages

ABBYY FineReader Engine provides a set of languages supported by default. These languages are called "predefined languages." The collection of available predefined languages represented by the **PredefinedLanguages** object is accessible via the **PredefinedLanguages** property of the **Engine** object. It is a collection of **PredefinedLanguage** objects.

The predefined languages are identified by their internal names. You may directly specify a recognition language by the name of the corresponding predefined language via the **IRecognizerParams::SetPredefinedTextLanguage** method. For the list of the internal names of the predefined languages see Predefined Languages in ABBYY FineReader Engine.

Recognition language for a text

The language which is used during recognition is represented by the **TextLanguage** object. The **RecognizerParams** object that specifies the recognition parameters stores a reference to the **TextLanguage** object. The recognition functions take this object either as a sub-object of the **PageProcessingParams** object passed to them as an input parameter, or from a block in a **Layout** object.

The **TextLanguage** object exposes the following main properties:

- **Internal name**. We recommend selecting a unique name for the internal language; it is already unique for the languages supplied in the ABBYY FineReader Engine distribution pack. Be sure to make the names of new languages unique.

- **Letter sets**. The **TextLanguage** object contains the following letter sets: punctuation marks that may be encountered between words, prohibited characters, and additional punctuation marks that go immediately before and after words.

- **Prohibiting dictionaries**. You can create a collection of prohibiting dictionaries using the **ProhibitingDictionaries** property of the **TextLanguage** object. The words from these dictionaries cannot be used as variants of a recognized word. But if no variants are left and using a prohibited word is the only option, words from these dictionaries may still appear in the recognized text. See Working with Dictionaries.

Recognition language for characters

During the recognition, the text is separated into words, with one or several recognition languages corresponding to each word. One recognition language is assigned to each character in a word. This recognition language is represented by the **BaseLanguage** object and is accessible via the **ITextLanguage::BaseLanguages** property.

The **BaseLanguage** object has the following properties:

- **Internal name**. We recommend selecting a unique name for the internal language; it is already unique for the languages supplied with the ABBYY FineReader Engine distribution pack. Be sure to make the names of new languages unique.
  If one base recognition language corresponds to one recognized word, the **ICharParams::LanguageName** property for each character in this word is set to the internal name of the base language after recognition. If several base recognition languages correspond to one word (e.g. for bilingual compound words), the **ICharParams::LanguageName** property for the characters in this word is empty. The **ICharParams::LanguageId** property contains the identifier of the base language no matter what the recognized word.

- **Letter sets**. A letter set comprises letters that form the alphabet of the language, letters that form its extended alphabet (used in loan words), punctuation marks that go immediately before and after words, characters that are allowed inside words but are ignored by the internal spelling check system, and symbols allowed in subscript and in superscript.

- **Dictionary**. A recognition language for a word may have a dictionary attached to it. See Working with Dictionaries.

Creating a compound recognition language

ABBYY FineReader Engine provides an easy way to create compound recognition languages made up of several predefined recognition languages. This is done via the **LanguageDatabase** object. For example, you may create a recognition language that includes both English and German words:

1. Create a **LanguageDatabase** object by calling the **IEngine::CreateLanguageDatabase** method.

2. Call the **ILanguageDatabase::CreateTextLanguage** or **ILanguageDatabase::CreateCompoundTextLanguage** method with the parameters "English" and "German."

3. Use the received **TextLanguage** object for text recognition.

The **LanguageDatabase** object also allows you to import custom user-defined languages created in ABBYY FineReader. ABBYY FineReader's Graphical User Interface provides a means for creating custom recognition languages with letter sets, dictionaries, and other parameters specified by the user. See the ABBYY FineReader User's Guide for details. The recognition languages created in this way are stored in a set of files and may be accessed by using the **LanguageDatabase** object. If you wish to use the languages created in ABBYY FineReader, do the following:

1. Create a **LanguageDatabase** object by calling the **IEngine::CreateLanguageDatabase** method.

2. Load the languages into the **LanguageDatabase** object using the **ILanguageDatabase::LoadFrom** method.

3. Get the required language by its name as a **TextLanguage** object from the **LanguageDatabase** object.

4. Use the received **TextLanguage** object for text recognition.

See also

Working with Dictionaries
Recognizing Words with Spaces
Recognizing Hieroglyphic Languages

Working with Layout and Blocks

When processing a document, ABBYY FineReader Engine first analyzes its layout and detects certain areas on the document pages. These areas are called "blocks." Blocks determine how and in what order the image areas should be recognized.

In ABBYY FineReader Engine, the **Layout** object serves as a storage for blocks and recognized text. The basic document processing scenarios work with the layout within the **FRDocument** object, which represents the document being processed. To access the layout of a document page, use the **IFRPage::Layout** property.

Geometrical characteristics of page layout

The **Layout** object has the following geometrical parameters: width and height. A user should not care about assigning values to them — this is done automatically when the **Layout** object is being used. An analysis or recognition method initializes the geometrical properties of the **Layout** object with the values of the corresponding properties of the black-and-white image page of the **ImageDocument** object. It is the black-and-white image page that is used for text recognition, which is why the geometrical characteristics of the black-and-white image page are copied into the **Layout** object.

Layout blocks

The **Layout** object provides access to the layout structure via the **Blocks** and **BlackSeparators** properties. Both these properties provide access to the **LayoutBlocks** sub-objects, which represent collections of blocks. The first one refers to the main set of layout blocks, which contains texts, tables, pictures, barcodes, and checkmarks. The second one refers to the collection of blocks for separators. Separators are black lines that are detected during the page layout analysis. They are used for more precise page layout reconstruction during export.

Each block has its region, which is a set of rectangles positioned one under another. A region is represented by the **Region** object.

Depending on the type of data contained in the block, blocks may be of different types, each having its own specific properties. These properties are accessible via the corresponding block type objects which can be received using the methods of the **Block** object. The corresponding block type interfaces are derived from the **IBlock** interface and inherit all its properties. The following block types are available:

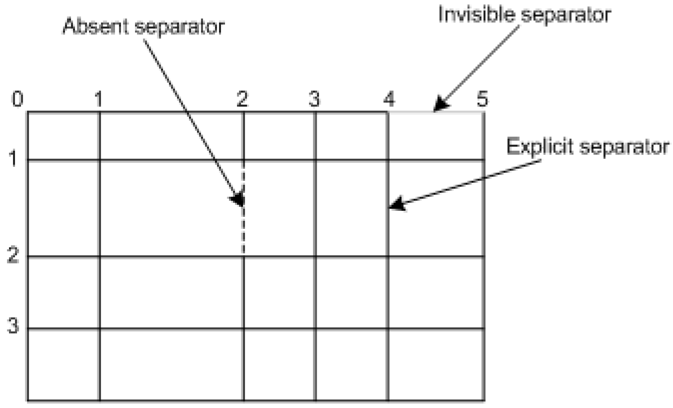
Text block

These blocks correspond to an image zone recognized as formatted text. Properties of this block type are accessible via the **TextBlock** object. The recognized text from the part of the image enclosed by this block is also accessible via this object.

Table block

The region of blocks of this type may consist of one rectangle only. The properties of this block type are accessible via the **TableBlock** object. The structure of the table is described by two collections of table separators, horizontal and vertical (the **TableSeparators** objects), and by a collection of table cells (the **TableCells** object). Each table cell is treated as a block of some type. A cell has four coordinates — the indexes of the left, right, top, and bottom separators that enclose it. The recognized text is a property of a single cell, not of the entire table. If a cell is a picture, the image enclosed in the cell bounds is not recognized and is displayed as a picture in the recognized text. Table separators are may be of different types. A separator type is in fact a property of the corresponding separator's portion which lies between its nearest intersections with other separators, and not of the entire separator. Separators may be of the following types:

- **Absent**. This type is assigned to table separators that go through merged cells.
- **Unknown**. This type is assigned by default to every newly added table separator.
- **Invisible**. This type is assigned to an "imaginary" table separator created as a result of table structure analysis in a place where the source table did not have one but where it should logically be.
- **Explicit**. Table separators of this type appear where the black lines of the source table are located.
- **Multiple**. This type of separator may appear as a result of table editing.



Base coordinates in a table and types of table separators

Raster picture block

This one represents an image zone treated as a raster picture. The part of the image that this block encloses is not recognized, and the block is exported "as is." The properties of this block type are represented by the **RasterPictureBlock** object.

Vector picture block

This one represents an image zone treated as a vector picture. Blocks of this type may appear in the layout only if a page has been analyzed with the **IPageAnalysisParams::DetectVectorGraphics** property set to TRUE. Usually, background pictures are recognized as blocks of this type. The properties of this block type are represented by the **VectorPictureBlock** object.

Barcode block

A part of image enclosed by a block of this type is treated as a barcode. ABBYY FineReader Engine may recognize barcodes of several types, it may also detect barcode types automatically. The information read from a recognized barcode is accessible via the barcode block specific properties represented by the **BarcodeBlock** object.

Checkmark block

A part of image enclosed by a block of this type is treated as a checkmark. It corresponds to an image area recognized as a checkmark. The information read from a recognized checkmark is accessible via the checkmark block specific properties represented by the **CheckmarkBlock** object.

Checkmarks group block

A part of image enclosed by a block of this type is treated as a checkmarks group. It corresponds to an image area recognized as checkmarks group. The information read from a recognized checkmarks group is accessible via the checkmarks group block specific properties represented by the **CheckmarkGroup** object.

Separator block

A part of image enclosed by a block of this type is treated as a separator. Separators are lines that are detected during the page layout analysis. They may be parts of a table, lines that separate different text elements, etc. The coordinates and type of a separator are accessible via the **SeparatorBlock** object.

Separators group block

A part of image enclosed by a block of this type is treated as a separators group. It corresponds to an image zone recognized as a group of separators. A group of separators usually includes four separators, which form a rectangle. For example, four lines of a table border are recognized as a separator group. Each separator group contains a collection of separator blocks. The specific properties of a separators group block are represented by the **SeparatorGroup** object.

Adding blocks manually

Blocks are found on a page automatically during layout analysis. But you may want to create a **Layout** object and add blocks manually. In this case:

1. Create a **Layout** object with the help of the **CreateLayout** method of the **Engine** object.
2. Create a **Region** object for the block using the **IEngine::CreateRegion** method and add rectangles to it using the **IRegion::AddRect** method.
3. Create a block of required type and add it into the collection using the **AddBlock** method of the **Layout** object.
4. Set the required parameters of the block (use the block properties object corresponding to the type of block).

Changing block type

A block type can only be changed using the following procedure:

1. Delete this block from the layout by calling the **ILayoutBlocks::Remove** method.
2. Create a **Region** object for the block using the **IEngine::CreateRegion** method and add rectangles to it using the **IRegion::AddRect** method.
3. Create a block of required type and add it into the collection using the **AddBlock** or **InsertBlock** method of the **Layout** object.

See also

Recognizing Checkmarks
Working with Text

Working with Text

The text that ABBYY FineReader Engine works with is plain text, i.e. it does not contain frames, tables, and so on. All characters are Unicode. Plain text may contain the following special characters:

- 0x2028 — Line break symbol.
- 0xFFFC — Object replacement character. Denotes an embedded picture inside the text.
- 0x0009 — Tabulation.
- 0x005E — Circumflex accent.

The attributes and formatting of a text is available via the corresponding objects and properties.

You can work with the recognized text of a document either via its page layout (**IFRPage::Layout** property) or via the logical structure of the document (**IFRPage::PageStructure** and **IFRDocument::DocumentStructure** properties). The recognized text in the layout becomes available after recognition, though some of its attributes are unavailable until page and document synthesis are performed. To access the recognized text in the logical structure of the document, you must first perform full document synthesis. This provides access to the full set of text attributes, including its role in the document and formatting attributes.

This section describes working with text via the page layout. For more information about working with text via the logical structure of a document, please see the Working with the Logical Structure of a Document section.

Recognized text in the layout

Only text, table, and barcode blocks contain text after recognition. Other blocks have no text. The **Text** object provides access to the recognized text of text and table blocks, while the **BarcodeText** object provides access to the text of a barcode block.

To access the recognized text of a block, do the following:

- **For text blocks**
  Use the **ITextBlock::Text** property.
- **For table blocks**
  1. Receive the collection of table cells using the **ITableBlock::Cells** property.
  2. Select the desired cell. Use the methods of the **TableCells** object.
  3. Receive the block object of the cell (the **ITableCell::Block** property).
  4. Check that the block is of type BT_Text (the **IBlock::Type** property) and receive the **TextBlock** object using the **IBlock::GetAsTextBlock** method.
  5. Use the **ITextBlock::Text** property.
- **For barcode blocks**
  Receive the barcode text using the **IBarcodeBlock::BarcodeText** or **IBarcodeBlock::Text** property. The first one returns the **BarcodeText** object, which is a collection of characters of the recognized barcode (the **BarcodeSymbol** objects). The

second one returns the text of the barcode as a single string. The **BarcodeText** allows you to edit the text of the barcode. The **IBarcodeBlock::Text** property is read-only.

Text and paragraphs

The **Text** object contains a collection of paragraphs. This collection is a **Paragraphs** object accessible via the **Paragraphs** property of the **Text** object. The **Paragraphs** object is a collection of **Paragraph** objects. The recognized text is accessible via the **IParagraph::Text** property. The text in the property is a Unicode string.

There also exists a **ParagraphParams** object that contains attributes specific to the whole paragraph, such as information on its alignment and indent. This object is accessible via the **IParagraph::ExtendedParams** property.

The **IParagraph::Lines** property provides access to a collection of paragraph lines represented by the **ParagraphLines** object, which, in turn, is a collection of **ParagraphLine** objects. The latter provides information on the geometrical position of a single paragraph line and so represents the division of the text into lines.

The **IParagraph::Words** property provides access to a collection of paragraph words represented by the **Words** object, which is a collection of **Word** objects. The **Word** object provides access to a single word of the paragraph.

Character attributes

Each character of the text has its own parameters. They are accessible via the **CharParams** object. The **CharParams** object has a large set of character attributes such as its geometrical parameters, its font, and language. The **CharParams** object contains the character itself in the **SelectedCharacterRecognitionVariant** property.

The position of a character in the text is defined by the index of its paragraph and its own index in this paragraph. There also exists a so-called "special position" in the text: the index of the paragraph is the total number of paragraphs and the index of the character is 0. This is the insertion point at the end of the text. Some methods of the **Text** object perform operations with the special position, i.e. insert another text fragment or picture in it.

The **SelectedCharacterRecognitionVariant** property of the **CharParams** object provides access to an extended set of attributes specific to a single character, represented by the **CharacterRecognitionVariant** object. These attributes are set during the recognition and provide some internal recognition information specific to the character. In particular, this object provides more precise information on character recognition certainty, the probability that the character is in a serif font, etc.

Text editing

You may try changing the attributes of the **Text** object, but you should do it very carefully if the text is to be exported into an external format. The ABBYY FineReader Engine export methods assume that the recognized text is the result of recognition, and that the user only corrected the recognition errors and made no other changes. The objects of the **Text** group have a lot of interdependent properties, and often changing one of these properties requires changing others as well. For this reason changes in the recognized text's attributes may sometimes result in unpredictable export results.

See also

Working with Layout and Blocks
Working with Languages
Using Voting Algorithms
Working with the Logical Structure of a Document

Working with the Logical Structure of a Document

The logical structure of a document is recreated during document synthesis, which is performed after recognition. During document synthesis, formatting attributes, including headers, footers, page numbers, fonts, styles etc., are also detected. ABBYY FineReader Engine provides the **DocumentStructure** and **PageStructure** objects and a set of their subobjects to access the results of document and page synthesis. You can access these objects via the **IFRDocument::DocumentStructure** and **IFRPage::PageStructure** properties.

Important! Pointers to a child object's interfaces are valid until the parent object exists. An attempt to access a child object after its parent object has been destroyed may result in an error. Therefore, you must keep the reference to the **FRDocument** object (which is the root for the document synthesis objects) while you work with the elements of the document structure. Please see Working with Properties for details.

Recognized text in the logical structure of a document

Important! To access the recognized text in the logical structure of a document, you must first perform full document synthesis. The document synthesis objects become valuable only after synthesis.

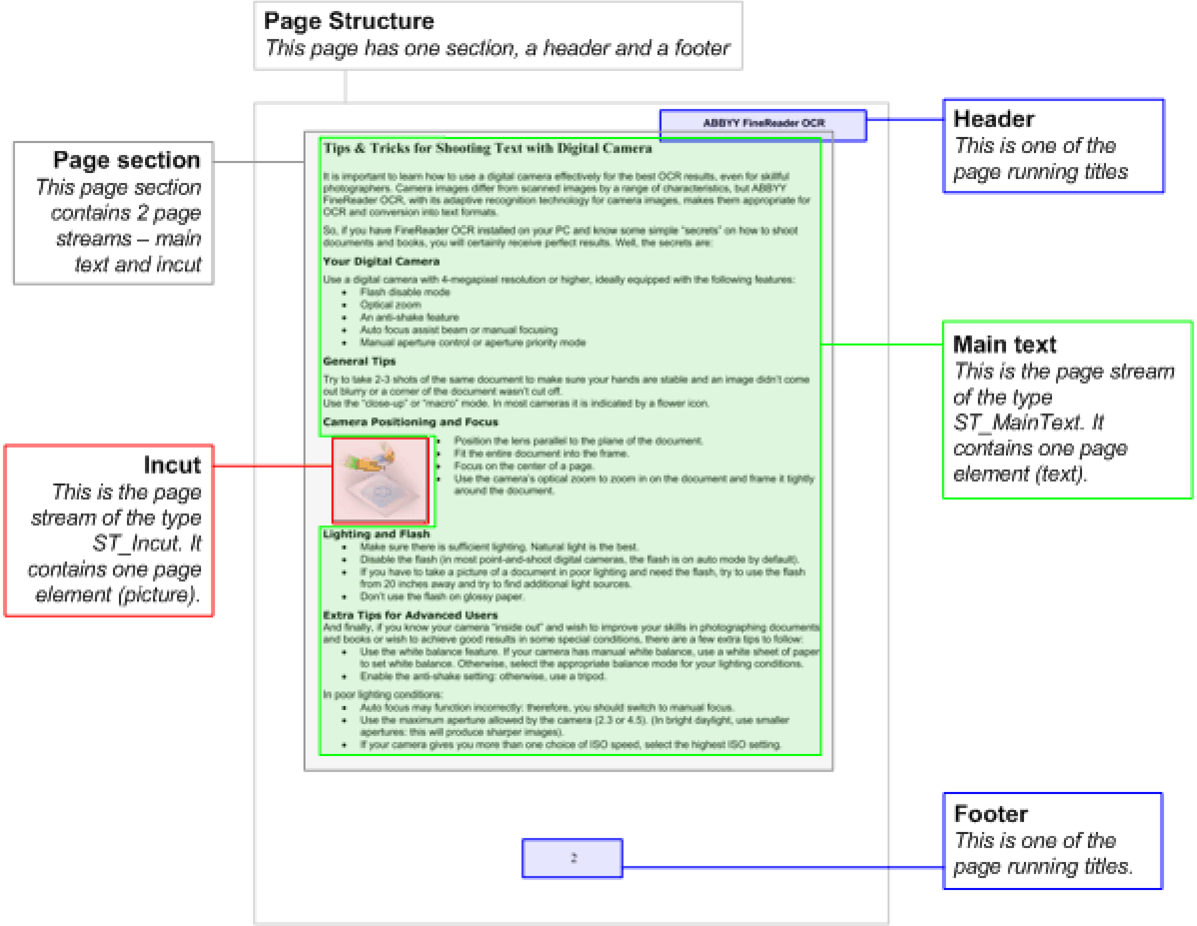
In the logical structure of a document, recognized text is an attribute of:

- page elements (**PageElement** object)
- document elements (**DocumentElement** object)
- running titles (**RunningTitle**, **RunningTitleSeriesText** objects)

Note: You can work with the recognized text of the document either via its page layout or via the logical structure. This section describes working with text via the logical structure. For information about working with text via the page layout, please see the Working with Text section.

Working with the text of a page

The page structure usually includes one or several page sections. Each of these sections consists of one or several page streams: main text, incuts, footnotes, and artefacts. Each page stream includes one or several page elements: text, table, barcode, or picture. Page structure may also include running titles.



Page Structure
This page has one section, a header and a footer

Page section
This page section contains 2 page streams – main text and incut

Header
This is one of the page running titles

Main text
This is the page stream of the type ST_MainText. It contains one page element (text).

Incut
This is the page stream of the type ST_Incut. It contains one page element (picture).

Footer
This is one of the page running titles.

ABBYY FineReader OCR

Tips & Tricks for Shooting Text with Digital Camera

It is important to learn how to use a digital camera effectively for the best OCR results, even for skillful photographers. Camera images differ from scanned images by a range of characteristics, but ABBYY FineReader OCR, with its adaptive recognition technology for camera images, makes them appropriate for OCR and conversion into text formats.

So, if you have FineReader OCR installed on your PC and know some simple "secrets" on how to shoot documents and books, you will certainly receive perfect results. Well, the secrets are:

Your Digital Camera

Use a digital camera with 4-megapixel resolution or higher, ideally equipped with the following features:

- Flash disable mode
- Optical zoom
- An anti-shake feature
- Auto focus assist beam or manual focusing
- Manual aperture control or aperture priority mode

General Tips

Try to take 2-3 shots of the same document to make sure your hands are stable and an image didn't come out blurry or a corner of the document wasn't cut off.
Use the "close-up" or "macro" mode. In most cameras it is indicated by a flower icon.

Camera Positioning and Focus

- Position the lens parallel to the plane of the document.
- Fit the entire document into the frame.
- Focus on the center of a page.
- Use the camera's optical zoom to zoom in on the document and frame it tightly around the document.

Lighting and Flash

- Make sure there is sufficient lighting. Natural light is the best.
- Disable the flash (in most point-and-shoot digital cameras, the flash is on auto mode by default).
- If you have to take a picture of a document in poor lighting and need the flash, try to use the flash from 20 inches away and try to find additional light sources.
- Don't use the flash on glossy paper.

Extra Tips for Advanced Users

And finally, if you know your camera "inside out" and wish to improve your skills in photographing documents and books or wish to achieve good results in some special conditions, there are a few extra tips to follow:

- Use the white balance feature. If your camera has manual white balance, use a white sheet of paper to set white balance. Otherwise, select the appropriate balance mode for your lighting conditions.
- Enable the anti-shake setting: otherwise, use a tripod.

In poor lighting conditions:

- Auto focus may function incorrectly: therefore, you should switch to manual focus.
- Use the maximum aperture allowed by the camera (2.3 or 4.5). (In bright daylight, use smaller apertures: this will produce sharper images).
- If your camera gives you more than one choice of ISO speed, select the highest ISO setting.

2

Main text

To work with the main text of a page, you can receive the corresponding **PageStream** object using the **IPageSection::MainStream** property. Then receive its collection of page elements — the **PageElements** object (**IPageStream::PageElements** property).

Working with the text of a page element:

For texts

Use the **GetAsText** method to receive the **Text** object.

For tables

1. Use the **GetAsTable** method to receive the **TextTable** object.
2. If you want to receive the text of a cell, receive this cell using the **ITextTable::Cell** property. If you want to receive the text of a caption, receive the collection of captions using the **ITextTable::Captions** property and select the desired caption from the collection.
3. Receive the **PageElement** object of the **TextTableCell** or **Caption** object (use the **Element** property).

4. The received page element will be of type PET_Text. Use its **GetAsText** method to receive the **Text** object.

For barcodes

1. Use the **GetAsBarcode** method to receive the **TextBarcode** object.
2. Use the **ITextBarcode::Text** property to receive the Text object.

Note: We recommend working with the text of barcodes via the layout, as this is more suitable for barcodes and does not require synthesis.

For pictures

For a picture you may receive a text of its caption.

1. Use the **GetAsPicture** method to receive the **TextPicture** object.
2. Receive the collection of captions using the **ITextPicture::Captions** property and select the desired caption from the collection.
3. Receive the **PageElement** object of the **Caption** object (use the **Element** property).
4. The received page element will be of type PET_Text. Use its **GetAsText** method to receive the **Text** object.

Incuts and footnotes

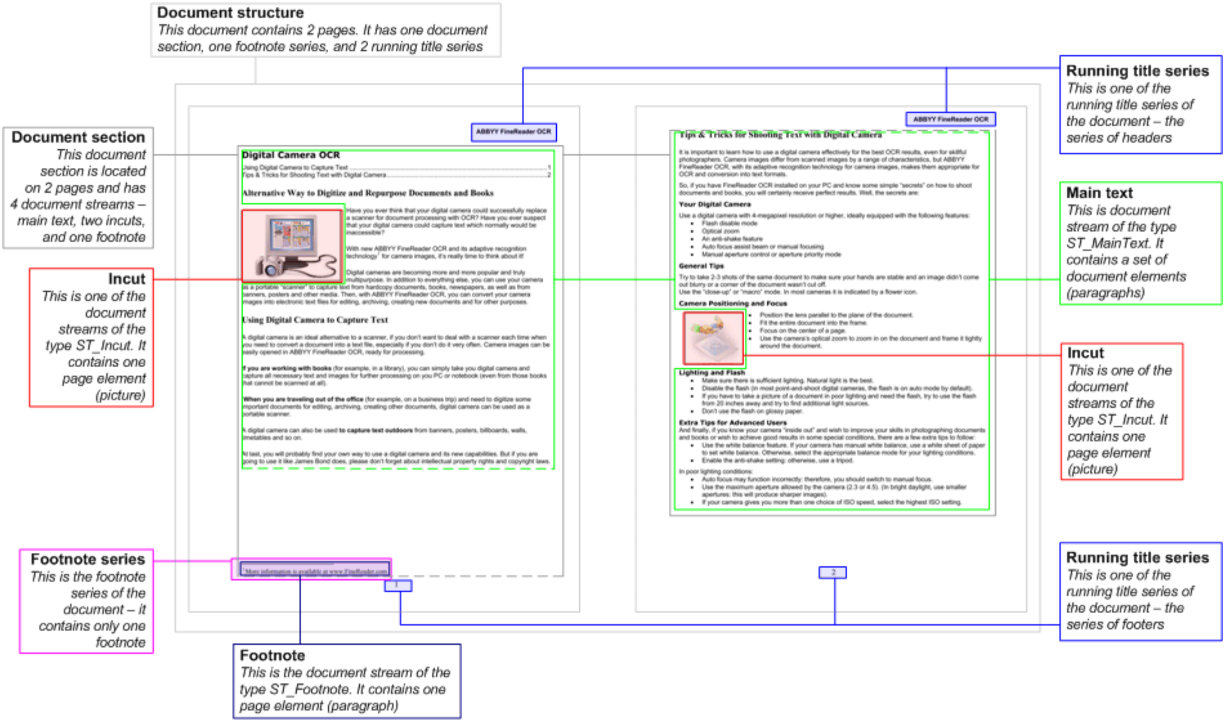
To work with the text of an incut or footnote, receive the collection of page streams (**IPageSection::PageStreams** property) and find the required **PageStream** object in the collection (**IPageStream::Type = ST_Incut** or **IPageStream::Type = ST_Footnote**). Then receive its collection of page elements — the **PageElements** object (**IPageStream::PageElements** property). Further work with the text of a page element is the same as for the main text (see above).

Running titles

To work with the text of a running title, receive the **RunningTitle** object using the **IPageStructure::Header** or **IPageStructure::Footer** property. Then use the **Text** property of the **RunningTitle** object.

Working with the text of the whole document

The document structure usually includes one or several document sections. Each of these sections consists of one or several document streams: main text, incuts, and footnotes. Artefacts are not document streams. Each page stream includes one or several page elements: paragraph, table, barcode, or picture. The document structure may also include a collection of running title series.


Document structure
This document contains 2 pages. It has one document section, one footnote series, and 2 running title series
Document section
This document section is located on 2 pages and has 4 document streams – main text, two incuts, and one footnote
Incut
This is one of the document streams of the type ST_Incut. It contains one page element (picture)
Footnote series
This is the footnote series of the document – it contains only one footnote
Footnote
This is the document stream of the type ST_Footnote. It contains one page element (paragraph)
Running title series
This is one of the running title series of the document – the series of headers
Main text
This is document stream of the type ST_MainText. It contains a set of document elements (paragraphs)
Incut
This is one of the document streams of the type ST_Incut. It contains one page element (picture)
Running title series
This is one of the running title series of the document – the series of footers
ABBYY FineReader OCR
Digital Camera OCR
Tips & Tricks for Shooting Text with Digital Camera
1
2

Main text, incuts, and footnotes

To work with the main text, the text of an incut or footnote, find the required **DocumentStream** object in the document section (**IDocumentSection::DocumentStream** property). Iterate through its elements (the **DocumentElement** objects) using the **FirstElement**, **LastElement**, **NextElement**, **PrevElement** properties.

The work with the text of a document element depends on its type:

For paragraphs

Use the **GetAsParagraph** method to receive the **Paragraph** object.

For tables

1. Use the **GetAsTable** method to receive the **TextTable** object.
2. If you want to receive the text of a cell, receive this cell using the **ITextTable::Cell** property. If you want to receive the text of a caption, receive the collection of captions using the **ITextTable::Captions** property and select the desired caption from the collection.
3. Receive the **PageElement** object of the **TextTableCell** or **Caption** object (use the **Element** property).
4. The received page element will be of type PET_Text. Use its **GetAsText** method to receive the **Text** object.

For barcodes

1. Use the **GetAsBarcode** method to receive the **TextBarcode** object.
2. Use the **ITextBarcode::Text** property to receive the Text object.

Note: We recommend working with the text of barcodes via the layout, as this is more suitable for barcodes and does not require synthesis.

For pictures

For a picture you may receive the text of its caption.

1. Use the **GetAsPicture** method to receive the **TextPicture** object.
2. Receive the collection of captions using the **ITextPicture::Captions** property and select the desired caption from the collection.
3. Receive the **PageElement** object of the **Caption** object (use the **Element** property).
4. The received page element will be of type PET_Text. Use its **GetAsText** method to receive the **Text** object.

Series of running title

You may receive the text of the whole series of running title:

1. Receive the **RunningTitleSeriesArray** object using the **IDocumentStructure::RunningTitleSeriesArray** property.
2. Find the desired **RunningTitleSeries** object in the collection and then, using its **FooterOnEven**, **FooterOnOdd**, **HeaderOnEven**, **HeaderOnOdd** properties, receive the **RunningTitleSeriesText** object.
3. Use the **Text** property of the **RunningTitleSeriesText** object to view all text of the series of running titles.

See also

Document Synthesis Objects
Working with Layout and Blocks

Using Voting API

Developers can combine several Engines in their recognition solutions. When multiple Engines generate different recognition variants for a character or word, the developer can select the best variant by voting between the variants. To enable voting, the ABBYY FineReader Engine has a special Voting API which provides access to different hypotheses of character or word recognition with corresponding weight values. In addition to voting, the developer can use the Voting API to check recognition results using his own databases and algorithms, and to correct text. For example, the developer can build words from letters or check all generated hypotheses.

Note: The Voting API is not available for recognizing handprinted texts.

The **WordRecognitionVariants** object represents a collection of hypotheses for a word, and the **CharacterRecognitionVariants** object represents a collection of hypotheses for a character. The elements of these collections are the **WordRecognitionVariant** and **CharacterRecognitionVariant** objects respectively.

The **WordRecognitionVariant** object represents a single hypothesis for a word and contains the text of the hypothesis, type of model, the average width of stroke, and information on whether the hypothesis has been found in the dictionary. The **GetCharParams** method of this object provides access to the parameters of a single character.

The **CharacterRecognitionVariant** object represents a single hypothesis for a character and contains character confidence, probability that a character is written with a serif font, and information on whether the character is superscript or subscript.

If you wish to save all hypotheses for a word or character during recognition, do the following:

1. Set the **SaveWordRecognitionVariants** and **SaveCharacterRecognitionVariants** properties of the **RecognizerParams** object to TRUE.

2. Pass the **RecognizerParams** object as a sub-object of the **PageProcessingParams** object to one of the ABBYY FineReader Engine recognition methods.

3. The collection of hypotheses is accessible after recognition through the **ICharParams::WordRecognitionVariants**, **ICharParams::CharacterRecognitionVariants** properties and the **IParagraph::GetWordRecognitionVariants** method.
   Note: These methods return zero for non-printable characters (spaces, carriage returns, etc.) and characters which were not recognized but added to the text during editing. Zero is also returned if the text was recognized by one of the previous ABBYY FineReader Engine versions. The hypotheses collections contain recognition variants ranked from best to worst. If the **SaveWordRecognitionVariants** or **SaveCharacterRecognitionVariants** property of the **RecognizerParams** object is set to FALSE, the corresponding collection will contain only one element.

Sample code in Visual Basic:

Visual Basic code

```
' Procedure of hypotheses generation for all words and characters of a text block
Private Sub GetVariants(block As FREngine.block)
  ' Collection of character recognition hypotheses
  Dim characterRecognitionVariants As FREngine.characterRecognitionVariants
  ' A single character recognition hypothesis
  Dim characterRecognitionVariant As FREngine.characterRecognitionVariant
  ' Collection of word recognition hypotheses
  Dim wordRecognitionVariants As FREngine.wordRecognitionVariants
  ' A single word recognition hypothesis
  Dim wordRecognitionVariant As FREngine.wordRecognitionVariant
  ' Create CharParams object
  Dim charParams As FREngine.charParams
  Set charParams = Engine.CreateCharParams
  ' Get the collection of paragraphs of the recognized text
  Dim paragraphs As FREngine.paragraphs
  Set paragraphs = block.GetAsTextBlock.text.paragraphs

  Dim i, j, k As Integer
  ' Iterate the collection of paragraphs
  For i = 0 To paragraphs.Count - 1
    ' Iterate characters in paragraph
    For j = 0 To paragraphs.Item(i).Length
      ' Get parameters of a single character
      paragraphs.Item(i).GetCharParams j, charParams
      ' Get the collection of character recognition hypotheses
      Set characterRecognitionVariants = charParams.CharacterRecognitionVariants
      ' Get the collection of word recognition hypotheses
      Set wordRecognitionVariants = charParams.WordRecognitionVariants

      ' Get a single word recognition hypothesis
      If Not (wordRecognitionVariant Is Nothing) Then
```

```
        For k = 0 To wordRecognitionVariants.Count - 1
          Set wordRecognitionVariant = wordRecognitionVariants.Item(k)
        Next k
      End If

      ' Get a single character recognition hypothesis
      If Not (characterRecognitionVariants Is Nothing) Then
        For k = 0 To characterRecognitionVariants.Count - 1
          Set characterRecognitionVariant = characterRecognitionVariants.Item(k)
        Next k
      End If
    Next j
  Next i
End Sub

...

' Create PageProcessingParams object
Dim pageProcessingParams As FREngine.PageProcessingParams
Set pageProcessingParams = Engine.CreatePageProcessingParams

pageProcessingParams.RecognizerParams.SaveCharacterRecognitionVariants = True
pageProcessingParams.RecognizerParams.SaveWordRecognitionVariants = True

frDocument.Process pageProcessingParams
Dim i As Integer
' Iterate layout blocks
For i = 0 To layout.Blocks.Count - 1
  If layout.Blocks.Item(i).Type = BT_Text Then
    ' Call GetVariants procedure for text blocks
    GetVariants layout.Blocks.Item(i)
  End If
Next i
...
```

What is the difference between the CharConfidence and the IsSuspicious properties

The **CharConfidence** property of the **PlainText** and the **CharacterRecognitionVariant** objects is the read-only long property which stores the value of character confidence. It is in the range from 0 to 100, and 255 corresponds to the fact that confidence is undefined. It represents an estimate of recognition confidence of a character in percentage points. The greater its value, the greater the confidence. Character confidence can be undefined, for example, for characters which were added during text editing.

Recognition confidence of a character image is a numerical estimate of the similarity of this image and the "ideal" whose recognition confidence would be 100%. When recognizing a character, the program provides several recognition variants which are ranked by their confidence values. For example, an image of the letter "e" may be recognized

- as the letter "e" with a confidence of 95%,
- as the letter "c" with a confidence of 85%,
- as the letter "o" with a confidence of 65%, etc.

The sum total of the confidence values of all the recognition variants of a character need not be 100%. The hypothesis with a higher confidence rating is selected as the recognition result. But the choice also depends on the context (i.e. the word to which the character belongs) and the results of a differential comparison. For example, if the word with the “e” hypothesis is not a dictionary word while the word with the “c” hypothesis is a dictionary word, the latter will be selected as the recognition result, and its confidence rating will be 85%. The rest of the recognition variants can be obtained as hypotheses.

The **IsSuspicious** property of the **CharParams** object is the Boolean property. This property set to TRUE means that the character was recognized unreliably. This property is determined by an algorithm which takes into account a number of parameters, such as recognition confidence of a character, nearby characters and their recognition confidence, hypotheses and their recognition confidence, the geometric parameters of a character, and context (i.e. the word to which a character belongs).

See also

CharacterRecognitionVariant
WordRecognitionVariant

Using Text Type Autodetection

Autodetection detects the type of a recognized piece of text. Autodetection is launched if the **TextTypes** property of the **RecognizerParams** object is set to several constants. This mode was primarily designed for recognizing forms. In the case of common OCR we recommend using it only if absolutely necessary.

When autodetection is on, ABBYY FineReader Engine will first try to detect the type of text in the specified block or group of blocks (for these blocks, the **TextTypes** property of the **RecognizerParams** object is set to several constants). ABBYY FineReader Engine will choose from the constants specified in the **TextTypes** property. This property contains an OR superposition of the **TextTypeEnum** enumeration constants which denote the possible text types used for recognition. For example, if it is set to TT_Normal | TT_Index, ABBYY FineReader Engine will assume that the text contains only common typographic text and digits written in a ZIP-code style, ignoring all other variants. The property cannot be set to TT_ToBeDetected. During autodetection, ABBYY FineReader Engine runs preliminary recognition for all of the text types specified in the **TextTypes** property. The preliminary OCR results are then compared, ABBYY FineReader Engine selects the type with the best preliminary results and runs the recognizer for this type.

Note: The RecognizerParams object also provides the TextType and PossibleTextTypes properties for text type autodetection. These properties are obsolete. We recommend using the TextTypes property instead.

How to use autodetection

Autodetection should be used for a set of blocks all of which contain text of the same type. If a separate text type must be selected for each block, you must call the **RecognizeBlocks** method for each block and the **RecognizerParams** object must list the possible text types.

Note: If a single block contains text of different types, this entire text within the block will be recognized as if it was of the same type. For better OCR results, draw separate blocks for each type of text. An exception to this rule is a situation when TT_Normal and TT_Gothic types are encountered in one block. If these types are both specified in the **TextTypes** property, recognition will run as normal.

Selecting the set of text types

The speed and accuracy of autodetection depend on the set of text types specified in the **TextTypes** property. Autodetection is fastest for combinations of TT_Normal, TT_Matrix, TT_Typewriter, TT_OCR_A, and TT_OCR_B types (which can be called the "fast autodetection set"). In this case the recognizer is launched only once, autodetection is carried out during OCR, and single words rather than blocks are used to detect the text type. If only one text type has been specified, autodetection is not launched — the Engine launches the recognizer which corresponds to the specified text type.

Note: If the **TextTypes** property is equal to any combination of TT_Matrix, TT_Typewriter, TT_OCR_A, and TT_OCR_B, then italic fonts and superscript/subscript will not be recognized, regardless of the values of the **ProhibitItalic**, **ProhibitSubscript**, and **ProhibitSuperscript** properties of the **RecognizerParams** object.

In the case of texts which are not covered by the "fast autodetection set," text types are detected by blocks, not by single words. This means that autodetection is slower if the set of possible text types includes text types other than TT_Normal, TT_Matrix, TT_Typewriter, TT_OCR_A, and TT_OCR_B. In this case the Engine needs to carry out preliminary OCR several times — once for the types from the "fast autodetection set" and one preliminary recognition session for each additional text type. Next the results are compared and the best text type is selected.

Important! Be sure to keep the number of text types in the PossibleTextTypes property to a minimum.

Note: If the TextTypes property is equal to any combination of TT_Handprinted and TT_Index, the TrainUserPatterns property of the RecognizerParams object cannot be set to TRUE.

See also

RecognizerParams
TextTypeEnum

Recognizing Checkmarks

ABBYY FineReader Engine 10 supports two block types for checkmarks: checkmark and checkmark group. A checkmark group block is a collection of checkmark blocks. These block types have the corresponding constants BT_Checkmark and BT_CheckmarkGroup in the **BlockTypeEnum** enumeration. The **CheckmarkBlock** and **CheckmarkGroup** objects provide access to the blocks of these types. To receive these objects, you should use the corresponding methods of the **Block** object.

You can recognize single checkmarks as well as checkmark groups.

One check box corresponds to one **CheckmarkBlock** object. Possible check box statuses: checked, not checked, corrected. They correspond to **CheckmarkCheckStateEnum**. A corrected checkmark is a checkmark that was put in the check box and then was crossed out by the user.

⚠**Important**! All the checkmarks within a checkmark group must have the same values for the IsCorrectionEnabled and CheckmarkType properties.

For a checkmark group, you can specify a minimum and maximum number of checked check boxes in the group (**MinimumCheckedInGroup** and **MaximumCheckedInGroup** respectively). These values can be set through **CheckmarkGroup** object and will be used during recognition. The checkmark type can be specified in the **ICheckmarkBlock::CheckmarkType** property.

Recognizing a group of checkmarks

1. Create a **Layout** object using the **IEngine::CreateLayout** method.
2. For each checkmark group:
    1. Create a **Region** object using the **IEngine::CreateRegion** method and add rectangles to it using the **IRegion::AddRect** method.
    2. Create a **Block** object of the checkmark group type and add it into the layout by using the **ILayout::AddBlock** method (use the BT_CheckmarkGroup constant and the created **Region** object as input parameters).
    3. Receive the **CheckmarkGroup** object (use the **IBlock::GetAsCheckmarkGroup** method) and set the required parameters (**MinimumCheckedInGroup**, **MaximumCheckedInGroup**).
3. For each checkmark:
    1. Create the **Region** object using the **IEngine::CreateRegion** method and add rectangles to it using the **IRegion::AddRect** method.
    2. Create a **Block** object of the checkmark type and add it into the checkmark group by using the **ICheckmarkGroup::AddCheckmark** method (use the created **Region** object as an input parameter).
    3. Receive the **CheckmarkBlock** object (use the **IBlock::GetAsCheckmarkBlock** method) and set the required parameters (**CheckmarkType**, **IsCorrectionEnabled**).
4. To recognize the checkmarks, use any of the available methods that perform recognition, such as **IFRPage::Recognize**, **IFRPage::RecognizeBlocks**, **IFRDocument::Recognize**, **IFRDocument::RecognizePages**, etc. (Do not forget to pass the created layout to the **FRPage** object.)

Sample code for checkmark recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
...
// Create a Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();

// Set block region
FREngine::IRegionPtr pRegion = Engine->CreateRegion();
pRegion->AddRect( 0, 0, 100, 50 );

// Create a block of the "checkmark group" type and add into the layout
FREngine::IBlockPtr pBlock = pLayout->AddBlock( FREngine::BT_CheckmarkGroup, pRegion );
FREngine::ICheckmarkGroupPtr pCheckmarkGroup = pBlock->GetAsCheckmarkGroup();

// Create blocks of the "checkmark" type
// and add them to the checkmark group
for( int i = 0; i < 5; i++ ) {
  FREngine::IRegionPtr pCheckmarkRegion = Engine->CreateRegion();
  pRegion->AddRect( 10, 10 + i * 20, 90, 10 + (i + 1) * 20 );
  FREngine::ICheckmarkBlockPtr pCheckmark = pCheckmarkGroup->AddCheckmark(
pCheckmarkRegion );
}
...
```

Visual Basic code

```
...
' Create a Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()

' Set block region
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect 0, 0, 100, 50

' Create a block of the "checkmark group" type and add it into the layout
Dim Block As FREngine.Block
Set Block = Layout.AddBlock(BT_CheckmarkGroup, Region)
Dim CheckmarkGroup As FREngine.CheckmarkGroup
Set CheckmarkGroup = Block.GetAsCheckmarkGroup

' Create blocks of the "checkmark" type
' and add them to the checkmark group
Dim i As Integer
For i = 0 To 4
Dim CheckmarkRegion As FREngine.Region
Set CheckmarkRegion = Engine.CreateRegion()
CheckmarkRegion.AddRect 10, 10 + i * 20, 90, 10 + (i + 1) * 20
Dim Checkmark As FREngine.block
Set Checkmark = CheckmarkGroup.AddCheckmark(CheckmarkRegion)
Next i
...
```

Recognizing a single checkmark

1. Create a **Layout** object using the **IEngine::CreateLayout** method.
2. Create the **Region** object using the **IEngine::CreateRegion** method and add rectangles to it using the **IRegion::AddRect** method.
3. Create a **Block** object of checkmark type and add it into the layout by using the **ILayout::AddBlock** method (use the BT_Checkmark constant and the created **Region** object as input parameters)
4. Receive the **CheckmarkBlock** object (use the **IBlock::GetAsCheckmarkBlock** method) and set the required parameters (**CheckmarkType**, **IsCorrectionEnabled**).
5. To recognize the checkmark, use any of the available recognition methods, such as **IFRPage::Recognize**, **IFRPage::RecognizeBlocks**, **IFRDocument::Recognize**, **IFRDocument::RecognizePages**, etc. (Do not forget to pass the created layout to the **FRPage** object.)

Sample code for checkmark recognition in C++ and Visual Basic:

Visual C++ (COM) code

```
...
// Create a Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
// Set block region
FREngine::IRegionPtr pRegion = Engine->CreateRegion();
```

```
pRegion->AddRect( 0, 0, 100, 50 );
// Create a block of the "checkmark" type and add into the layout
FREngine::IBlockPtr pCheckmark = pLayout->AddBlock( FREngine::BT_Checkmark, pRegion );
...
```

Visual Basic code

```
...
' Create a Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()
' Set block region
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect 0, 0, 100, 50
' Create a block of the "checkmark" type and add it into the layout
Dim Checkmark As FREngine.block
Set Checkmark = Layout.AddBlock(BT_Checkmark, Region)
...
```

See also

CheckmarkBlock
CheckmarkGroup
Working with Layout and Blocks

Recognizing Handprinted Texts

ABBYY FineReader Engine includes ABBYY FineReader ICR (Intelligent Character Recognition) technology which allows you to recognize handprinted texts.

Important!

- Not all recognition languages are available for handprint recognition. The languages which are available for handprint recognition are marked with a special comment in the List of predefined languages.
- In order to recognize Cyrillic handprinted texts, your license must support the Cyrillic ICR module.

You need to set up certain recognition parameters which tell ABBYY FineReader Engine that the text to be recognized is handprinted. This should be done for all blocks which are to be recognized as handprinted.

Note: Automatic layout analysis is not available for handprinted text. The coordinates of the blocks that contain handprinted text must be specified manually. See Working with Layout and Blocks for details.

To set up recognition parameters, do the following for each block with handprinted characters:

1. Specify the **TextTypes** property of the **RecognizerParams** object as TT_Handprinted.
2. [Optional] Specify the **WritingStyle** property of the **RecognizerParams** object which provides additional information about the writing style of the handprinted letters.
3. [Optional] Handprinted letters can often be enclosed in a frame, box, etc. In this case you can use the **FieldMarkingType** property of the **RecognizerParams** object. This property specifies the type of marking around the letters (e.g. underline, frame, box, etc.).
   Note: For the correct operation of this property, please use the **CellsCount** property which allows you to set up the number of character cells in the recognized block.

Sample code in C++(COM) and Visual Basic:

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object
FREngine::IEnginePtr Engine;
...
// Open an image file
...
```

```
// Create a Layout object
FREngine::ILayoutPtr layout = Engine->CreateLayout();
// Set block region
FREngine::IRegionPtr pRegion = Engine->CreateRegion();
pRegion->AddRect( 491, 314, 2268, 404 );

// Create a new block
FREngine::IBlockPtr newBlock = layout->AddBlock( FREngine::BT_Text, pRegion );
FREngine::ITextBlockPtr textBlock = newBlock->GetAsTextBlock();
// Specify the text type
textBlock->RecognizerParams->TextTypes = FREngine::TT_Handprinted;
// Specify the type of marking around the letters
textBlock->RecognizerParams->FieldMarkingType = FREngine::FMT_SimpleText;
// Specify the letters writing style
textBlock->RecognizerParams->WritingStyle = FREngine::WS_American;

// Recognition and export
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object
Public Engine As FREngine.Engine
...
' Open an image file
...
' Create a Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()
' Set block region
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect 491, 314, 2268, 404

' Create a new block
Dim newBlock As FREngine.Block
Set newBlock = Layout.AddBlock(BT_Text, Region)
Dim textBlock As FREngine.textBlock
Set textBlock = newBlock.GetAsTextBlock
' Specify the text type
textBlock.RecognizerParams.TextTypes = TT_Handprinted
' Specify the type of marking around the letters
textBlock.RecognizerParams.FieldMarkingType = FMT_SimpleText
' Specify the letters writing style
textBlock.RecognizerParams.WritingStyle = WS_American

' Recognition and export
...
```

See also

RecognizerParams
List of the Predefined Languages

Recognizing Hieroglyphic Languages

This section deals with certain peculiarities of recognizing and exporting texts in hieroglyphic languages with ABBYY FineReader Engine 10.

First, in order to recognize hieroglyphic languages you must have an ABBYY FineReader Engine license that supports the Chinese, Japanese, and Korean language modules. For more information about licenses and modules, see the Licensing section.

Recognition languages

For hieroglyphic texts, ABBYY FineReader Engine supports the following predefined recognition languages:

- "ChinesePRC"
- "ChineseTaiwan"
- "Japanese"
- "Korean"
- "KoreanHangul"

To select one of the predefined hieroglyphic languages, you can use the **SetPredefinedTextLanguage** method of the **RecognizerParams** object.

ABBYY FineReader Engine supports recognition language combinations consisting of several hieroglyphic languages or combinations of hieroglyphic languages and non-hieroglyphic languages.

Fonts

To prevent any distortions of hieroglyphic characters, you must specify a font which includes hieroglyphs, e.g. Arial Unicode MS, SimSun. You can do this with the help of the **ISynthesisParamsForDocument::AddRecognizedTextFontName** method.

Export

When you export hieroglyphic languages to PDF in any mode other than PDF Image Only (**IPDFExportParams::TextExportMode** = PEM_ImageOnly), fonts are embedded and they are taken from the **Text** object, which represents the recognized text (for the **PDFExportParamsOld** object this means that TRUE is assigned for the **EmbedFonts** property, and FM_UseFontsFromIText for the **FontMode** property).

You can export hieroglyphic languages to PDF/A in "text under the page image" mode (IPDFExportParams::TextExportMode = PEM_ImageOnText).

The procedure of recognition and export

To process documents written in hieroglyphic languages, do the following:

1. Create a **PageProcessingParams** object using the **CreatePageProcessingParams** method of the **Engine** object.
2. Specify a hieroglyphic recognition language. Use the **SetPredefinedTextLanguage** method of the **RecognizerParams** subobject of the **PageProcessingParams** object.
3. Create a **SynthesisParamsForDocument** object using the **CreateSynthesisParamsForDocument** method of the **Engine** object.
4. Specify a font which includes hieroglyphs, e.g. Arial Unicode MS. Use the **ISynthesisParamsForDocument::AddRecognizedTextFontName** method.
5. Pass these parameter objects to the **Process** method of the **FRDocument** object. If you use methods of the **Engine** object, you should call one of the synthesis methods of the **Engine** object with the created **SynthesisParamsForDocument** object as a parameter before export.
6. Perform export of the recognized text with the help of the **Export** method of the **FRDocument** object. If you export to PDF of PDF/A format, specify the required export mode.

Sample code for processing hieroglyphic languages in C++ and Visual Basic:

Visual C++ (COM) code

```
...
// Create a PageProcessingParams object
```

```
FREngine::IPageProcessingParamsPtr pPageProcessingParams = Engine-
>CreatePageProcessingParams();
// Specify hieroglyphic recognition language
pPageProcessingParams->RecognizerParams->SetPredefinedTextLanguage( "Japanese" );
// Create a SynthesisParamsForDocument object
FREngine::ISynthesisParamsForDocumentPtr pSynthesisParams = Engine-
>CreateSynthesisParamsForDocument();
// Specify font
pSynthesisParams->CleanRecognizedTextFontNames();
pSynthesisParams->AddRecognizedTextFontName( "Arial Unicode MS" );
// Recognize and export the document. Suppose that we have already created the
FRDocument object.
frDocument->Process( pPageProcessingParams, 0, pSynthesisParams )
frDocument->Export( L"D:\\Demo.rtf", FREngine::FEF_RTF, 0 );
...
```

Visual Basic code

```
...
' Create a PageProcessingParams object
Dim pageProcessingParams As FREngine.pageProcessingParams
Set pageProcessingParams = Engine.CreatePageProcessingParams
' Specify hieroglyphic recognition language
pageProcessingParams.RecognizerParams.SetPredefinedTextLanguage "Japanese"
' Create a SynthesisParamsForDocument object
Dim synthesisParams As FREngine.SynthesisParamsForDocument
Set synthesisParams = Engine.CreateSynthesisParamsForDocument
' Specify font
synthesisParams.CleanRecognizedTextFontNames
synthesisParams.AddRecognizedTextFontName "Arial Unicode MS"
' Recognize and export the document. Suppose that we have already created the
FRDocument object.
frDocument.Process pageProcessingParams, Nothing, synthesisParams
frDocument.Export "D:\Demo.rtf", FEF_RTF, Nothing
...
```

See also

Working with Languages

Recognizing with Training

ABBYY FineReader Engine can read texts set in practically any font regardless of print quality. Consequently, no prior training is normally required before recognition can take place. ABBYY FineReader Engine, nevertheless, features a number of user pattern training tools.

Train User Pattern mode may come in useful when:

- recognizing texts set in decorative fonts
- recognizing texts containing unusual characters (e.g. mathematical symbols)
- recognizing large volumes (more than a hundred pages) of texts of low print quality

Note: Use Train User Pattern mode only if one of the above applies. In other cases you may obtain a slight increase in recognition quality, but the time and effort involved will probably outweigh the benefit received.

Pattern training works as follows. One or two pages are recognized in training mode, and, subsequently, a pattern is created. A *pattern* is a set of pairs "a character image — the character itself" created during pattern training. A pattern is used as a source of additional information during recognition. ABBYY FineReader Engine then uses this pattern to aid recognition of the remaining text.

Sometimes two or even three characters may get "stuck" together, and ABBYY FineReader Engine may be unable to enclose each character in an individual frame to separate them. If this proves to be the case (i.e. you cannot move the frame so that it contains only one whole character and no other character parts), you can train ABBYY FineReader Engine to recognize the inseparable character combinations in their entirety. Examples of character combinations frequently found stuck together include ff, fi, and fl. Such combinations are referred to as ligatures.

You can find additional information in Training User Patterns.

Note:

- A pattern is only useful in the case of documents that have the same font, font size, and resolution as the document used to create the user pattern.
- Pattern training is not supported for hieroglyphic languages.
- Pattern training cannot be performed when recognizing in parallel processes.

To recognize with training

1. Create a **RecognizerParams** object.
2. Set the **IRecognizerParams::TrainUserPatterns** property to TRUE.
3. Create an empty user pattern file by using the **IEngine::CreateEmptyUserPattern** method.
4. Specify the full path to this user pattern file in the **IRecognizerParams::UserPatternsFile** property.
5. Call a recognition method (e.g. **IFRDocument::Process**) with these recognition parameters. Whenever an unknown character is encountered, the **Pattern Training** dialog will open, with the character image displayed within it.
6. Train your pattern — recognize one or more pages in **Train User Pattern** mode. Trained characters are saved in the user pattern file.
7. [Optional] If you wish to edit this pattern, call the **EditUserPattern** method of the **Engine** object.
8. Recognize the images by using this pattern.

Note: If the **IRecognizerParams::UseBuiltInPatterns** property is set to TRUE, then ABBYY FineReader Engine will use its own built-in patterns for recognition. Set this property to FALSE when you do not want to use the standard ABBYY FineReader Engine patterns for character recognition. This may be useful for recognition of texts typed in decorative or non-standard fonts, in which case you can use your own user-defined patterns trained specifically for these fonts. If the **UserPatternsFile** property (where the path to the user-defined pattern file is stored) is empty, the **UseBuiltInPatterns** property is ignored.

Sample code in C++ and Visual Basic:

Visual C++ (COM) code

```
FREngine::IEnginePtr Engine;
 FREngine::IFRDocumentPtr frDocument;
 ...
 // Create a PageProcessingParams object
 FREngine::IPageProcessingParamsPtr pParams = Engine->CreatePageProcessingParams();
 // Set the TrainUserPatterns property
 pParams->RecognizerParams->TrainUserPatterns = VARIANT_TRUE;
 // Create an empty user pattern file
 Engine->CreateEmptyUserPattern(L"D:\\test.ptn");
 // Set the full path to the user pattern file
 pParams->RecognizerParams->UserPatternsFile = L"D:\\test.ptn";

 // Analyze and recognize the image
 frDocument->Process( pParams, 0, 0 );
 ...
```

Visual Basic code

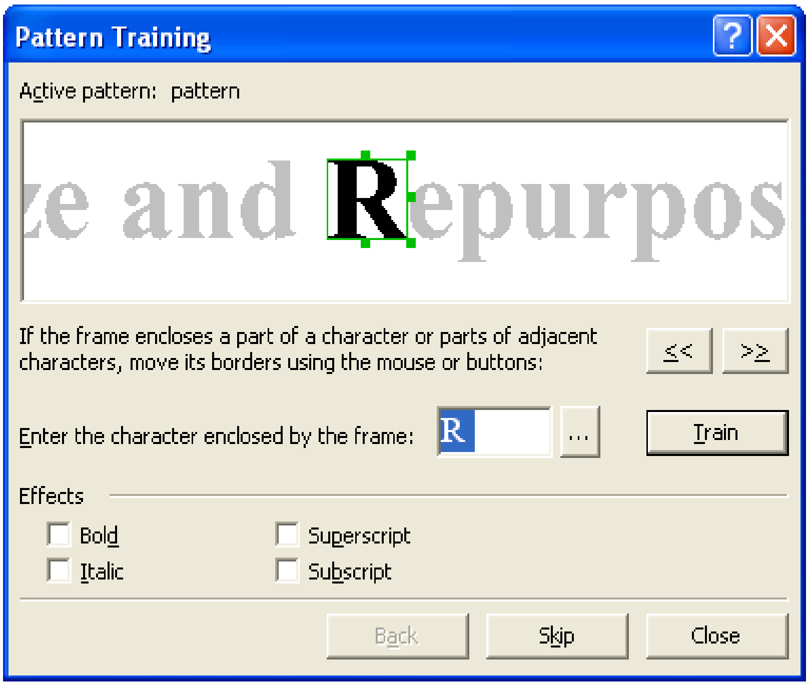
```
Public Engine As FREngine.Engine
 Dim frDocument As FREngine.FRDocument
 ...
 ' Create a PageProcessingParams object
 Dim Params As FREngine.PageProcessingParams
 Set Params = Engine.CreatePageProcessingParams()
 ' Set the TrainUserPatterns property
Params.RecognizerParams.TrainUserPatterns = True
 ' Create an empty user pattern file
Engine.CreateEmptyUserPattern("D:\test.ptn")
 ' Set the full path to the user pattern file
Params.RecognizerParams.UserPatternsFile = "D:\test.ptn"

 ' Analyze and recognize the image
frDocument.Process Params
...
```

See also

Training User Patterns

Training User Patterns

If the **IRecognizerParams::TrainUserPatterns** property is set to TRUE, the **Train User Pattern** mode will be used during the recognition. Whenever an unknown character is encountered, the **Pattern Training** dialog will open, with the character image displayed within it.



Active pattern: pattern

If the frame encloses a part of a character or parts of adjacent characters, move its borders using the mouse or buttons:

Enter the character enclosed by the frame: R

Note: You can also use the IEngine::TrainUserPattern method to perform pattern training without showing the dialog. This method takes as input parameters the TrainingImagesCollection object, which stores a collection of character images, and the character itself.

Training to recognize a character

The frame in the top dialog window should enclose **a single character**, and this character **must be fully enclosed** by the frame. If the frame encloses only part of the character or more than one character, click the frame borders and move them so that the above-stated requirements are met. The << and >> buttons move the frame border as well (and are useful for training italic symbols). Once you have positioned the frame correctly, type in the character and click the **Train** button.

Note:

- You may only train the system to read characters included in the alphabet. If you wish to train ABBYY FineReader Engine to read characters that cannot be entered from the keyboard, use a combination of two characters to denote these non-existent characters or copy the required character from the **Character Table** (click the ... button in the **Pattern Training** dialog to open the **Character Table**).
- If you wish to train the system to retain character formatting, select the corresponding **Italic** or **Bold** item in the **Pattern Training** dialog before clicking the **Train** button.

- Make sure that only uppercase/lowercase characters are entered when training uppercase/lowercase character images respectively.

If you make a mistake during training, click the **Back** button to return the frame to its previous position. The last "image — character" pair to be entered will automatically be removed from the pattern. Note that this "undo" function is limited to the last word trained.

Training to recognize ligatures

A *ligature* is a combination of two or three characters "stuck" together, for example, fi, fl, ffi. These characters are difficult to separate because they are "stuck" together as part of the printing process. In fact, better results can be obtained by treating them as "single" compound characters.

Training ligatures is no different to training separate characters:

1. Type the necessary character combination and click the **Train** button.
2. The frame in the top dialog window should enclose **the entire ligature**. You can move the frame border using the mouse or by clicking the << and >> buttons.

Each pattern may contain up to 1000 new characters. However, you should not create too many ligatures, as it may adversely affect the recognition quality

Training limitations

You should also take the following limitations into account when you train ABBYY FineReader Engine:

- ABBYY FineReader Engine does not differentiate between certain characters which are usually considered different. Such images are recognized as one and the same character. For example, the straight ('), right (’) and left (‘) apostrophes are kept in the pattern as one character — the straight apostrophe. Thus, you will never see the right and left apostrophes in the recognized text, even if you try to train them.

- In some cases a certain image is recognized as a certain character depending on its environment.

- Pattern training is not supported for hieroglyphic languages.

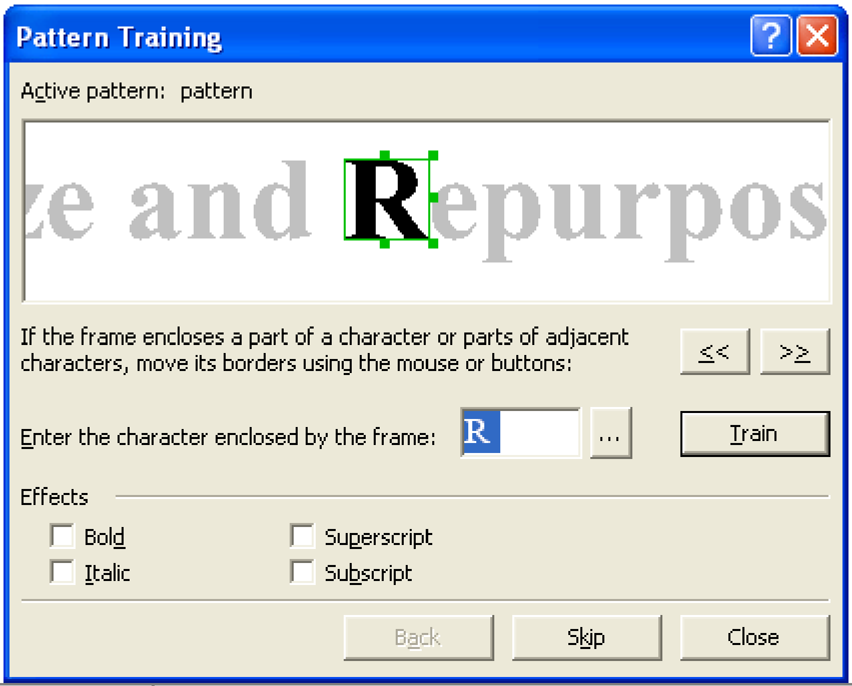
See also

Recognizing with Training
RecognizerParams

Pattern Training Dialog Box

This dialog box displays during recognition if the **IRecognizerParams::TrainUserPatterns** property is TRUE and some user pattern file is specified in the **IRecognizerParams::UserPatternsFile** property.

The top dialog window displays the character you train. The frame enclosing the character **must fully enclose the character or several characters** (in case you train ligatures).



Pattern Training

Active pattern: pattern

If the frame encloses a part of a character or parts of adjacent characters, move its borders using the mouse or buttons:

Enter the character enclosed by the frame: R

Effects

□ Bold
□ Italic
□ Superscript
□ Subscript

Option	Option description
Active pattern	Displays the active user pattern file that should by specified in the **IRecognizerParams::UserPatternsFile** property.
<< **button**	Moves the enclosing frame left. Move the frame so as to enclose the entire character.
>> **button**	Moves the enclosing frame right. Move the frame so as to enclose the entire character.
Enter character	Specify the name of the character enclosed by the frame in the top window. If you train ABBYY FineReader to recognize characters you cannot type, you may use two–character combinations as captions, or you may copy the necessary character from the **character table**. Click the ... button to open the table.
... **button**	Opens the **character table**. You may choose the necessary character from the table and copy it to the **Enter character** field.
Train (button)	Trains the character, i.e. adds the new pair "character image–character caption" to the active pattern.
Effects group	
Italic	Specifies that the current character is italic.
Bold	Specifies that the current character is bold.
Superscript	Specifies that the current character is superscript.
Subscript	Specifies that the current character is subscript.
Back (button)	If you have made a mistake during training, you may click the **Back** button and the frame will go back to the previous position, and the latest pair "image–character" will be removed from the pattern. This "undo" is limited: you may only "undo" training in the last word.
Skip (button)	Skips the current character. Use this button if what the frame encloses is not a character, or if you were unable to enclose a character or a group of character fully. **Note:** If you click the **Skip** button, you'll see the ^ character in the recognized text in the place of the character you skipped.

See also

RecognizerParams
Recognizing with Training
Training User Patterns

Working with Dictionaries

ABBYY FineReader Engine allows you to attach dictionaries of various types to a recognition language, which greatly improves recognition quality.

Dictionary types

Dictionaries may be of several *types*:

Standard dictionary

This type of dictionary is already provided for the predefined languages that have built-in dictionary support (see the comments in the list of predefined languages). Additionally, for some languages there are dictionaries of specialized terms (e.g. medical and law). Standard dictionaries are represented by three or four files. They have names which are usually the same as the full or short name of the language and an .amd, .amm, .amt or .ame extension. Files with .amd, .amm and .amt extensions are always present (they are stored in folder **/Bin**) and cannot be changed. No .ame file is provided with ABBYY FineReader Engine: this is a file for storing a dictionary extension, i.e. words added to the dictionary by the user. You can create a dictionary extension in ABBYY FineReader and then copy the created file to the folder **/Bin** in the ABBYY FineReader Engine folder. ABBYY FineReader stores the extensions of standard dictionaries in %appdata%\ABBYY\FineReader\10.00\UserDictionaries.
This dictionary type is described by the **StandardDictionaryDescription** object.

User dictionary

Can be created either by using the **Dictionary** object or in ABBYY FineReader (see the ABBYY FineReader help file for more details on creating user dictionaries). The **Dictionary** object allows you to add and remove words using its methods, and to edit the dictionary with the help of the **Dictionary** dialog box. This dialog box allows you to import any text file in Windows ANSI and Unicode encoding (the only requirement is that words must be separated by spaces or other non-alphabetic characters).
This dictionary type is described by the **UserDictionaryDescription** object.

Regular-expression-based dictionary

Specifies the rules that define what words are allowed in a language and what words are not allowed.
This dictionary type is described by the **RegExpDictionaryDescription** object.

External dictionary

Allows you to implement your own type of dictionary. This dictionary is represented as the **IExternalDictionary** interface, which is implemented on the client side.
This dictionary type is described by the **ExternalDictionaryDescription** object.

ABBYY FineReader Engine provides a **DictionaryDescription** object for describing all types of dictionary. This is the basic object from which the descriptions of different dictionary types are inherited.

All these dictionary descriptions are elements of the **DictionaryDescriptions** collection.

Creating a dictionary description

To create dictionary descriptions, the **CreateStandardDictionaryDesc**, **CreateUserDictionaryDesc**, **CreateRegExpDictionaryDesc**, and **CreateExternalDictionaryDesc** methods of the **Engine** object are used. These methods return a reference to the object describing the corresponding dictionary type.

Dictionary properties

For each dictionary, the identification property of the dictionary must be specified:

- For a standard dictionary (**StandardDictionaryDescription**), specify its **LanguageId** property, which defines the ID of the language.
- For a user dictionary (**UserDictionaryDescription**), specify its **FileName** property, which provides the path to the user dictionary.
- For a regular-expression-based dictionary (**RegExpDictionaryDescription**), use the **SetText** method to specify the regular expression. See Semantics of ABBYY FineReader Engine Regular Expressions.
- For an external dictionary (**ExternalDictionaryDescription**), use the **SetDictionary** method to specify the dictionary.

All dictionary types are assigned a *weight*. The weight of a dictionary affects the weight of words from the given dictionary when they are detected during recognition. The weight parameter is a percentage and must be non-negative. A weight of 0 does not automatically mean that there is no such dictionary. Weights of more than 100 percent are allowed, but the user must be very careful when using such parameters. The weight is specified in the **IDictionaryDescription::Weight** property and is set to 100 by default.

Standard dictionaries also have a **CanUseTrigrams** option which allows or forbids the program to use trigrams built on the basis of the selected dictionary. *Trigrams* are combinations of three letters. Not all of these combinations occur in real words. A word with a non-dictionary trigram is very likely to be unpronounceable. Trigrams are used to cut off unreliable words. We recommend enabling trigrams for "general" standard dictionaries and disabling them for dictionaries of terms.

Dictionaries of a recognition language

A text recognition language (the **TextLanguage** object) can have both dictionaries containing words of the language and dictionaries with prohibited words. The first ones are specified for each basic recognition language of the text language and are accessible via the **IBaseLanguage::DictionaryDescriptions** property. A base language may have no dictionary attached to it. The prohibiting dictionaries are attached directly to the text recognition language through the **ITextLanguage::ProhibitingDictionaries** property.

If you want only the dictionary words to be allowed during recognition, set the **IBaseLanguage::AllowWordsFromDictionaryOnly** property to TRUE. In this case, a word that is not found in the dictionary of the base language can appear in the recognized text only if ABBYY FineReader Engine found no dictionary variants.

How to attach a dictionary to a recognition language

1. Create a **TextLanguage** object and receive its collection of base languages (the **BaseLanguages** object).
2. Create a **BaseLanguage** object and receive its collection of dictionary descriptions (the **DictionaryDescriptions** object).
3. Create a dictionary description with the help of the **CreateStandardDictionaryDesc**, **CreateUserDictionaryDesc**, **CreateRegExpDictionaryDesc**, or **CreateExternalDictionaryDesc** method of the **Engine** object.
4. [Optional] Specify the weight of the created dictionary.
5. Specify the identification property of the dictionary: the **LanguageId** property for a standard dictionary, the **FileName** property for a user dictionary, call the **IRegExpDictionaryDescription::SetText** method for a regular-expression-based dictionary, or call the **IExternalDictionaryDescription::SetDictionary** method for an external dictionary.
6. Add the newly created dictionary description to the collection of dictionary descriptions of the base language. Use the **Add** method of the **DictionaryDescriptions** collection.
   Note: You can create several dictionaries of different types and add them to the **DictionaryDescriptions** collection of one base recognition language.
7. [Optional] Specify other properties of the **BaseLanguage** object.
8. Add the **BaseLanguage** object with the attached dictionary to the **BaseLanguages** collection.
9. [Optional] Set the prohibiting dictionaries using the **ProhibitingDictionaries** property of the **TextLanguage** object.
10. Assign the created **TextLanguage** object to the **TextLanguage** property of the **RecognizerParams** object.

Sample code in C++ and Visual Basic:

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Create a TextLanguage object and receive its collection of base languages
 FREngine::ITextLanguagePtr pTextLang = Engine->CreateTextLanguage();
 FREngine::IBaseLanguagesPtr pBaseLangCollection = pTextLang->BaseLanguages;

 // Create a BaseLanguage object and receive its collection of dictionary descriptions
 FREngine::IBaseLanguagePtr pBaseLang = Engine->CreateBaseLanguage();
 pBaseLang->InternalName = L"SampleBaseLanguage";
 pBaseLang->put_LetterSet( FREngine::BLLS_Alphabet, L"abc123" );

 FREngine::IDictionaryDescriptionsPtr pDictDescCollection = pBaseLang-
>DictionaryDescriptions;

 // Create a standard dictionary description
 FREngine::IStandardDictionaryDescriptionPtr pDicDescription =
  Engine->CreateStandardDictionaryDesc();

 // [optional] Specify the weight of the created dictionary
 pDicDescription->Weight = 100;
 // Specify the identification property of the dictionary
 pDicDescription->LanguageId = FREngine::LI_EnglishUnitedStates;

 // Add the created dictionary to the DictionaryDescriptions collection
 pDictDescCollection->Add( pDicDescription );

 // [optional] Specify other properties of the BaseLanguage base language
 pBaseLang->AllowWordsFromDictionaryOnly = VARIANT_TRUE;

 // Add the BaseLanguage object with the attached dictionary
 // to the BaseLanguages collection
 pBaseLangCollection->Add( pBaseLang );

 // Create a RecognizerParams object
 FREngine::IRecognizerParamsPtr pParams = Engine->CreateRecognizerParams();
 // Assign the created TextLanguage object to the TextLanguage property
 pParams->TextLanguage = pTextLang;
 ...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Create a TextLanguage object and receive its collection of base languages
 Dim TextLang As FREngine.TextLanguage
 Set TextLang = Engine.CreateTextLanguage

 ' Create a BaseLanguage object and receive its collection of dictionary descriptions
 Dim BaseLangCollection As FREngine.BaseLanguages
 Set BaseLangCollection = TextLang.BaseLanguages

 Dim BaseLang As FREngine.BaseLanguage
 Set BaseLang = Engine.CreateBaseLanguage
 BaseLang.InternalName = "SampleBaseLanguage"
 BaseLang.LetterSet(BLLS_Alphabet) = "abc123"

 Dim DictDescCollection As FREngine.DictionaryDescriptions
 Set DictDescCollection = BaseLang.DictionaryDescriptions

 ' Create a standard dictionary description
 Dim DicDescription As FREngine.StandardDictionaryDescription
 Set DicDescription = Engine.CreateStandardDictionaryDesc

 ' [optional] Specify the weight of the created dictionary
 DicDescription.Weight = 100

 ' Specify the identification property of the dictionary
 DicDescription.LanguageId = LI_EnglishUnitedStates

 ' Add the created dictionary to the DictionaryDescriptions collection
 DictDescCollection.Add DicDescription

 ' [optional] Specify other properties of the BaseLanguage base language
 BaseLang.AllowWordsFromDictionaryOnly = True

 ' Add the BaseLanguage object with the attached dictionary
 ' to the BaseLanguages collection
 BaseLangCollection.Add BaseLang

 ' Create a RecognizerParams object
 Dim Params As FREngine.RecognizerParams
 Set Params = Engine.CreateRecognizerParams
 ' Assign the created TextLanguage object to the TextLanguage property
 Set Params.TextLanguage = TextLang
 ...
```

Cache dictionaries

A cache dictionary is a small dictionary (about a hundred words) which can be changed easily during processing. Cache dictionaries can be used when it is possible to select a dictionary more precisely, e.g. if you find new information about the document during processing. Such dictionaries are suitable for field level recognition.

For example, suppose there are two fields on a form you need to recognize: the name of a city and the name of a street. You have recognized the name of the city and you have the list of streets in this city. In this case you may load the appropriate cache dictionary with the street names and thus recognize the name of the street more quickly and accurately.

ABBYY FineReader Engine provides the **AddWordsToCacheDictionary**, **AddWordToCacheDictionary**, and **CleanCacheDictionary** methods of the **DocumentAnalyzer** object for working with cache dictionaries.

See also

Working with Languages
Recognizing Words with Spaces

Working with ABBYY FineReader Engine Regular Expressions

Regular expressions are used in regular-expression-based dictionaries to define what words are allowed in a language and what are not.

The ABBYY FineReader Engine regular expression alphabet is described in the following table:

Item name	Conventional regular expression sign	Usage examples and explanations
Any character	.	_c.t_ — denotes words like “cat”, “cot”
Character from a character range	[]	_[b-d]ell_ — denotes words like “bell”, “cell”, “dell” _[ty]ell_ — denotes words “tell” and “yell”
Character out of a character range	[^]	_[^y]ell_ — denotes words like “dell”, “cell”, “tell”, but forbids “yell” _[^n-s]ell_ — denotes words like “bell”, “cell”, but forbids “nell”, “oell”, “pell”, “qell”, “rell”, and “sell"
Or	|	_c(a|u)t_ — denotes words “cat” and “cut"
0 or more occurrences in a row	*	_10*_ — denotes numbers 1, 10, 100, 1000, etc.
1 or more occurrences in a row	+	_10+_ — allows numbers 10, 100, 1000, etc., but forbids 1.
Letter or digit	[0-9a-zA-Z]	_[0-9a-zA-Z]_ — allows a single character; _[0-9a-zA-Z]+_ — allows any word
Capital Latin letter	[A-Z]	
Small Latin letter	[a-z]	
Capital Cyrillic letter	[А-Я]	
Small Cyrillic letter	[а-я]	
Digit	[0-9]	
Space	\s	
Character, used by system.	@	
Word from dictionary	@(Dictionary)	The Dictionary parameter sets the path to the user dictionary from which words must be taken. Backslahes in the path must be doubled. For example: _@(D:\\MyFolder\\MyDictionary.amd)_.

Note:

- Some characters used in regular expressions are “auxiliary,” i.e. they are used for system purposes. As you can see from the list above, such characters include square brackets, periods, etc. If you wish to enter an auxiliary character as a normal one, put a backslash (\) before it. Example: *[t-v]x+* denotes words like “tx”, “txx”, “txxx”, etc., “ux”, “uxx”, etc., but *\[t-v\]x+* denotes words like “[t-v]x”, “[t-v]xx”, “[t-v]xxx” etc.

- If you need to group certain regular expression elements, use brackets. For example, *(a|b)+|c* denotes “c” and any combinations like “abbbaaabbb”, “ababab”, etc. (a word of any non-zero length in which there may be any number of a's and b's in any order), whilst *a|b+|c* denotes “a”, “c”, and “b”, “bb”, “bbb”, etc.

Sample regular expressions

Regular expression for dates

The number denoting day may consist of one digit (e.g. 1, 2 etc.) or two digits (e.g. 02, 12), but it cannot be zero (00 or 0). The regular expression for the day should then look like this: *((|0)[1-9])|([1|2][0-9])|(30)|(31)*.

The regular expression for the month should look like this: *((|0)[1-9])|(10)|(11)|(12)*.

The regular expression for the year should look like this: *(((19)|(20))[0-9][0-9])|([0-9][0-9])*.

What is left is to combine all this together and separate the numbers by a period (e.g. 1.03.1999). The period is an auxiliary sign, so we must put a backslash (\) before it. The regular expression for the full date should then look like this:

(((|0)[1-9])|([1|2][0-9])|(30)|(31))\.(((|0)[1-9])|(10)|(11)|(12))\.((((19)|(20))[0-9][0-9])|([0-9][0-9]))

Regular expression for e-mail addresses

You can easily make a language for denoting e-mail addresses. The regular expression for an e-mail address should look like this:

[a-zA-Z0-9_\-\.]+\@[a-zA-Z0-9\.\-]+\.[a-zA-Z]+

See also

Working with Dictionaries
RegExpDictionaryDescription

Recognizing Words with Spaces

ABBYY FineReader Engine allows you to add words with spaces to a dictionary. This feature can be very useful for checking words like “New York.” We recommend using a dictionary for words with spaces when recognizing fields (small areas) on the image. Fields are cut from the document and passed to ABBYY FineReader Engine for recognition as small images, each representing one word.

To recognize words with spaces, do the following:

1. Add the "space" character to the alphabet of the current language.
2. Add the necessary words with spaces to the dictionary.
3. Set the **OneWordPerLine** property of the **RecognizerParams** object to TRUE.

Below is a detailed description of this operation:

1. Create a new text language on the basis of a predefined language. To do this, create a **TextLanguage** object using the **CreateTextLanguage** method of the **Engine** object and copy the attributes of the predefined language.
2. Add the "space" character to the **BaseLanguage** object within **TextLanguage** object, using the **LetterSet** property of the **BaseLanguage** object.
3. Create a new dictionary and add all the necessary words with spaces to this dictionary. You can use the **Dictionary** object to do this.
4. Create a **UserDictionaryDescription** object. Assign the path to the new dictionary to the **FileName** property of this object.
5. Add the **UserDictionaryDescription** object to the **DictionaryDescriptions** collection of the **BaseLanguage** object.
6. In the **RecognizerParams** object of all text blocks, assign the previously created **TextLanguage** object to the **TextLanguage** property, and the TRUE value to the **OneWordPerLine** property.

Sample code in Visual C++ (COM), where the "space" character has been added to the alphabet of the English language, and the word “New York” has been added to the dictionary:

Visual C++ (COM) code

```
...
 // Create a new TextLanguage object
 FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

 // Copy all attributes from the predefined English language
 FREngine::ITextLanguagePtr pEnglishLanguage =
           Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
 pTextLanguage->CopyFrom( pEnglishLanguage );
 pTextLanguage->InternalName = L"SampleTL";

 // Bind new dictionary to the first (and single) BaseLanguage object within
TextLanguage
 FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

 // Change the internal dictionary name to a user-defined name
 pBaseLanguage->InternalName = L"SampleBL";

 // Add the "space" character
 _bstr_t alphabet = pBaseLanguage->GetLetterSet( FREngine::BLLS_Alphabet );
 pBaseLanguage->PutLetterSet( FREngine::BLLS_Alphabet, alphabet + L" " );

 // Create a new dictionary
 _bstr_t dictionaryFile = L"D:\\sample.amd";

 FREngine::IDictionaryPtr pDictionary =
         Engine->CreateNewDictionary( dictionaryFile,
  FREngine::LI_EnglishUnitedStates );
  pDictionary->Name = L"Sample";

 // Add words with space to the dictionary
 pDictionary->AddWord( "New York", 100 );

 // Get the collection of dictionary descriptions and remove all items
 FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
  pBaseLanguage->DictionaryDescriptions;
 pDictionaryDescriptions->RemoveAll();

 // Create a user dictionary description and add it to the collection
 FREngine::IUserDictionaryDescriptionPtr userDic =
  Engine->CreateUserDictionaryDesc();
 // Specify the path to the dictionary which contains words with spaces
 userDic->FileName = dictionaryFile;
 pDictionaryDescriptions->Add( userDic );

 // Specify the properties of the RecognizerParams object of all text blocks
 // Iterate blocks
 for( long i = pLayout->Blocks->Count - 1; i >= 0; i-- ) {
   FREngine::BlockTypeEnum blockType = pLayout->Blocks->Item( i )->Type;
   // Find the text block
   if( blockType != FREngine::BT_Text ) {
     pLayout->Blocks->Remove(i);
   } else {
     pLayout->Blocks->Item(i)->GetAsTextBlock()->RecognizerParams->
  TextLanguage = pTextLanguage;
     pLayout->Blocks->Item(i)->GetAsTextBlock()->RecognizerParams->
  OneWordPerLine = VARIANT_TRUE;
   }
 }
 ...
```

See also

Working with Languages
Working with Dictionaries

Setting up Scanning Options

ABBYY FineReader Engine 10 provides the user control over the scanning parameters via the API. The following parameters are accessible via the ABBYY FineReader Engine 10 API: brightness, color, resolution, the size of the scanning area, duplex scanning mode, a pause between pages and some more. The scanning parameters are set by using the **ScanSourceSettings** property of the **ScanManager** object. This property is required to get to access to the **ScanSourceSettings** object which, in its turn, provides access to the scanning settings of a source. You can find some useful recommendations in the Tips for Document Scanning section.

To set up the scanning parameters:

1. Create a **ScanManager** object.

2. From the **ScanSources** property of the **ScanManager** object, choose the scan source.

3. Create a **ScanSourceSettings** object and initialize it with the settings of the selected scanner.

4. Set the **ScanOptionsInterfaceType** property of the **ScanManager** object to SOIT_None, in this case no interface will be displayed.

5. Tune the scanning options. Select the appropriate values for brightness, resolution, and the other parameters in the corresponding properties of the **ScanSourceSettings** object.

6. Set these scanning parameters in the **ScanSourceSettings** property of the **ScanManager** object.

7. Specify the folder name in which scanned pages will be stored. The folder name should be a **String** variable, for example, **ScanFolder**.

8. Scan paper documents, save the images to the **ScanFolder** folder, and save the full path to the image files to the **StringsCollection** object by using the **Scan** method of the **ScanManager** object**.** You can get the image file names from this **StringsCollection** object and open the files as usual image files by using the methods of the **Engine** object, for example, **OpenImage** or **PrepareAndOpenImage**.

Sample code in C++ and Visual Basic:

Visual C++ (COM) code

```
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Create ScanManager object
FREngine::IScanManagerPtr scanManager = Engine->CreateScanManager();

// Specify the scan source
FREngine::IStringsCollectionPtr sources = scanManager->ScanSources;
_bstr_t scanner = sources->Item( 0 );

// Create a ScanSourceSettings object
FREngine::IScanSourceSettingsPtr scanSettings = scanManager-
>GetScanSourceSettings(scanner);

// Do not display any interface
scanManager->ScanOptionsInterfaceType = FREngine::SOIT_None;

// Tune the scanning options
scanSettings->Resolution = 300;
scanSettings->PictureMode = FREngine::SPM_Grayscale;

// Set up the scanning options
scanManager->PutScanSourceSettings(scanner, scanSettings);

// The name of the folder in which scanned pages will be stored
char scanFolder[MAX_PATH + 1];
```

```
// Scan and save images into scanFolder folder
FREngine::IStringsCollectionPtr scannedImages =
       scanManager->Scan( scanner, scanFolder, VARIANT_FALSE );
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Create a ScanManager object
Dim ScanManager As FREngine.ScanManager
Set ScanManager = Engine.CreateScanManager

' Specify the scan source
Dim Scanner As String
Scanner = ScanManager.ScanSources(0)

' Create a ScanSourceSettings object
Dim ScanSettings As FREngine.ScanSourceSettings
Set ScanSettings = ScanManager.ScanSourceSettings(Scanner)

' Do not display any interface
ScanManager.ScanOptionsInterfaceType = SOIT_None

' Tune the scanning options
ScanSettings.Resolution = 300
ScanSettings.PictureMode = SPM_Grayscale

' Set up the scanning options
ScanManager.ScanSourceSettings(Scanner) = ScanSettings

' The name of the folder in which scanned pages will be stored
Dim ScanFolder As String

' Collection of the full paths to the image files that were received from the scanner
Dim ScannedImages As FREngine.StringsCollection

' Scan paper documents and save the images into the ScanFolder folder
Set ScannedImages = ScanManager.Scan(Scanner, ScanFolder, False)
...
```

See also

ScanManager
ScanSourceSettings
Tips for Document Scanning

Best Practices

This section provides recommendations on how to scan and photograph documents and how to set recognition parameters in order to achieve the best recognition results:

- Tips for Document Scanning
- Tips for Taking Photos
- Improving Recognition Quality

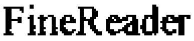
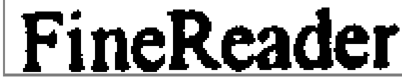
Tips for Document Scanning

Recognition quality depends largely on the quality of the image, which greatly depends on the settings used during the document scanning process.

Font Is Too Small

For optimal recognition results, scan documents printed in very small fonts at higher resolutions.

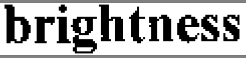
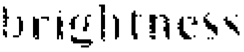
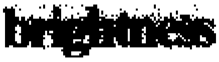
You can specify the desired resolution in the **Resolution** property of the **ScanSourceSettings** object.

Source image	Recommended resolution
 FineReader	300 dpi for typical texts (printed in fonts of size 10 pt or larger)
 FineReader	400-600 dpi for texts printed in smaller fonts (9 pt or smaller)

Tuning Brightness

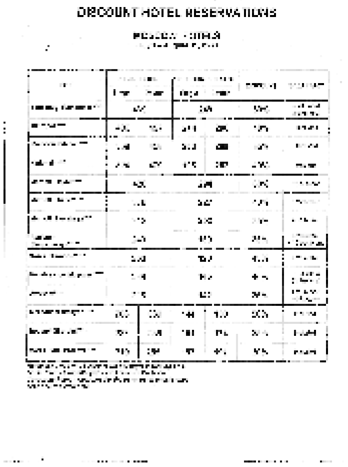
You may need to adjust the brightness setting when scanning in black-and-white mode. You can specify the desired brightness in the **Brightness** property of the **ScanSourceSettings** object. A medium value of around 50% should suffice in most cases.

If the resulting image contains too many "torn" or "stuck" together letters, troubleshoot using the table below.

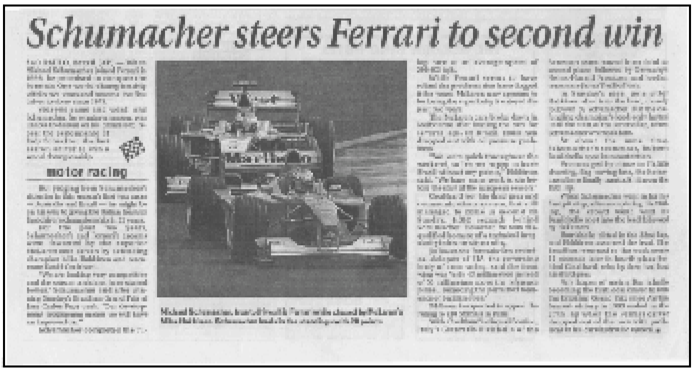
Your image looks like this	Recommendations
 brightness	This image is suitable for recognition.
 brightness characters are "torn" or very light	- Lower the brightness to make the image darker. - Scan in grayscale. Brightness will be tuned automatically.
 brightness characters are distorted, stuck together, or filled out	- Increase the brightness to make the image brighter. - Scan in grayscale. Brightness will be tuned automatically.

Print Quality

Poor-quality documents with "noise" (i.e. random black dots or speckles), blurred and uneven letters, or skewed lines and shifted table borders may require specific scanning settings. For example, this fax and newspaper:



DISCOUNT HOTEL RESERVATIONS



Schumacher steers Ferrari to second win

motor racing

Poor-quality documents are best scanned in grayscale. When scanning in grayscale, the program will select the optimal brightness value automatically.

Grayscale mode retains more information about the letters in the scanned text to achieve better recognition results when recognizing documents of medium to poor quality.

See also

Setting up Scanning Options
Tips for Taking Photos
Improving Recognition Quality

Tips for Taking Photos

Taking photos of documents requires some skill and practice. The characteristics of your camera and shooting conditions are also important.

📝**Note:** For detailed information about the settings of your camera, please refer to the documentation supplied with your camera.

Before taking a picture:

1. Make sure that the page fits entirely within the frame.
2. Make sure that lighting is evenly distributed across the page and that there are no dark areas or shadows.
3. Straighten out the page if required and position the camera parallel to the plane of the document so that the lens looks to the center of the text being photographed.

The topics below outline the required camera specifications and shooting modes.

Digital Camera Requirements

Minimum Requirements

- 2-megapixel sensor
- Variable focus lens (fixed-focus cameras, common in cell phones and hand-held devices, will usually produce images unsuitable for OCR)

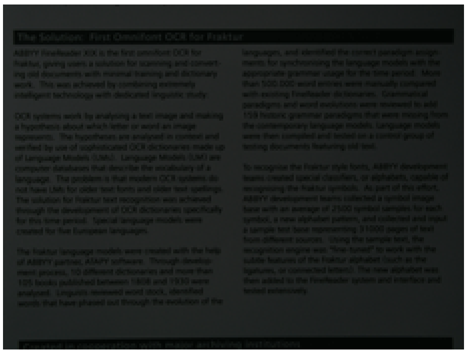
Recommended Requirements

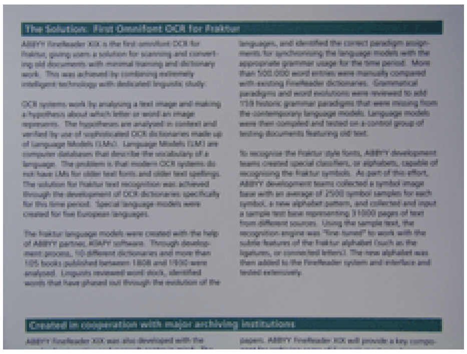
- 5-megapixel sensor
- Flash disable feature
- Manual aperture control or aperture priority mode
- Manual focusing
- An anti-shake system, otherwise the use of a tripod is recommended
- Optical zoom

Shooting Modes

Lighting

Make sure there is enough light (preferably daylight). In artificial lighting, use two light sources positioned so as to avoid shadows.





The Solution: First Omnifont OCR for Fraktur

ABBYY FineReader XIX is the first omnifont OCR for Fraktur, giving users a solution for scanning and converting old documents with minimal training and dictionary work. This was achieved by combining extremely intelligent technology with dedicated linguistic study.

OCR systems work by analysing a text image and making a hypothesis about which letter or word an image represents. The hypotheses are analysed in context and verified by use of sophisticated OCR dictionaries made up of Language Models (LMs). Language Models (LM) are computer databases that describe the vocabulary of a language. The problem is that modern OCR systems do not have LMs for older text fonts and older text spellings. The solution for Fraktur text recognition was achieved through the development of OCR dictionaries specifically for this time period. Special language models were created for five European languages.

The Fraktur language models were created with the help of ABBYY partner, ATAPY software. Through development process, 10 different dictionaries and more than 105 books published between 1808 and 1930 were analysed. Linguists reviewed word stock, identified words that have phased out through the evolution of the languages, and identified the correct paradigm assignments for synchronising the language models with the appropriate grammar usage for the time period. More than 500.000 word entries were manually compared with existing FineReader dictionaries. Grammatical paradigms and word evolutions were reviewed to add 159 historic grammar paradigms that were missing from the contemporary language models. Language models were then compiled and tested on a control group of testing documents featuring old text.

To recognise the Fraktur style fonts, ABBYY development teams created special classifiers, or alphabets, capable of recognising the Fraktur symbols. As part of this effort, ABBYY development teams collected a symbol image base with an average of 2500 symbol samples for each symbol, a new alphabet pattern, and collected and input a sample text base representing 31000 pages of text from different sources. Using the sample text, the recognition engine was "fine-tuned" to work with the subtle features of the Fraktur alphabet (such as the ligatures, or connected letters). The new alphabet was then added to the FineReader system and interface and tested extensively.

Created in cooperation with major archiving institutions

ABBYY FineReader XIX was also developed with the

papers. ABBYY FineReader XIX will provide a key compo-

Positioning the Camera

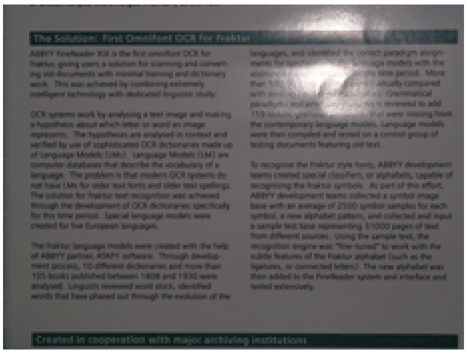
If possible, use a tripod. Position the lens parallel to the plane of the document and point it toward the center of the text.

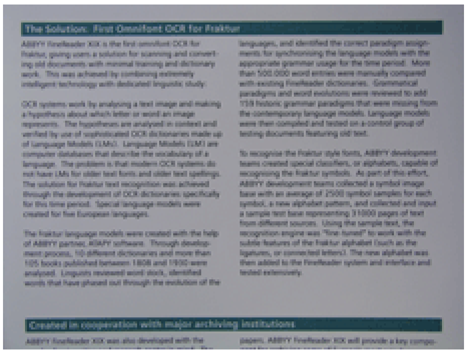
At full optical zoom, the distance between the camera and the document must be sufficient to fit the entire document into the frame. Usually this distance will be 50-60 cm.

Flash

Whenever possible, turn off the flash to avoid glare and sharp shadows on the page. In poor lighting conditions, try using the flash from a distance of about 50 cm, or, preferably, use additional lighting.

⚠**Important**! Using the flash when photographing documents printed on glossy paper causes the worst glare.





The Solution: First Omnifont OCR for Fraktur

ABBYY FineReader XIX is the first omnifont OCR for Fraktur, giving users a solution for scanning and converting old documents with minimal training and dictionary work. This was achieved by combining extremely intelligent technology with dedicated linguistic study.

OCR systems work by analysing a text image and making a hypothesis about which letter or word an image represents. The hypotheses are analysed in context and verified by use of sophisticated OCR dictionaries made up of Language Models (LMs). Language Models (LM) are computer databases that describe the vocabulary of a language. The problem is that modern OCR systems do not have LMs for older text fonts and older text spellings. The solution for Fraktur text recognition was achieved through the development of OCR dictionaries specifically for this time period. Special language models were created for five European languages.

The Fraktur language models were created with the help of ABBYY partner, ATAPY software. Through development process, 10 different dictionaries and more than 105 books published between 1808 and 1930 were analysed. Linguists reviewed word stock, identified words that have phased out through the evolution of the languages, and identified the correct paradigm assignments for synchronising the language models with the appropriate grammar usage for the time period. More than 500,000 word entries were manually compared with existing FineReader dictionaries. Grammatical paradigms and word evolutions were reviewed to add 159 historic grammar paradigms that were missing from the contemporary language models. Language models were then compiled and tested on a control group of testing documents featuring old text.

To recognize the Fraktur style fonts, ABBYY development teams created special classifiers, or alphabets, capable of recognising the Fraktur symbols. As part of this effort, ABBYY development teams collected a symbol image base with an average of 2500 symbol samples for each symbol, a new alphabet pattern, and collected and input a sample text base representing 31000 pages of text from different sources. Using the sample text, the recognition engine was "fine-tuned" to work with the subtle features of the Fraktur alphabet (such as the ligatures, or connected letters). The new alphabet was then added to the FineReader system and interface and tested extensively.

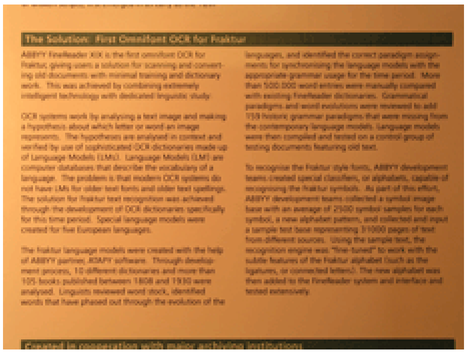
Created in cooperation with major archiving institutions

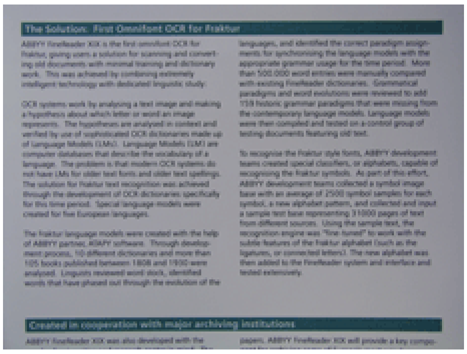
ABBYY FineReader XIX was also developed with the

papers. ABBYY FineReader XIX will provide a key compo-

White Balance

If your camera allows, use a white sheet of paper to set white balance. Otherwise, select the white balance mode which best suits the current lighting conditions.





The Solution: First Omnifont OCR for Fraktur

ABBYY FineReader XIX is the first omnifont OCR for Fraktur, giving users a solution for scanning and converting old documents with minimal training and dictionary work. This was achieved by combining extremely intelligent technology with dedicated linguistic study.

OCR systems work by analysing a text image and making a hypothesis about which letter or word an image represents. The hypotheses are analysed in context and verified by use of sophisticated OCR dictionaries made up of Language Models (LMs). Language Models (LM) are computer databases that describe the vocabulary of a language. The problem is that modern OCR systems do not have LMs for older text fonts and older text spellings. The solution for Fraktur text recognition was achieved through the development of OCR dictionaries specifically for this time period. Special language models were created for five European languages.

The Fraktur language models were created with the help of ABBYY partner, ATAPY software. Through development process, 10 different dictionaries and more than 105 books published between 1808 and 1930 were analysed. Linguists reviewed word stock, identified words that have phased out through the evolution of the languages, and identified the correct paradigm assignments for synchronising the language models with the appropriate grammar usage for the time period. More than 500,000 word entries were manually compared with existing FineReader dictionaries. Grammatical paradigms and word evolutions were reviewed to add 159 historic grammar paradigms that were missing from the contemporary language models. Language models were then compiled and tested on a control group of testing documents featuring old text.

To recognize the Fraktur style fonts, ABBYY development teams created special classifiers, or alphabets, capable of recognising the Fraktur symbols. As part of this effort, ABBYY development teams collected a symbol image base with an average of 2500 symbol samples for each symbol, a new alphabet pattern, and collected and input a sample text base representing 31000 pages of text from different sources. Using the sample text, the recognition engine was "fine-tuned" to work with the subtle features of the Fraktur alphabet (such as the ligatures, or connected letters). The new alphabet was then added to the FineReader system and interface and tested extensively.

Created in cooperation with major archiving institutions

ABBYY FineReader XIX was also developed with the

papers. ABBYY FineReader XIX will provide a key compo-

What do I do if...

There is not enough light

Try the following:

- Select a greater aperture value
- Select a greater ISO value for sensitivity
- Use manual focusing if the camera cannot lock the focus automatically

The picture is too dark and low-contrast

Try using additional light sources. Otherwise, increase the aperture value.

The picture is not sharp enough

Auto focus may not work properly in poor lighting or when photographing at a close distance. In poor lighting conditions, try using an additional light source. When photographing a document up close, try using the Macro (or Close-Up) mode. Otherwise, if possible, focus the camera manually.

If only a part of the picture is blurred, try reducing the aperture value. Increase the distance between the document and the camera and use maximum zoom. Focus on a point anywhere in between the center and a border of the image.

In poor lighting conditions, when shooting in auto mode, the camera will use slower shutter speeds, which makes the resulting photo less sharp. In this case, try the following:

- Enable the anti-shake system, if available.

- Use auto release to prevent the shaking of the camera caused by pressing the shutter release button (even when using a tripod).

The flash causes glare in the center of the picture

Turn off the flash. Otherwise, try photographing from a greater distance.

See also

Tips for Document Scanning

Improving Recognition Quality

Recognition quality depends not only on the quality of the image (see recommendations for scanning and taking photos), but on the recognition settings as well.

Print Type

When recognizing draft dot-matrix printouts or typewritten texts, recognition quality can sometimes be improved by selecting the right text type. You can specify the text type in the **TextTypes** property of the **RecognizerParams** object. By default, the value of this property is TT_Normal, which corresponds to common typographic text. However, you may also select a more specific type.

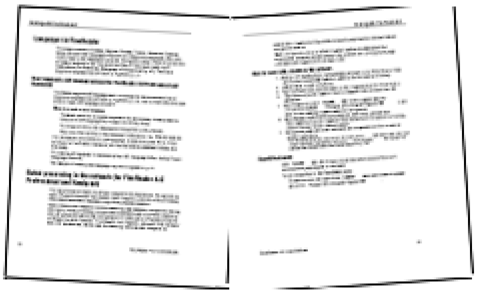
software	An example of typewritten text. All letters are of equal width (compare, for example, "w" and "a"). Select TT_Typewriter for texts of this type.
software	An example of draft dot-matrix text. Character lines are made up of dots. Select TT_Matrix for texts of this type.

Document Languages

ABBYY FineReader Engine recognizes both mono- and multi-lingual (e.g. written in several languages) documents. For multi-lingual documents, you must specify several recognition languages. English is the default recognition language. To change the default recognition language, use the **SetPredefinedTextLanguage** method of the **RecognizerParams** object.

Scanning Facing Pages

When scanning facing pages of a book, both pages will appear as a single image, e.g.



To improve recognition quality, split the facing pages into two separate images. You can find the position where to split the image into pages using the **IFRPage::FindPageSplitPosition** and **IDocumentAnalyzer::FindPageSplitPosition** methods.

When scanning very thick books, the text close to the binding may be distorted. The **IFRPage::RemoveGeometricalDistortions** and **IDocumentAnalyzer::RemoveGeometricalDistortions** methods straighten out distorted lines on an image.

Photo Correction

OCR quality may be affected by distorted text lines close to the margins, by document skew, by noise, and other defects commonly found on digital photos. A set of photo correction methods allows you to straighten out text lines, remove motion blur, and reduce noise:

- to straighten out distorted lines on an image, use the **IFRPage::RemoveGeometricalDistortions** and **IDocumentAnalyzer::RemoveGeometricalDistortions** methods

- to remove motion blur, use the **IImageDocument::RemoveCameraBlur** method

- to reduce noise, use the **IImageDocument::RemoveCameraNoise** method

See also

Tips for Document Scanning
Tips for Taking Photos

Description of the ABBYY FineReader Engine Samples

The ABBYY FineReader Engine 10 distribution pack contains a set of sample source code showing how to use Engine in different scenarios. This section contains a short description of these samples. The detailed description of the samples can be found in the **Code Samples Library** provided with this distribution pack (**Start > Programs > ABBYY FineReader Engine 10 > Code Samples Library**).

The ABBYY FineReader Engine samples are provided for Visual Basic, Visual Basic .Net, Delphi, raw C++, C++ with the Native COM Support, C#, and script languages. For each of these developer platforms, a similar set of samples is provided.

All sample code can be found in:

- **%ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\FineReader Engine** — for Windows 2000, Windows XP, Windows Server 2003;

- **%ProgramData%\ABBYY\SDK\10\FineReader Engine** — for Windows Vista, Windows Server 2008, Windows 7.

Sample Name	**Description**
Hello	Performs document conversion with just a few lines of code. This sample will help you to start development using the ABBYY SDK.
RecognizedTextProcessing	Calculates recognition statistics (e.g. the number of suspicious characters and rejects, the number of words which are not in the dictionary).
EventsHandling	Illustrates the use of the callback interfaces using the **FRDocument** callback interface (**IFRDocumentEvents**) as an example. The sample shows the progress of recognition and export during image processing.
CustomLanguage	Creates a new recognition language and changes its dictionary to a manually-created sample dictionary. After recognition, it calculates the number of words in the text and how many of them were found in the user dictionary.
CommandLineInterface	*Exists only for raw C++*. Provides the command line interface of ABBYY FineReader Engine. The sample produces a CommandLineInterface utility, which supports most of the ABBYY FineReader Engine API functions through numerous keys.
FRECOMWrapper	*Exists only for C++ (Native COM support)*. Provides an easy way to use ABBYY FineReader Engine from script languages. The sample produces FREngineWrap.dll, which can be used to get an ABBYY FineReader Engine object from a script language such as VBScript or JavaScript.
PDFExportProfiles	*Exists only for C#*. Shows the advantages of using export profiles during export to PDF.
MultiProcessingRecognition	*Exists only for C#*. Shows the gain in speed when using multi-processing recognition.

API Reference

Alphabetical List of the ABBYY FineReader Engine 10 Objects and Interfaces

The ABBYY FineReader Engine functionality is implemented in a number of objects that provide methods for working with images, page layout and blocks, languages and recognized text. The interfaces of the ABBYY FineReader Engine objects are derived from **IDispatch**, that is they support early and late binding.

⚠**Important!** Pointers to child object's interfaces are valid until the parent object exists. An attempt to access a child object after its parent object has been destroyed may result in error. Please, see for details **Working with Properties**.

Name	**Description**
Artefact	Exposes properties of an artefact.
BackgroundLayer	Exposes properties of a background layer of a page.
BarcodeBlock	Provides access to specific properties of the barcode block.
BarcodeParams	Allows you to tune the parameters of barcode block recognition.
BarcodeSymbol	Provides access to the properties of one character of a recognized barcode.
BarcodeText	Represents a text of a recognized barcode as a collection of characters.
BaseLanguage	Represents a base recognition language.
BaseLanguages	This object is a collection of base languages (**BaseLanguage** objects).
Block	This object represents a single block.
Caption	Provides access to specific properties of a table or picture caption.
Captions	Provides access to the collection of captions of a table or picture (**Caption** objects).
CharacterRecognitionVariant	This object represents a variant of a character's recognition.
CharacterRecognitionVariants	This object represents a collection of variants of a character's recognition (**CharacterRecognitionVariant** objects).
CharParams	Allows you to access different parameters of a single character in the recognized text.
CheckmarkBlock	This object provides access to specific properties of a checkmark block.
CheckmarkGroup	This object exposes methods and properties for working with a group of checkmarks.
Dictionary	This object is designed for working with user-defined dictionaries.
DictionaryDescription	This object is a dictionary description which may be typecast to one of its child objects: **StandardDictionaryDescription**, **UserDictionaryDescription** or **RegExpDictionaryDescription**.
DictionaryDescriptions	This object is a collection of dictionaries.
DocumentAnalyzer	This object exposes a set of analysis and recognition functions.
DocumentContentInfo	This object contains information about author, keywords, subject, title of the document and stores document information dictionary.
DocumentElement	Provides access to one element of the document stream.
DocumentInfo	Stores service information about the open PDF file.
DocumentInformationDictionary	Represents a document information dictionary which contains metadata from the PDF file.
DocumentInformationDictionaryItem	This object is an element of a document information dictionary.

DocumentSection	Represents one logical section of the document.
DocumentStream	Provides access to one document stream.
DocumentStructure	Provides access to the logical structure of the document.
DocumentStructureDetectionParams	This object is used for setting up the parameters of the document structure detection during document synthesis.
Engine	This is the main ABBYY FineReader Engine object.
EnumDictionaryWords	Serves for iterating words included in a user-defined dictionary.
Exporter	Provides tools for saving recognized text into files in external formats.
ExternalDictionaryCallback	This is a callback interface which is used to deliver information about dictionary words to the recognizer.
ExternalDictionaryDescription	Provides access to an external dictionary.
FontFormattingDetectionParams	This object is used for setting up the parameters of font formatting detection during document synthesis.
FontFormattingDetectionParamsForPage	Specifies the parameters of font formatting detection at the stage of page synthesis.
FontStyle	Exposes properties of a font style.
Footnote	Exposes properties of a footnote.
FootnoteSeries	Stores the parameters of one series of footnotes.
FootnoteSeriesArray	Represents an array of footnote series.
FRDocument	Corresponds to a document which may contain several pages.
FRPage	Corresponds to a page of document.
FRPages	This object is a collection of document pages.
FRRectangle	Represents a location and size of a rectangle.
FuzzyString	Represents a fuzzy string.
FuzzyStringsCollection	Collection of the **FuzzyString** objects.
GlobalStyleStorage	Provides access to the styles of the document.
HTMLExportParams	Provides functionality for tuning parameters of recognized text export in HTML format.
Hyperlink	This object represents a hyperlink.
IDocumentAnalyzerEvents	This is callback interface that is used for reporting events from the **DocumentAnalyzer** object to the listeners.
IExporterEvents	This is callback interface that is used for reporting events from the **Exporter** object to the listeners.
IExternalDictionary	This is interface for external dictionary.
IFRDocumentEvents	This is callback interface that is used for reporting events from the **FRDocument** object to the listeners.
IFRPageEvents	This is callback interface that is used for reporting events from the **FRPage** object to the listeners.
IFRPagesEvents	This is callback interface that is used for reporting events from the **FRPages** object to the listeners.
IImageDocumentEvents	This is callback interface that is used for reporting events from the **ImageDocument** object to the listeners.
IImagePasswordCallback	This is callback interface that is used for processing password-protected image files
Image	This object represents a single "image plane" of an open image.

ImageDocument	This object corresponds to an open image.
ImageDocumentsCollection	Collection of **ImageDocument** objects.
ImageModification	This object is used to store parameters of image modification.
ImageProcessingParams	Specifies how an image will be preprocessed before analysis and recognition.
Incut	Exposes method and properties of an incut.
IRecognizedPages	This interface contains properties and methods necessary for passing recognized texts and images of the pages to be exported, one-by-one.
IScanManagerEvents	This is callback interface that is used for interaction of the **ScanManager** object with its listeners.
JpegExtendedParams	This object is used to store parameters used when saving images in JPEG format.
LanguageDatabase	Provides means for performing advanced operations with recognition languages.
Layout	Exposes methods and properties for working with the image layout.
LayoutBlocks	This object represents a collection of blocks (**Block** objects).
LayoutsCollection	Collection of **Layout** objects.
License	Provides access to the current license parameters.
LicenseCollection	Provides access to a collection of available (activated) licenses.
List	Represents one list template. It is a collection of list levels (**ListLevel** objects).
ListLevel	Provides access to the parameters of one level of a list.
ListParams	Provides access to the parameters of the list to which a paragraph belongs.
LongsCollection	Collection of **long** type variables.
MainText	Exposes method and properties of a main text.
MultipageImageWriter	This object is used for saving several images into a single image file.
MultiProcessingParams	Provides access to the parameters of multiple CPU cores usage.
ObjectsExtractionParams	Provides access to parameters used for objects extraction.
OrientationDetectionParams	Provides access to the parameters used for tuning the page orientation detection.
PageAnalysisParams	Provides access to parameters used for tuning the page layout analysis process.
PageBlackSeparator	Represents a single page black separator.
PageElement	Represents an element of a recognized page.
PageElements	Collection of page elements (**PageElement** objects).
PageProcessingParams	This object is used for tuning different parameters of layout analysis and recognition.
PageSection	Represents one page section.
PageSections	Collection of page sections (**PageSection** objects).
PageStream	Represents a page stream.
PageStreams	Collection of page streams (**PageStream** objects).
PageStructure	Provides access to the logical structure of the page.
Paragraph	Exposes methods and properties for working with a single paragraph of the recognized text.
ParagraphLine	Represents a single line in the paragraph of a recognized text.
ParagraphLines	Represents a collection of paragraph lines.
ParagraphParams	This object exposes extended properties of a single paragraph.

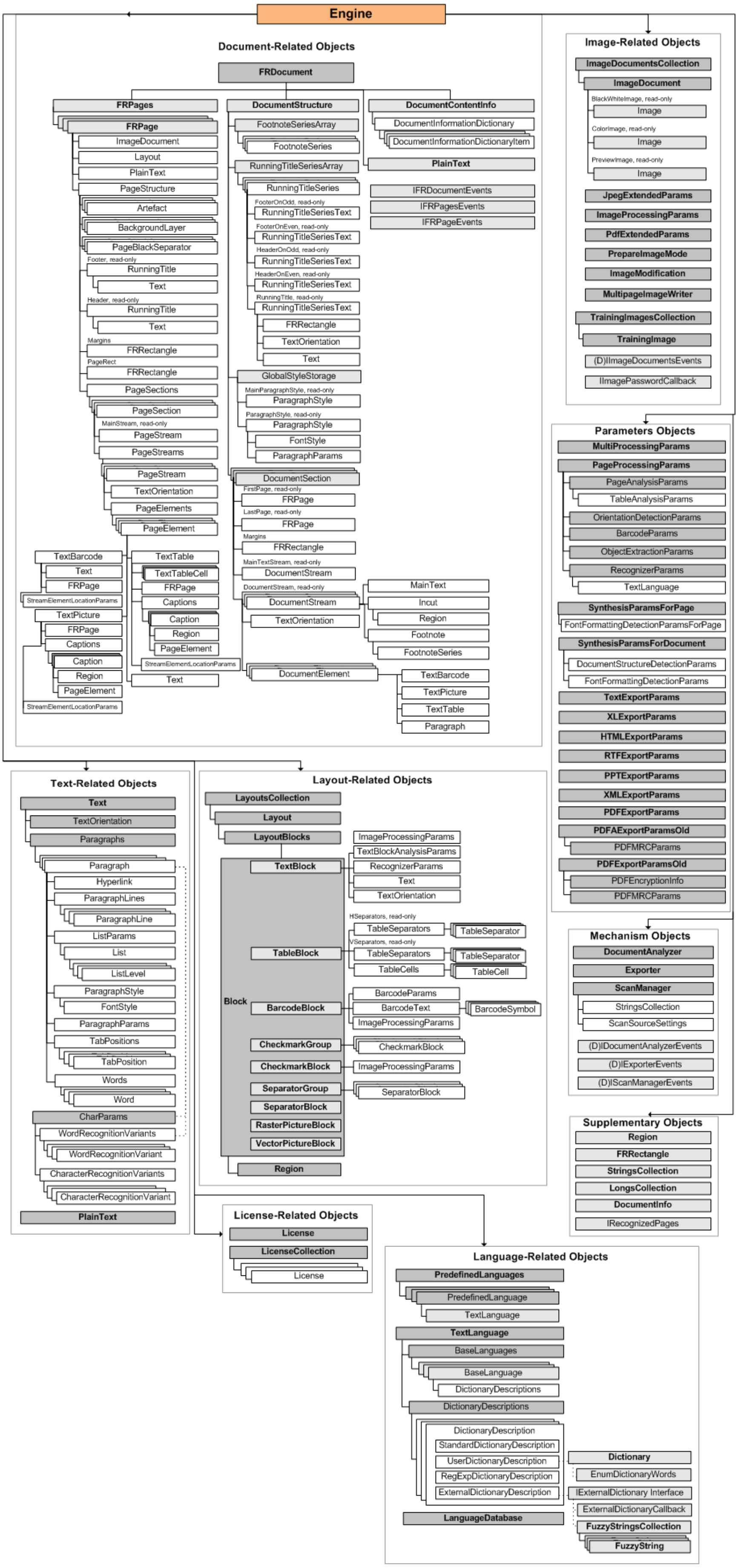
Paragraphs	This object represents collection of paragraphs of the recognized text.
ParagraphStyle	Exposes properties of the paragraph style.
PDFAExportParamsOld	Provides functionality for tuning parameters of recognized text export in PDF/A format. This object is obsolete, we recommend you to use the **PDFExportParams** object.
PDFEncryptionInfo	This object provides access to encryption parameters of the PDF file during export.
PDFExportParams	Provides functionality for tuning parameters of recognized text export in PDF (PDF/A) format.
PDFExportParamsOld	Provides functionality for tuning parameters of recognized text export in PDF format. This object is obsolete, we recommend you to use the **PDFExportParams** object.
PdfExtendedParams	Provides functionality for tuning the parameters of saving an image to PDF format
PDFMRCParams	Provides functionality for tuning Mixed Raster Content (MRC) parameters of the PDF (PDF/A) file during export.
PlainText	Represents recognized text without formatting.
PPTExportParams	Provides functionality for tuning parameters of recognized text export in PPTX format.
PredefinedLanguage	Represents a single predefined recognition language.
PredefinedLanguages	Represents ABBYY FineReader Engine predefined languages collection.
PrepareImageMode	Contains different attributes specifying how an image will be prepared during conversion to internal format.
RasterPictureBlock	Provides access to specific properties of the raster picture block.
RecognizerParams	Allows you to tune parameters of text recognition.
RegExpDictionaryDescription	This object provides access to a regular-expression-based dictionary.
Region	This supplementary object is designed to store the information on ABBYY FineReader Engine block's region.
RTFExportParams	Provides functionality for tuning parameters of recognized text export in RTF/DOC/DOCX format.
RunningTitle	Provides access to a single header or footer on a page.
RunningTitleSeries	Stores the parameters of one series of running titles.
RunningTitleSeriesArray	Represents an array of running title series (**RunningTitleSeries** objects).
RunningTitleSeriesText	Provides access to the text of the running title series.
ScanManager	Exposes a set of properties and methods required to perform scanning.
ScanSourceSettings	Provides access to the scanning settings of a source.
SeparatorBlock	Provides access to specific properties of a separator block.
SeparatorGroup	Represents a group of separator blocks (**SeparatorBlock** objects).
StandardDictionaryDescription	Provides access to a standard dictionary.
StreamElementLocationParams	Allows you to locate a text element in a column.
StringsCollection	Collection of strings.
SynthesisParamsForDocument	This object is used for setting up the parameters of the document synthesis.
SynthesisParamsForPage	This object is used for setting up the parameters of the page synthesis.
TableAnalysisParams	Provides access to parameters affecting table block analysis process.
TableBlock	Provides access to specific properties of a table block.
TableCell	Represents a single table cell in a table block.

TableCells	Represents collection of table cells of a table block.
TableSeparator	Represents a single table separator in a table block.
TableSeparators	This object is a collection of table block separators.
TabPosition	Stores information about all tab stops in a single paragraph.
TabPositions	Provides access to all the tab stops in a single paragraph.
Text	This object represents recognized text.
TextBarcode	Provides access to specific properties of a barcode in a logic structure of a document.
TextBlock	Provides access to specific properties of the text block.
TextBlockAnalysisParams	Specifies how a text block should be analyzed.
TextExportParams	Provides functionality for tuning parameters of recognized text export in TXT and CSV formats.
TextLanguage	Represents the language of recognition for a text.
TextOrientation	Represents a text orientation.
TextPicture	Provides access to specific properties of a picture in a logic structure of a document.
TextTable	Provides access to specific properties of a table in a logic structure of a document.
TextTableCell	Provides access to specific properties of a table cell in a logic structure of a document.
TrainingImage	Represents a single training image.
TrainingImagesCollection	Collection of **TrainingImage** objects.
UserDictionaryDescription	This object provides access to a user dictionary.
VectorPictureBlock	Provides access to specific properties of a vector picture block.
Word	This object represents a word.
WordRecognitionVariant	This object represents a variant of a word's recognition.
WordRecognitionVariants	This object represents a collection of variants of a word's recognition (**WordRecognitionVariant** objects).
Words	This object represents a collection of words (**Word** objects).
XLExportParams	Provides functionality for tuning parameters of recognized text export in XLS/XLSX format.
XMLExportParams	Provides functionality for tuning parameters of recognized text export in XML format.

See also

Object Diagram

ABBYY FineReader Engine 10 Object Diagram



GetEngineObject Function

This function is the only means to get a pointer to the **IEngine** interface. It takes as an input parameter a serial number of a Developer License. You may pass the necessary serial number to this function, or select the serial number later.

Visual Basic Syntax

```
Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                         ByVal developerSN As String, _
                         ByVal reserved1 As String, _
                         ByVal reserved2 As String, _
                         engine As FREngine.Engine) As Long
```

C++ Syntax

```
HRESULT __stdcall GetEngineObject(
                         BSTR developerSN,
                         BSTR reserved1,
                         BSTR reserved2,
                         IEngine** engine);
```

Parameters

developerSN

[in] A string containing developer serial number that matches the necessary license. This parameter may be 0 or an empty string. In this case, you must select the current license later using the **IEngine::SetCurrentLicense** method.

reserved1

[in] Reserved for future use. The empty string "" or **Null** pointer must be passed.

reserved2

[in] Reserved for future use. The empty string "" or **Null** pointer must be passed.

engine

[out] A pointer to **IEngine*** pointer variable that receives the interface pointer to the resulting **Engine** object.

Return Values

The function may return standard return values of ABBYY FineReader Engine functions.

Remarks

The **Engine** object is a singleton, so only one object of this type may be created in a single instance of the application that uses ABBYY FineReader Engine. Repeated attempts to create the **Engine** object will return the same object. It is important to notice that the process of creation of the **Engine** object takes rather long time because it requires loading not only the FREngine.dll, but a whole set of other DLLs.

It is possible to create and run the Engine object on a multi–processor system, but there can be only one **Engine** object in each process. A second call of **GetEngineObject** within the same process will return the reference to the existing object. Therefore, you should create a separate **Engine** object for each process by calling the **GetEngineObject** function.

It is prohibited to initialize ABBYY FineReader Engine at the entry points of other DLLs, and also in constructors and destructors of static and global objects implemented in DLLs, because they are called at the DLL entry points. This restriction is due to the fact that the Win32 **LoadLibrary** function is not re–entrant. A user should initialize ABBYY FineReader Engine elsewhere, for example, in **WinMain** function of an EXE module.

During initialization ABBYY FineReader Engine will reset the LC_CTYPE setting of msvcrt.dll to operating system defaults. This fact should be taken into account if your application depends upon the msvcrt.dll locale–dependent services.

The **GetEngineObject** function creates the **Engine** object no matter the serial number specified or not. If the serial number was not specified, you can receive the collection of available activated licenses (both local licenses activated on this current computer and network licenses) with the help of the **IEngine::Licenses** property. Then you can view license properties and select the current license using the **IEngine::SetCurrentLicense** method. While no activated license is specified as current, only the **StartLogging**, **StopLogging** methods and the **CurrentLicense** and **Licenses** properties of the **Engine** object are available. Other methods of the **Engine** object will return CLASS_E_NOTLICENSED error code.

Sample

Visual C++ (COM) code

```
HMODULE m_libraryHandle;
 IEngine* m_enginePtr;
 BOOL LoadEngine( const CHAR* filePath, const WCHAR* developerSerialNumber )
 {
    if( m_enginePtr != NULL )
       return TRUE;
    if( m_libraryHandle == NULL ) {
       // load ABBYY FineReader Engine main DLL module
       m_libraryHandle = ::LoadLibraryEx( filePath, 0, LOAD_WITH_ALTERED_SEARCH_PATH );
       if( m_libraryHandle == NULL )
          return FALSE;
    }
    // Obtain ABBYY FineReader Engine main object
    typedef HRESULT (STDAPICALLTYPE* GetEngineFuncType)( const WCHAR*, const WCHAR*,
  const WCHAR*, IEngine** );
    GetEngineFuncType getEngineFunc = (GetEngineFuncType)::GetProcAddress(
  m_libraryHandle, "GetEngineObject" );
    if( getEngineFunc == NULL || getEngineFunc( developerSerialNumber,
  NULL, NULL, &m_enginePtr ) != S_OK ) {
       UnloadEngine();
       return FALSE;
    }
    return TRUE;
 }
```

Visual Basic code

```
Public Engine As FREngine.Engine
 Private Declare Function GetEngineObject Lib "FREngine.dll" ( _
                ByVal DeveloperSN As String, _
                ByVal Reserved1 As String, _
                ByVal Reserved2 As String, _
                EngineObj As FREngine.Engine) As Long
 Sub Engine_Load(ByVal DeveloperSN As String)
     ' Visual Basic may load libraries from the current path only
     ChDir "Path to the folder with FREngine.dll"

     ' this conversion is needed to pass a Unicode string as a DLL function parameter
correctly
     Dim DeveloperSN_WideChar As String
     DeveloperSN_WideChar = StrConv(DeveloperSN, vbUnicode)
     If GetEngineObject(DeveloperSN_WideChar, "", "", Engine) <> 0 Then
         MsgBox "Error loading ABBYY FineReader Engine"
     End If
 End Sub
```

See also

Licensing
Modules
DeinitializeEngine

See samples: Hello, RecognizedTextProcessing, CustomLanguage, EventsHandling

DeinitializeEngine function

This function deinitializes ABBYY FineReader Engine 10. It is called automatically when all the references to ABBYY FineReader Engine API objects are released, so there is no need to call it manually except for debug purposes (to determine which objects are not released).

Visual Basic Syntax

```
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
```

C++ Syntax

```
HRESULT __stdcall DeinitializeEngine();
```

Return Values

This function returns E_FAIL if not all objects are released. In that case you can get the list of not released objects using the **IEngine::StartLogging** method. The function may return standard return values of ABBYY FineReader Engine functions.

Remarks

It is prohibited to deinitialize ABBYY FineReader Engine at the entry points of other DLLs, and also in constructors and destructors of static and global objects implemented in DLLs, because they are called at the DLL entry points. This restriction is due to the fact that the Win32 **FreeLibrary** function is not re–entrant. A user should deinitialize ABBYY FineReader Engine elsewhere, for example, in **WinMain** function of an EXE module.

Sample

Visual C++ (COM) code

```
HMODULE m_libraryHandle;
IEngine* m_enginePtr;
BOOL UnloadEngine()
{
   if( m_libraryHandle == NULL )
      return TRUE;
   // Free Engine object
   if( m_enginePtr != NULL ) {
      m_enginePtr->Release();
      m_enginePtr = 0;
   }
   // Deinitialize the Engine
   typedef HRESULT (STDAPICALLTYPE* UnloadFuncType)();
   UnloadFuncType unloadFunc = (UnloadFuncType)::GetProcAddress(
             m_libraryHandle, "DeinitializeEngine" );
   if( unloadFunc == NULL || unloadFunc() != S_OK )
      return FALSE;
   // Now we can free library safely
   ::FreeLibrary( m_libraryHandle );
   m_libraryHandle = NULL;

   return TRUE;
}
```

Visual Basic code

```
Public Engine As FREngine.Engine
Private Declare Function DeinitializeEngine Lib "FREngine.dll" () As Long
Sub Engine_Unload()
    Set Engine = Nothing
    ChDir "Path to the folder with FREngine.dll"
    DeinitializeEngine
End Sub
```

See also

GetEngineObject

See samples: Hello, RecognizedTextProcessing, CustomLanguage, EventsHandling

Engine Object (IEngine Interface)

This object is the top object in the hierarchy of ABBYY FineReader Engine objects. It exposes a set of creation, analysis, recognition, and export methods. Its properties reflect the global settings of ABBYY FineReader Engine.

The **Engine** object is the only externally creatable object among the ABBYY FineReader Engine objects. To create this object, use the **GetEngineObject** function. The **Engine** object is a singleton, so only one object of this type may be created in a single instance of the application that uses ABBYY FineReader Engine. Repeated attempts to create the **Engine** object will return the same object.

After you receive a reference to the **Engine** object, you can:

- Set the parameters of ABBYY FineReader Engine, such as the user interface language, the parent window of the client application, application title, etc. Use the **properties** of the **Engine** object.
- Load the most suitable settings for your scenario, which are provided in a set of predefined profiles. To load a profile, use the **LoadPredefinedProfile** method.
- Proceed by creating a **FRDocument** object. This object corresponds to a document and exposes the main recognition functionality of ABBYY FineReader Engine. The object allows you to process multi-page documents easily. To create this object, use the **CreateFRDocumentFromImage** or **CreateFRDocument** method.
- Create some additional ABBYY FineReader Engine objects with the help of **creation methods**.
- Use additional services of ABBYY FineReader Engine via **supplementary methods**.
- Use the **processing methods** of the **Engine** object. These methods are suitable only for working with one-page documents. For multi-page documents, ABBYY FineReader Engine provides a more convenient way of processing. We recommend that you create an **FRDocument** object and use its methods and properties for processing.

See also

Properties
Creation methods
Supplementary methods
Processing methods
Object diagram

See samples: Hello, RecognizedTextProcessing, CustomLanguage, EventsHandling, FRECOMWrapper

Properties of the Engine Object

The **Engine** object exposes a set of properties reflecting global settings of ABBYY FineReader Engine. With the help of these properties you can set the parameters of ABBYY FineReader Engine such as user interface language, parent window of the client application, application title, etc.

- In certain cases, such as training and editing of a user pattern, or editing of a user dictionary, ABBYY FineReader Engine may display dialogs and message boxes. Messages and other text in these dialogs, as well as error description strings (**IErrorInfo** object), recognition tips, etc., will be written in the specified user interface language. See the description of the **MessagesLanguage** property.
- The parent window is the window that serves as parent for dialogs and message boxes. Assign the handle of the main application window to this property. ABBYY FineReader Engine uses the standard MFC procedure to find the most suitable parent window for popup windows. Keep it in mind to initialize the parent window handle with correct value or ABBYY FineReader Engine may not perform correctly. See the description of the **ParentWindow** property.
- The application title is the name of the application that uses ABBYY FineReader Engine. This title serves as the caption of message boxes. See the description of the **ApplicationTitle** property.

⚠**Important!** Pointers to child object's interfaces are valid until the parent object exists. An attempt to access a child object after its parent object has been destroyed may result in error. Please, see for details **Working with Properties**.

Name	**Type**	**Description**
ApplicationTitle	**String**	Assign the name of your application to this parameter. It will be used as the title for message boxes.
CurrentLicense	**License**, read-only	Returns the current license.

CreateImageDocumentsInMemory	**Boolean**	Specifies if ABBYY FineReader Engine should create the **ImageDocument** objects in memory. If this property is set to TRUE, the **ImageDocument** objects are saved in the memory, otherwise the objects are saved to files on disk. Set this property to FALSE when processing many **ImageDocument** objects, in which case it decreases memory usage. This property is TRUE by default.
Licenses	LicenseCollection, read-only	Returns a collection of available (activated) licenses.
MessagesLanguage	**MessagesLanguageEnum**	Defines the language of interaction between ABBYY FineReader Engine and the user. All message boxes, error messages, and recognizer tips will be in this language. This parameter stays on between sessions. In order that interface language changes are fully applied, either specify required value for this property and reload the Engine object, or in the registry modify data of the **HKEY_CURRENT_USER\Software\ABBYY\SDK\10\FineReader Engine\InterfaceLanguage** value. The value data can be one of the **MessagesLanguageEnum** enumerator values. **Note:** The locale for the selected messages language must be installed on the computer.
MultiProcessingParams	MultiProcessingParams, read-only	Provides access to the parameters of multiprocessing and multiple CPU cores usage.
ParentWindow	**Long**	Stores HWND handle — casted to **Long** — of the main window of an application which uses ABBYY FineReader Engine. This parameter is used to correctly process dialogs and message boxes. You may change this parameter at any time or not set it at all. ABBYY FineReader Engine uses the standard MFC procedure for finding the main window. If the main window owns any popup windows, the last active popup will be used as the parent window rather than the window specified by this property. If you do not set a value for this property, the procedure of finding the main window may fail, and then ABBYY FineReader Engine will perform incorrectly.
Path	**String**, read-only	Returns the path to the folder that contains the ABBYY FineReader Engine executables.
PredefinedLanguages	PredefinedLanguages, read-only	Provides access to the collection of predefined languages of ABBYY FineReader Engine.
RecognitionSpeedLimit	**Long**	Specifies recognition speed limitation in characters per second. It allows you to specify maximum recognition speed. It may be set to 0, which means that there is no limit on recognition speed. Recognition speed can be limited in the license. In this case, the minimum value is used.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object
FREngine::IEnginePtr Engine;
// Set parent window and caption for Engine's dialog and message boxes
Engine->ParentWindow = ( long ) hDlg;
Engine->ApplicationTitle = L"Hello";
```

Visual Basic code

```
' Global ABBYY FineReader Engine object
Public Engine As FREngine.Engine
' Set parent window and caption for Engine's dialog and message boxes
Engine.ParentWindow = Me.hWnd
Engine.ApplicationTitle = "Hello"
```

See also

Engine
Working with Properties

See samples: Hello, RecognizedTextProcessing, EventsHandling, CustomLanguage

Creation Methods of the Engine Object

The **Engine** object exposes the methods which create other ABBYY FineReader Engine objects.

Note: If you work with programming languages which do not have garbage collections (for example, Ñ++), you must either use smart pointer classes (see the samples in C++(COM)) or release objects that were created by creation methods when they are no longer needed. Prior to the Engine deinitialization, you must release all created objects. Otherwise, the **DeinitializeEngine** function returns E_FAIL.

Name	Description
CreateBarcodeParams	Creates the **BarcodeParams** object.
CreateBaseLanguage	Creates the **BaseLanguage** object.
CreateCharParams	Creates the **CharParams** object.
CreateDocumentAnalyzer	Creates the **DocumentAnalyzer** object.
CreateDocumentInfo	Creates the **DocumentInfo** object.
CreateDocumentInformationDictionary	Creates the DocumentInformationDictionary object.
CreateEmptyUserPattern	Creates an empty user pattern file (*.ptn) at the specified location.
CreateExporter	Creates the **Exporter** object.
CreateExternalDictionaryDesc	Creates the ExternalDictionaryDescription object.
CreateFRDocument	Creates the **FRDocument** object.
CreateFRDocumentFromImage	Opens image file and creates the **FRDocument** object.
CreateHTMLExportParams	Creates the **HTMLExportParams** object.
CreateHyperlink	Creates the **Hyperlink** object.
CreateImageDocumentsCollection	Creates the ImageDocumentsCollection object.
CreateImageModification	Creates the **ImageModification** object.
CreateImageProcessingParams	Creates the ImageProcessingParams object.
CreateJpegExtendedParams	Creates the **JpegExtendedParams** object.
CreateLanguageDatabase	Creates the **LanguageDatabase** object.
CreateLayout	Creates the **Layout** object.
CreateLayoutBlocks	Creates the **LayoutBlocks** object.
CreateLayoutsCollection	Creates the **LayoutsCollection** object.
CreateLongsCollection	Creates the **LongsCollection** object.
CreateMultipageImageWriter	Creates a **MultipageImageWriter** object that may be used for saving several images into a single multipage image file.
CreateNewDictionary	Creates a new empty user dictionary at the specified location and returns interface pointer of the **Dictionary** object associated with it.
CreateObjectsExtractionParams	Creates the ObjectsExtractionParams object.
CreateOrientationDetectionParams	Creates the OrientationDetectionParams object.
CreatePageAnalysisParams	Creates the **PageAnalysisParams** object.
CreatePageProcessingParams	Creates the **PageProcessingParams** object.
CreateParagraphParams	Creates the **ParagraphParams** object.

CreatePDFAExportParamsOld	Creates the **PDFAExportParamsOld** object.
CreatePDFEncryptionInfo	Creates the **PDFEncryptionInfo** object.
CreatePDFExportParams	Creates the **PDFExportParams** object.
CreatePDFExportParamsOld	Creates the **PDFExportParamsOld** object.
CreatePdfExtendedParams	Creates the **PdfExtendedParams** object.
CreatePPTExportParams	Creates the **PPTExportParams** object.
CreatePrepareImageMode	Creates the **PrepareImageMode** object.
CreateRecognizerParams	Creates the **RecognizerParams** object.
CreateRectangle	Creates the **FRRectangle** object.
CreateRegExpDictionaryDesc	Creates the **RegExpDictionaryDescription** object.
CreateRegion	Creates the **Region** object.
CreateRTFExportParams	Creates the **RTFExportParams** object.
CreateScanManager	Creates the **ScanManager** object.
CreateStandardDictionaryDesc	Creates the **StandardDictionaryDescription** object.
CreateStringsCollection	Creates the **StringsCollection** object.
CreateSynthesisParamsForDocument	Creates the **SynthesisParamsForDocument** object.
CreateSynthesisParamsForPage	Creates the **SynthesisParamsForPage** object.
CreateTableAnalysisParams	Creates the **TableAnalysisParams** object.
CreateTextExportParams	Creates the **TextExportParams** object.
CreateTextOrientation	Creates the **TextOrientation** object.
CreateTextLanguage	Creates the **TextLanguage** object.
CreateTrainingImage	Creates the **TrainingImage** object.
CreateTrainingImagesCollection	Creates the **TrainingImagesCollection** object.
CreateUserDictionaryDesc	Creates the **UserDictionaryDescription** object.
CreateXLExportParams	Creates the **XLExportParams** object.
CreateXMLExportParams	Creates the **XMLExportParams** object.

See also

Engine

See samples: Hello

Creation Methods of the Engine Object

The **Engine** object contains a number of methods that create other objects of the ABBYY FineReader Engine objects hierarchy. They all have similar semantics. All newly created objects have default values, or, if a profile has been previously loaded, the values set by this profile are used.

Visual Basic Syntax

```
Method Create<ObjectName>(
) As <ObjectName>
```

C++ Syntax

```
HRESULT Create<ObjectName>(
  I<ObjectName>** result
);
```

Parameters

result

[out] A pointer to ***I<ObjectName>**** pointer variable that receives the interface pointer of the created object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Create Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Create the Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()
...
```

See also

Engine

See samples: Hello, CustomLanguage, RecognizedTextProcessing, EventsHandling

CreateEmptyUserPattern Method of the Engine Object

This method creates an empty user pattern file (*.ptn) at the specified location.

Visual Basic Syntax

```
Method CreateEmptyUserPattern(
  fileName As String
)
```

C++ Syntax

```
HRESULT CreateEmptyUserPattern(
  BSTR fileName
);
```

Parameters

fileName

[in] This variable contains the full path to the newly created user pattern file, e.g. "C:\pattern.ptn".

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- User patterns are files that specify a number of pairs "image of a character — the character itself". User patterns may be used during recognition to identify characters of non–standard fonts. To use a user pattern during recognition, specify a path to it in the **IRecognizerParams::UserPatternsFile** property. User patterns may be edited by user via the **IEngine::EditUserPattern** method or trained during recognition. See Recognizing with Training.

- The method is enabled only if the license supports the User Patterns Training module.

- Pattern training is not supported for hieroglyphic languages.

See also

Recognizing with Training
Engine
IEngine::EditUserPattern

CreateFRDocumentFromImage Method of the Engine Object

This method opens image file and creates the **FRDocument** object.

Visual Basic Syntax

```
Method CreateFRDocumentFromImage(
  imageFileName As String,
  prepareMode   As PrepareImageMode,
) As FRDocument
```

C++ Syntax

```
HRESULT CreateFRDocumentFromImage(
  BSTR               imageFileName,
  IPrepareImageMode* prepareMode,
  IFRDocument**      createdDocument
);
```

Parameters

imageFileName

[in] This variable contains a full path to the image file to open. For example "C:\MyPictures\MyPic.bmp".

prepareMode

[in] This parameter refers to the **PrepareImageMode** object which specifies how an image will be preprocessed during opening.

createdDocument

[out, retval] A pointer to **IFRDocument*** pointer variable that receives the interface pointer of the resulting **FRDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

Engine
FRDocument
IEngine::CreateFRDocument

See samples: Hello, RecognizedTextProcessing, CustomLanguage

CreateLayoutBlocks Method of the Engine Object

This method creates a **LayoutBlocks** object of the type specified.

Visual Basic Syntax

```
Method CreateLayoutBlocks(
  parentLayout As Layout
) As LayoutBlocks
```

C++ Syntax

```
HRESULT CreateLayoutBlocks(
  ILayout*        parentLayout,
  ILayoutBlocks** result
);
```

Parameters

parentLayout

[in] This parameter refers to the parent **Layout** object. Must not be NULL.

result

[out, retval] A pointer to **ILayoutBlocks*** pointer variable that receives the interface pointer of the created collection of blocks. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
LayoutBlocks

CreateMultipageImageWriter Method of the Engine Object

This method creates a **MultipageImageWriter** object that may be used for saving multiple images into a single multipage image file.

Visual Basic Syntax

```
Method CreateMultipageImageWriter(
  fileName   As String,
  fileFormat As ImageFileFormatEnum
) As MultipageImageWriter
```

C++ Syntax

```
HRESULT CreateMultipageImageWriter(
  BSTR                   fileName,
  ImageFileFormatEnum    fileFormat,
  IMultipageImageWriter** result
);
```

Parameters

fileName

[in] This parameter contains the full path to the multipage image file where the images will be saved. For example, "C:\MyPic.tif". If a file with this name already exists, it will be overwritten without prompt.

fileFormat

[in] A variable of the **ImageFileFormatEnum** type that specifies the format of the output file. Not all formats defined by this enumeration are supported for writing.

result

[out, retval] A pointer to **IMultipageImageWriter*** pointer variable that receives the interface pointer of the **MultipageImageWriter** output object. This object allows one to append images to the end of the multipage image file.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

Note that not all formats available for writing are suitable for multipage images. Therefore, you can create the **MultipageImageWriter** object for one-page formats, but can add no more than one page to the resulting file.

See also

Engine
MultipageImageWriter

CreateNewDictionary Method of the Engine Object

This method creates a new empty user dictionary at the specified location and returns interface pointer of the **Dictionary** object associated with it.

Visual Basic Syntax

```
Method CreateNewDictionary(
  fileName   As String,
  languageId As LanguageIdEnum,
) As Dictionary
```

C++ Syntax

```
HRESULT CreateNewDictionary(
  BSTR           fileName,
  LanguageIdEnum languageId,
  IDictionary**  result
);
```

Parameters

fileName

[in] This variable contains the full path to the dictionary file to be created.

languageId

[in] A variable of **LanguageIdEnum** type that defines the language for the dictionary. Do not pass the LI_Japanese, LI_Korean, LI_KoreanJohab, LI_ChinesePRC, LI_ChineseTaiwan, LI_ChineseHongKong, LI_ChineseSingapore, or LI_ChineseMacau constant to this parameter, as user dictionaries cannot be created for corresponding languages.

result

[out, retval] A pointer to **IDictionary*** pointer variable that receives the interface pointer of the **Dictionary** object associated with the newly created dictionary. You may then edit this dictionary via this object methods.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Working with Dictionaries
Engine
Dictionary

Creation DictionaryDescription Methods of the Engine Object

The **Engine** object provides four methods that create types of dictionaries: standard, user, "regular expression-based", or external. They all have similar semantics.

Visual Basic Syntax

```
Method Create<DictDesc>(
) As <DictDesc>
```

C++ Syntax

```
HRESULT Create<DictDesc>(
  I<DictDesc>** result
);
```

Parameters

result

[out] A pointer to ***I<DictDesc>**** pointer variable that receives the interface pointer of the created object, it may be a **StandardDictionaryDescription**, **UserDictionaryDescription, RegExpDictionaryDescription**, or **ExternalDictionaryDescription** object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
```

```
FREngine::IEnginePtr Engine;
...
// Create the StandardDictionaryDescription object
FREngine::IStandardDictionaryDescriptionPtr pStdDict = Engine-
>CreateStandardDictionaryDesc();
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Create the StandardDictionaryDescription object
Dim StdDict As FREngine.StandardDictionaryDescription
Set StdDict = Engine.CreateStandardDictionaryDesc
...
```

See also

Working with Dictionaries
Engine

Supplementary Methods of the Engine Object

The **Engine** object exposes the methods which provide additional services.

Name	Description
ConvertLanguageIdToLCID	Converts ABBYY FineReader Engine internal representation of language ID by the **LanguageIdEnum** into Win32 standard LCID.
ConvertLCIDToLanguageId	Converts Win32 standard LCID into ABBYY FineReader Engine internal representation of language ID by the **LanguageIdEnum**.
EditUserPattern	Produces the dialog box for editing user pattern in the specified path.
LoadModule	Allows you to load only the modules which will be necessary for application operation.
LoadPredefinedProfile	Loads one of the predefined profiles.
LoadProfile	Loads a user profile file.
MergePatterns	Merges several user pattern files into one file.
OpenExistingDictionary	Opens an existing user-defined dictionary for editing.
SetCurrentLicense	Sets current license.
StartLogging	Enables logging of errors, warnings and method calls.
StopLogging	Disables logging of errors, warnings and method calls.
TrainUserPattern	Allows you to perform pattern training.

See also

Engine

ConvertLanguageIdToLCID Method of the Engine Object

This method converts the Win32 standard LANGID, represented by **LanguageIdEnum**, into the Win32 standard LCID. The return value of this function may be directly cast to the LCID type.

Visual Basic Syntax

```
Method ConvertLanguageIdToLCID(
  languageId As LanguageIdEnum
) As Long
```

C++ Syntax

```
HRESULT ConvertLanguageIdToLCID(
  LanguageIdEnum languageId,
  long*          result
);
```

Parameters

languageId

[in] This variable of the **LanguageIdEnum** type specifies Win32 standard LANGID.

result

[out, retval] A pointer to a **long** variable that receives the return value of this function. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method may return 0 in case the passed value of **LanguageIdEnum** does not denote any of the Win32 LCIDs.

See also

Engine
IEngine::ConvertLCIDToLanguageId

ConvertLCIDToLanguageId Method of the Engine Object

This method converts the Win32 standard LCID into the Win32 standard LANGID, represented by **LanguageIdEnum**.

Visual Basic Syntax

```
Method ConvertLCIDToLanguageId(
  win32Locale As Long
) As LanguageIdEnum
```

C++ Syntax

```
HRESULT ConvertLCIDToLanguageId(
  long            win32Locale,
  LanguageIdEnum* result
);
```

Parameters

win32Locale

[in] This variable contains a Win32 standard LCID casted to the **long** type.

result

[out] A pointer to **LanguageIdEnum** variable that receives the return value of this method. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method may return LI_Null in case the passed Win32 LCID is not supported by ABBYY FineReader Engine.

See also

Engine
IEngine::ConvertLanguageIdToLCID

EditUserPattern Method of the Engine Object

This method displays the **User Pattern** dialog box that allows a user to edit user pattern file.

Visual Basic Syntax

```
Method EditUserPattern(
  fileName As String
)
```

C++ Syntax

```
HRESULT EditUserPattern(
  BSTR fileName
);
```

Parameters

fileName

[in] This variable contains the full path to the user pattern file that is to be edited. This file should be created by the **IEngine::CreateEmptyUserPattern** method or by ABBYY FineReader.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- User patterns are files that specify a number of pairs "image of a character — the character itself". User patterns may be used during recognition to identify characters of non-standard fonts. To use a user pattern during recognition, specify a path to it in the **IRecognizerParams::UserPatternsFile** property. See Recognizing with Training.

- The method is enabled only if the license supports the User Patterns Training module.

- Pattern training is not supported for hieroglyphic languages.

See also

Recognizing with Training
Engine
IEngine::CreateEmptyUserPattern

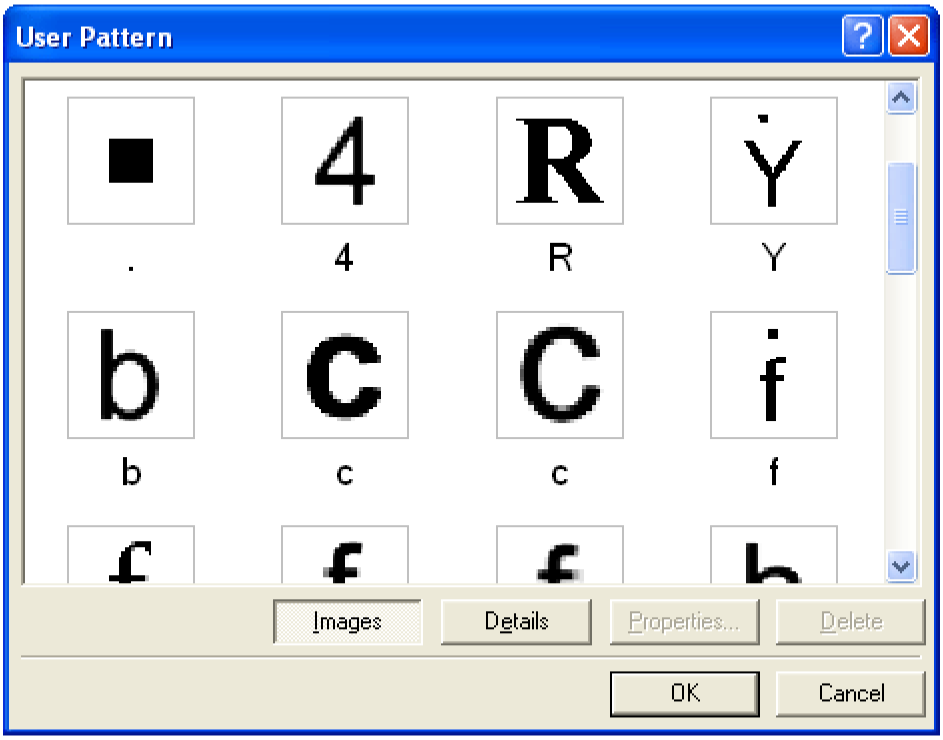
The User Pattern Dialog Box

All the trained characters together with their captions images are displayed in this dialog box. Images with the same captions are put one under another.

The trained characters are displayed in two modes:

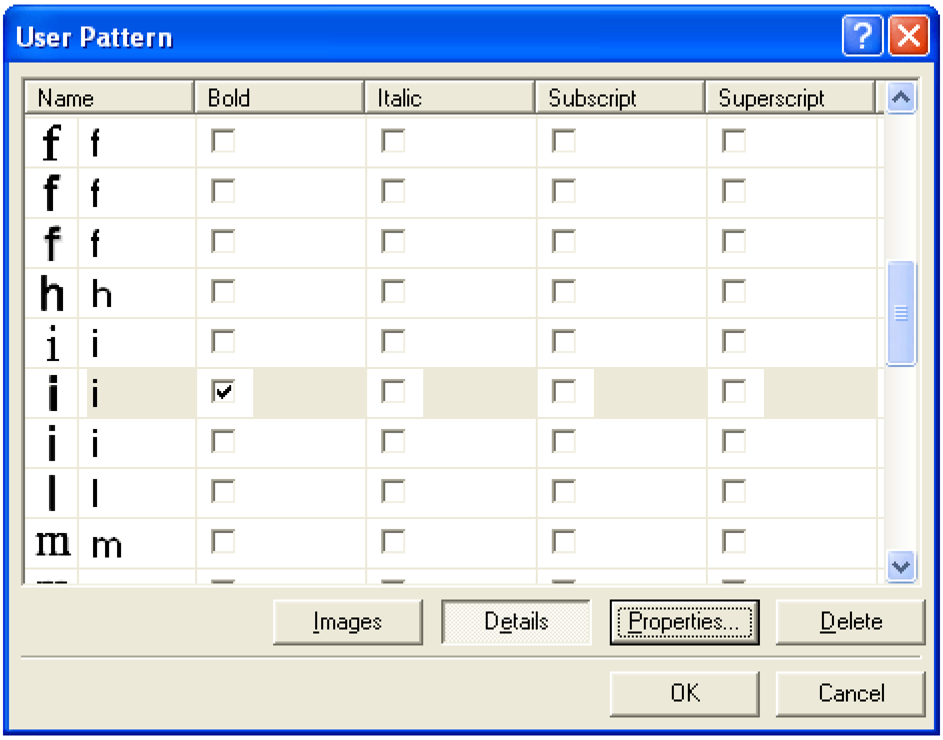
Images mode

Displays pairs "image – caption".



Details mode

Displays a table with the character image, caption and properties columns. You may edit the character caption and properties right in this table.



Name	Bold	Italic	Subscript	Superscript
f	□	□	□	□
f	□	□	□	□
f	□	□	□	□
h	□	□	□	□
i	□	□	□	□
i	☑	□	□	□
i	□	□	□	□
l	□	□	□	□
m	□	□	□	□

Click the **Images** or **Details** buttons to switch between modes.

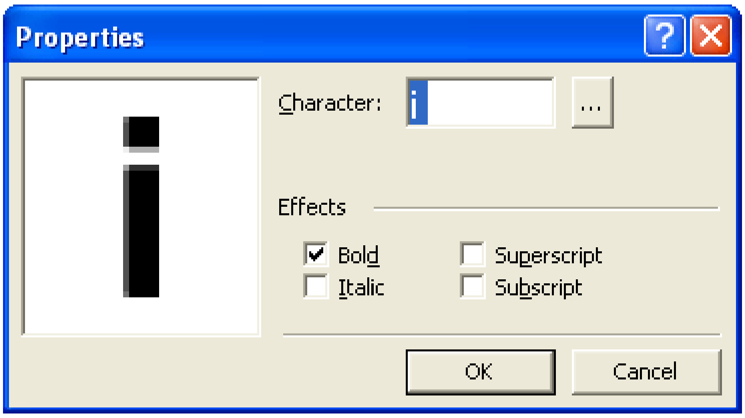
The **Properties** button opens the **Character Properties** dialog box. You may change the character caption and properties in this dialog box.

The **Delete** button deletes the selected characters from the pattern.

See also

IEngine::EditUserPattern
Character Properties Dialog Box

The Character Properties Dialog Box



Character: i

☑ Bold
□ Italic
□ Superscript
□ Subscript

The left dialog window displays an "average" character image (all similar images are put one under another). The **Character** field displays the character caption.

If the caption is incorrect, type the correct caption. If you train ABBYY FineReader to recognize characters you cannot type, you may use two–character combinations as captions, or you may copy the necessary character from the **character table**. Click the … button to open the table.

Effects group

- If the characters you train are set in italics or bold and you want to keep these effects in the recognized text, do not forget to set the **Italic** and/or **Bold** items in **Pattern Training** dialog box.
- If the character you train is superscript (subscript), set the **Superscript** (**Subscript**) item in the **Effects** group.

See also

IEngine::EditUserPattern
User Pattern Dialog Box

LoadModule Method of the Engine Object

ABBYY FineReader Engine has several functional modules:

- **Export** for export of recognition results
- **Document Analyzer** for document analysis
- **Recognizer** for printed text recognition
- **RecognizerHP** for handprinted text recognition and checkmark recognition
- **FineReader Engine Processor** for parallel recognition
- **Chinese Traditional Patterns** for recognition of texts in Chinese Traditional language
- **Chinese Simplified Patterns** for recognition of texts in Chinese Simplified language
- **Japanese Patterns** for recognition of texts in Japanese language
- **Korean Patterns** for recognition of texts in Korean and Korean (Hangul) languages
- **European Patterns** for recognition of texts in European languages

By default, these modules are loaded when the need arises. When a method dealing with one of the functionalities is first called, the corresponding module is loaded. For example, the **Export** module will be loaded after any export method is called.

The **LoadModule** method allows you to load the modules which will be necessary for application operation.

Visual Basic Syntax

```
Method LoadModule(
  value As FREngineModuleEnum
)
```

C++ Syntax

```
HRESULT LoadModule(
  FREngineModuleEnum value
);
```

Parameters

value

[in] This variable of the **FREngineModuleEnum** type specifies the module which is to be loaded.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine

LoadPredefinedProfile Method of the Engine Object

This method loads one of the predefined profiles. For more information about working with profiles, see the Working with Profiles section of this Help file.

⚠**Important!** The profiles may require additional modules available in the license. See details in the descriptions of corresponding scenarios.

Visual Basic Syntax

```
Method LoadPredefinedProfile(
  ProfileName As String
)
```

C++ Syntax

```
HRESULT LoadPredefinedProfile(
  BSTR ProfileName
);
```

Parameters

ProfileName

[in] This variable contains a profile name. It may be one of the following:

- *DocumentConversion_Accuracy*
  Suitable for converting documents into an editable format. The settings have been optimized for accuracy.
- *DocumentConversion_Speed*
  Suitable for converting documents into an editable format. The settings have been optimized for processing speed.
- *DocumentArchiving_Accuracy*
  Suitable for creating an electronic archive. The settings have been optimized for accuracy.
- *DocumentArchiving_Speed*
  Suitable for creating an electronic archive. The settings have been optimized for processing speed.
- *BookArchiving_Accuracy*
  Suitable for creating an electronic library. The settings have been optimized for accuracy.
- *BookArchiving_Speed*
  Suitable for creating an electronic library. The settings have been optimized for processing speed.
- *TextExtraction_Accuracy*
  Suitable for extracting text from a document. The settings have been optimized for accuracy.

- *TextExtraction_Speed*
  Suitable for extracting text from a document. The settings have been optimized for processing speed.
- *FieldLevelRecognition*
  Suitable for recognizing short text fragments.
- *BarcodeRecognition*
  Suitable for barcode extraction.
- *Version9Compatibility*
  Provided for compatibility, sets the processing parameters to the default values of ABBYY FineReader Engine 9.0.
- *Default*
  Sets all the processing parameters to the default values.

Note: You can view the list of settings provided by these profiles in the descriptions of corresponding main usage scenarios.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

We recommend calling this method before the **FRDocument** object is created. If you have already used the **FRDocument** object (have already performed analysis and recognition) and now want to load a profile and recognize pages of the document again, call the **IFRPage::CleanRecognizerSession** method for each page before loading a profile.

See also

Engine
Working with Profiles

LoadProfile Method of the Engine Object

This method loads a user profile file. For more information about working with profiles, see the Working with Profiles section of this Help file.

Visual Basic Syntax

```
Method LoadProfile(
  filePath As String
)
```

C++ Syntax

```
HRESULT LoadProfile(
  BSTR filePath
);
```

Parameters

filePath

[in] This variable contains a path to the profile file. You can specify either a full path to the file, or a path relative to the current directory. If this variable contains an empty string, standard default values for all newly created objects will be used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

We recommend calling this method before the **FRDocument** object is created. If you have already used the **FRDocument** object (have already performed analysis and recognition) and now want to load a profile and recognize pages of the document again, call the **IFRPage::CleanRecognizerSession** method for each page before loading a profile.

See also

Engine
Working with Profiles

MergePatterns Method of the Engine Object

This method merges several user pattern files into one file.

Visual Basic Syntax

```
Method MergePatterns(
  SourceFilesNames    As StringsCollection,
  DestinationFileName As String
)
```

C++ Syntax

```
HRESULT MergePatterns(
  IStringsCollection* SourceFilesNames,
  BSTR                DestinationFileName
);
```

Parameters

SourceFilesNames

[in] This variable of the **StringsCollection** type contains a collection of the full paths to the user pattern files that are to be merged. These files should be created by the **IEngine::CreateEmptyUserPattern** method or by ABBYY FineReader.

DestinationFileName

[in] This variable contains the full path to the resulting user pattern file.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

User patterns are files that define a number of pairs "image of a character — the character itself". User patterns may be used during recognition to identify characters of non-standard fonts. To use a user pattern during recognition, set the path to it in the **IRecognizerParams::UserPatternsFile** property.

See also

Engine
IEngine::CreateEmptyUserPattern
Recognizing with Training

OpenExistingDictionary Method of the Engine Object

This method opens an existing user dictionary for editing. It returns the interface pointer to the **Dictionary** object associated with the dictionary. The **OpenExistingDictionary** method can open dictionaries created with the help of the **IEngine::CreateNewDictionary** method, as well as user dictionaries (*.amd) created in ABBYY FineReader. The user dictionaries in ABBYY FineReader are created together with user languages and are saved in the folder of the current document. For more details on the creation of user languages and dictionaries, see the ABBYY FineReader help file.

Visual Basic Syntax

```
Method OpenExistingDictionary(
  fileName As String
) As Dictionary
```

C++ Syntax

```
HRESULT CreateNewDictionary(
  BSTR          fileName,
  IDictionary** result
);
```

Parameters

fileName

[in] This variable contains the full path to the dictionary file.

result

[out] A pointer to **IDictionary*** pointer variable that receives the interface pointer to the **Dictionary** object associated with the dictionary. You may then edit this dictionary via this object's methods.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Working with Dictionaries
Engine
IEngine::CreateNewDictionary
Dictionary

SetCurrentLicense Method of the Engine Object

This method sets the current license.

⚠**Important!** All the ABBYY FineReader Engine objects, which were in use before this method call, become invalid (except the current **Engine** object). The only thing that you can do with these objects is to call **Release** method for them.

Visual Basic Syntax

```
Method SetCurrentLicense(
  license      As License
  serialNumber As String
)
```

C++ Syntax

```
HRESULT SetCurrentLicense(
  ILicense* license
  BSTR      serialNumber
);
```

Parameters

license

[in] This parameter refers to the **License** object representing the license.

serialNumber

[in] This variable contains the serial number of the Developer License. This serial number must correspond to the serial number from the **SerialNumber** property of the **License** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine

StartLogging Method of the Engine Object

This method enables logging of errors, warnings and method calls.

Visual Basic Syntax

```
Method StartLogging(
  logFileName      As String,
  writeMethodCalls As Boolean
)
```

C++ Syntax

```
HRESULT StartLogging(
  BSTR         logFileName,
  VARIANT_BOOL writeMethodCalls
);
```

Parameters

logFileName

[in] This parameter contains the full path to the log file. If the file does not exist, it will be created.

writeMethodCalls

[in] This parameter enables logging calls of ABBYY FineReader Engine methods to the log file. The format is as follows:
<time of call>, <duration of execution (in ms)>, <Interface::Method (parameter1, ...)>
For example, **12:40:31.254, 17 ms, IRecognizerParams::put_OneWordPerLine(VARIANT_TRUE)**

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The log file can contain a list of objects that have not been deleted before the deinitialization of the **Engine** object. To do this, do not call the **StopLogging** method which disables logging, before the deinitialization.

See also

Engine
IEngine::StopLogging

StopLogging Method of the Engine Object

This method disables logging of errors, warnings and method calls.

Visual Basic Syntax

```
Method StopLogging()
```

C++ Syntax

```
HRESULT StopLogging();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IEngine::StartLogging

TrainUserPattern Method of the Engine Object

This method allows you to perform pattern training. The method adds new pairs "a character image — the character itself" to the specified pattern.

Visual Basic Syntax

```
Method TrainUserPattern(
  fileName            As String,
  trainingImages      As TrainingImagesCollection,
  characterOrLigature As String,
  flags               As Long,
  textType            As TextTypeEnum
)
```

C++ Syntax

```
HRESULT TrainUserPattern(
  BSTR                       fileName,
  ITrainingImagesCollection* trainingImages,
  BSTR                       characterOrLigature,
  long                       flags,
  TextTypeEnum               textType
);
```

Parameters

fileName

[in] This variable specifies the path to the user pattern file.

trainingImages

[in] This variable refers to the **TrainingImagesCollection** object that stores a collection of character images.

characterOrLigature

[in] This variable specifies the character which is represented by the collection of images.

flags

[in] This parameter contains a bitwise OR combination of the **UPTF_** prefixed flags which specifies the character attributes.

textType

[in] This parameter specifies the text type of the character using the **TextTypeEnum** enumeration constant.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
UPTF_ prefixed flags
Recognizing with Training

Processing Methods of the Engine Object

The **Engine** object exposes a set of processing methods. These methods are suitable only for working with one-page documents. For multi-page documents ABBYY FineReader Engine provides more convenient way of processing. We recommend to create **FRDocument** object and use its methods and properties for processing.

Methods for working with images

These methods allow you to open images, bitmaps, and convert them into ABBYY FineReader Engine internal format.

Name	Description
GetNumberOfPagesInImageFile	Returns the number of pages in an image file.
LoadImageDocFromFile	Restores the contents of the **ImageDocument** object from the file previously saved with the help of the **IImageDocument::SaveToFile** method.
LoadImageDocFromMemory	Restores the contents of the **ImageDocument** object from the global memory.
OpenBitmapImage	Opens bitmap image given its HBITMAP handle.
OpenDib	Opens a device-independent bitmap image file.
OpenImage	Opens an image in ABBYY FineReader Engine internal format.
OpenMemoryImage	Opens an image from the buffer in memory.
PrepareAndOpenBitmap	Converts bitmap image into ABBYY FineReader Engine internal format and opens it.
PrepareAndOpenDib	Converts device-independent bitmap into ABBYY FineReader Engine internal format and opens it.
PrepareAndOpenImage	Converts an image file into ABBYY FineReader Engine internal format and opens it.
PrepareAndOpenMemoryImage	Creates a copy of a memory image in ABBYY FineReader Engine internal format and opens it.
PrepareBitmap	Creates a copy of the specified bitmap image in ABBYY FineReader Engine internal format
PrepareDib	Creates a copy of the specified device-independent bitmap in ABBYY FineReader Engine internal format.
PrepareImage	Creates a copy of an image file in ABBYY FineReader Engine internal format.
PrepareMemoryImage	Creates a copy of a memory image in the file in ABBYY FineReader Engine internal format.

Analysis, recognition and synthesis methods

These methods are analogous to those of the **DocumentAnalyzer** object, though have simpler semantics. They create the **DocumentAnalyzer** object internally and use its methods.

Name	Description
AnalyzeAndRecognizePage	Performs layout analysis, recognition, and page synthesis of the specified image.
AnalyzeAndRecognizePages	Performs layout analysis, recognition, and page synthesis of a collection of images.
AnalyzePage	Performs layout analysis of an image.
AnalyzePages	Performs the layout analysis of a collection of images.
RecognizeImageAsPlainText	Opens an image file, recognizes it and returns recognized text in a special "plain text" format.
RecognizeImageDocumentAsPlainText	Recognizes an image and returns recognized text in a special "plain text" format.
RecognizeImageFile	Performs analysis, recognition, and synthesis of the specified image file and export of the recognized text into a file in external format.
RecognizePage	Recognizes parts of the specified image that lay inside the blocks in the specified **Layout** object.
RecognizePages	Recognizes those parts of the images from the collection that lay inside the blocks of the specified layout collection.
SynthesizePages	Performs the document synthesis of a collection of recognized images.
SynthesizePagesEx	Performs the document synthesis of a collection of recognized images. This method is optimized from the point of view of memory consumption.

Export methods

These methods are analogous to those of the **Exporter** object and have simple semantics. They create the **Exporter** object internally and use its methods.

Name	Description
ExportPage	Saves recognized text from one page into a file of one of the available formats.
ExportPages	Saves recognized text from several pages into a file of one of the available formats.

See also

Engine

GetNumberOfPagesInImageFile Method of the Engine Object

This method returns the number of pages in an image file.

Visual Basic Syntax

```
Method GetNumberOfPagesInImageFile(
  fileName         As String,
  passwordCallback As ImagePasswordCallback
) As Long
```

C++ Syntax

```
HRESULT GetNumberOfPagesInImageFile(
  BSTR                    fileName,
  IImagePasswordCallback* passwordCallback,
  long*                   result
);
```

Parameters

fileName

[in] This variable contains the full path to the image file. For example "C:\MyPictures\MyPic.tif".

passwordCallback

[in] This variable refers to the interface of the user-implemented object of the type **ImagePasswordCallback** which is used to handle possible password requests for accessing images in PDF format. This parameter is optional and may be 0 in which case password-protected files cannot be processed.

result

[out, retval] A pointer to **long** variable that receives the return value of this method.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IImagePasswordCallback

LoadImageDocFromFile Method of the Engine Object

This method restores the contents of the **ImageDocument** object from the file previously saved with the help of the **IImageDocument::SaveToFile** method.

Visual Basic Syntax

```
Method LoadImageDocFromFile(
   fileName As String
) As ImageDocument
```

C++ Syntax

```
HRESULT LoadImageDocFromFile(
  BSTR             fileName
  IImageDocument** result
);
```

Parameters

fileName

[in] This variable contains a full path to the file with image document.

result

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer of the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IImageDocument::SaveToFile

LoadImageDocFromMemory Method of the Engine Object

This method restores the contents of the **ImageDocument** object from the global memory. The contents must be loaded to the memory with the help of the **IImageDocument::SaveToMemory** method. This method returns the HGLOBAL handle of the memory from where the object's contents are loaded.

Visual Basic Syntax

```
Method LoadImageDocFromMemory(
   hGlobal As Long
) As ImageDocument
```

C++ Syntax

```
HRESULT LoadImageDocFromMemory(
  long             hGlobal
  IImageDocument** result
);
```

Parameters

hGlobal

[in] This parameter specifies the HGLOBAL handle of the memory from where the object's contents should be loaded. The parameter is statically casted to the **Long** type. This handle should be the one obtained from the **IImageDocument::SaveToMemory** method, and should be valid (not freed by the **GlobalFree** function).

result

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer of the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The size of the memory area that the object allocates, can be obtained by calling the **GlobalSize** function.

See also

Engine
IImageDocument::SaveToMemory

OpenBitmapImage Method of the Engine Object

This method opens a bitmap image given its HBITMAP handle.

Visual Basic Syntax

```
Method OpenBitmapImage(
  bitmapHandle As Long
  resolution   As Long
) As ImageDocument
```

C++ Syntax

```
HRESULT OpenBitmapImage(
  long             bitmapHandle,
  long             resolution,
  IImageDocument** result
);
```

Parameters

bitmapHandle

[in] This variable contains the handle of the GDI object (HBITMAP), statically casted to the **Long** type. This handle should be available to the process that operates ABBYY FineReader Engine.

resolution

[in] Resolution of the bitmap in dpi (dots per inch).

result

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer of the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method may open only black-and-white bitmaps.

See also

Engine
IEngine::PrepareBitmap

IEngine::PrepareAndOpenBitmap

OpenDib Method of the Engine Object

This method opens a bitmap image specified by its bitmap handle (DIB format) to a global memory block (HGLOBAL).

Note: DIB must be created using Windows API.

Visual Basic Syntax

```
Method OpenDib(
  dibHglobal As Long,
  resolution As Long
) As ImageDocument
```

C++ Syntax

```
HRESULT OpenDib(
  long              dibHglobal,
  long              resolution,
  IImageDocument** result
);
```

Parameters

dibHglobal

[in] This variable contains the handle of the memory block (HGLOBAL) where the bitmap is saved, statically casted to the **Long** type. This handle should be available to the process that uses ABBYY FineReader Engine.

resolution

[in] Resolution of the bitmap in dpi (dots per inch).

result

[out] A pointer to **IImageDocument*** pointer variable that receives the interface pointer to the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method allows you to open only black-and-white images.

See also

Engine
IEngine::PrepareAndOpenDib
IEngine::PrepareDib
IEngine::OpenBitmapImage

OpenImage Method of the Engine Object

This method allows you to open images in ABBYY FineReader Engine internal format. Images in other formats cannot be opened using this method.

Visual Basic Syntax

```
Method OpenImage(
  fileName     As String
) As ImageDocument
```

C++ Syntax

```
HRESULT OpenImage(
  BSTR              fileName,
  IImageDocument** result
);
```

Parameters

fileName

[in] This variable contains a full path to the folder with image in internal format.

result

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer to the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method is primarily designed for multipage scenarios when it is not possible to keep all the images in memory. In this case, **ImageDocument** objects are saved on disk, and then **OpenImage** method is called as needed.

To open images in other formats, use the **IEngine::PrepareAndOpenImage** method.

See also

Working with Images
Engine
IEngine::PrepareAndOpenImage
IEngine::PrepareImage

OpenMemoryImage Method of the Engine Object

This method opens an image from the buffer in memory. The image should be isotropic, that is its horizontal resolution should equal the vertical one.

Visual Basic Syntax

```
Method OpenMemoryImage(
  format     As MemoryImageFormatEnum,
  width      As Long,
  height     As Long,
  byteWidth  As Long,
  resolution As Long,
  rawDataPtr As Long
) As ImageDocument
```

C++ Syntax

```
HRESULT OpenMemoryImage(
  MemoryImageFormatEnum format,
  long                  width,
  long                  height,
  long                  byteWidth,
  long                  resolution,
  long                  rawDataPtr,
  IImageDocument**      result
);
```

Parameters

format

[in] This parameter of **MemoryImageFormatEnum** type describes the format of the memory image file that is to be opened.

width

[in] This parameter specifies the width of the image in pixels.

height

[in] This parameter specifies the height of the image in pixels.

byteWidth

[in] This parameter specifies the width of the line of image in bytes (padding included).

resolution

[in] This parameter specifies resolution of the image in dots per inch.

rawDataPtr

[in] This parameter is treated as a pointer to memory buffer containing image data. See Memory image format description section for details.

result

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer to the resulting **ImageDocument** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The image data should exist while the **ImageDocument** object received from this method exists.

See also

Engine
IEngine::PrepareMemoryImage
IEngine::PrepareAndOpenMemoryImage
MemoryImageFormatEnum
Memory image format description

PrepareAndOpenBitmap Method of the Engine Object

This method converts a bitmap image into ABBYY FineReader Engine internal format and opens it.

Visual Basic Syntax

```
Method PrepareAndOpenBitmap(
  bitmapHandle As Long,
  xResolution  As Long,
  yResolution  As Long,
  prepareMode  As PrepareImageMode
) As ImageDocument
```

C++ Syntax

```
HRESULT PrepareAndOpenBitmap(
  long               bitmapHandle,
  long               xResolution,
  long               yResolution,
  IPrepareImageMode* prepareMode
  IImageDocument**   imageDocument
);
```

Parameters

bitmapHandle

[in] This variable contains the handle of the GDI object (HBITMAP), statically casted to the **Long** type. This handle should be available to the process that operates ABBYY FineReader Engine.

xResolution

[in] Horizontal resolution of the bitmap.

yResolution

[in] Vertical resolution of the bitmap.

prepareMode

[in] This variable refers to the **PrepareImageMode** object that stores parameters for bitmap conversion in internal format. This parameter may be 0 in which case default parameters of the image preparation mode are used.

imageDocument

[out] A pointer to **IImageDocument*** pointer variable that receives the interface pointer to the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IEngine::PrepareAndOpenImage
IEngine::OpenImage
IEngine::PrepareBitmap
IEngine::OpenBitmapImage

PrepareAndOpenDib Method of the Engine Object

This method converts a bitmap image specified by its bitmap handle from DIB (Device-Independent Bitmap) format into ABBYY FineReader Engine internal format and opens it.

Note: DIB must be created using Windows API.

Visual Basic Syntax

```
Method PrepareAndOpenDib(
  dibHglobal   As Long,
  xResolution  As Long,
  yResolution  As Long,
  prepareMode  As PrepareImageMode
) As ImageDocument
```

C++ Syntax

```
HRESULT PrepareAndOpenDib(
  long               dibHglobal,
  long               xResolution,
  long               yResolution,
  IPrepareImageMode* prepareMode
  IImageDocument**   imageDocument
);
```

Parameters

dibHglobal

[in] This variable contains the handle of the memory block (HGLOBAL) where the bitmap is saved, statically casted to the **Long** type. This handle should be available to the process that operates ABBYY FineReader Engine.

xResolution

[in] Horizontal resolution of the bitmap.

yResolution

[in] Vertical resolution of the bitmap.

prepareMode

[in] This variable refers to the **PrepareImageMode** object that stores parameters for bitmap conversion in internal format. This parameter may be 0 in which case default parameters of the image preparation mode are used, or, if a profile has been loaded, the parameters set by this profile are used.

imageDocument

[out] A pointer to **IImageDocument*** pointer variable that receives the interface pointer to the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IEngine::OpenDib
IEngine::PrepareDib
Working with Profiles

PrepareAndOpenImage Method of Engine Object

This method converts an image file into ABBYY FineReader Engine internal format and opens it.

Visual Basic Syntax

```
Method PrepareAndOpenImage(
  fileName         As String,
  prepareMode      As PrepareImageMode,
  passwordCallback As ImagePasswordCallback,
  documentInfo     As DocumentInfo
) As ImageDocument
```

C++ Syntax

```
HRESULT PrepareAndOpenImage(
  BSTR                    fileName,
  IPrepareImageMode*      prepareMode,
  IImagePasswordCallback* passwordCallback,
  IDocumentInfo*          documentInfo,
  IImageDocument**        result
);
```

Parameters

fileName

[in] This variable contains the full path to the image file to open. For example "C:\MyPictures\MyPic.bmp".

prepareMode

[in] This variable refers to the **PrepareImageMode** object that defines the mode of converting the image file into ABBYY FineReader Engine internal format. This parameter may be 0 in which case the default parameters of the image preparation mode are used, or, if a profile has been loaded, the parameters set by this profile are used.

passwordCallback

[in] This variable refers to the interface of the user-implemented object of the type **ImagePasswordCallback** which is used to handle possible password requests for accessing images in PDF format. This parameter is optional and may be 0 in which case password-protected files cannot be processed.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being opened.

result

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer of the resulting **ImageDocument** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

If an image file specified is not in ABBYY FineReader Engine internal format, it is automatically converted into this format. This may take some time, the parameters specified by the *prepareMode* input parameter are used. If the image file is already in ABBYY FineReader Engine internal format, it is not prepared, but opened, thus the *prepareMode* parameter is ignored.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
```

```
FREngine::IEnginePtr Engine;
...
// Create and customize image loading parameters
FREngine::IPrepareImageModePtr prepareImageMode = Engine->CreatePrepareImageMode();
// Turn off skew correction
prepareImageMode->CorrectSkewMode = 0;
// Turn on mirroring
prepareImageMode->MirrorImage = VARIANT_TRUE;
// Open the image file
FREngine::IImageDocumentPtr pImageDoc =
      Engine->PrepareAndOpenImage( "D:\\Demo.tif", prepareImageMode, 0, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Create and customize image loading parameters
Dim PrepareImageMode As FREngine.PrepareImageMode
Set PrepareImageMode = Engine.CreatePrepareImageMode
' Turn off skew correction
PrepareImageMode.CorrectSkewMode = 0
' Turn on mirroring
PrepareImageMode.MirrorImage = True
' Open the image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( "D:\Demo.tif", PrepareImageMode)
```

See also

Engine
IEngine::OpenImage
IEngine::OpenBitmapImage
IEngine::PrepareImage
IImagePasswordCallback
Working with Images
Working with Profiles

PrepareAndOpenMemoryImage Method of the Engine Object

This method creates a copy of a memory image in a temporary folder on disk in ABBYY FineReader Engine internal format and opens it. This method is equivalent to successive calls to the **IEngine::PrepareMemoryImage** and **IEngine::OpenImage**.

Visual Basic Syntax

```
Method PrepareAndOpenMemoryImage(
  format       As MemoryImageFormatEnum,
  width        As Long,
  height       As Long,
  byteWidth    As Long,
  xResolution  As Long,
  yResolution  As Long,
  rawDataPtr   As Long,
  prepareMode  As PrepareImageMode
) As ImageDocument
```

C++ Syntax

```
HRESULT PrepareAndOpenMemoryImage(
  MemoryImageFormatEnum format,
  long                  width,
  long                  height,
  long                  byteWidth,
  long                  xResolution,
  long                  yResolution,
  long                  rawDataPtr,
  IPrepareImageMode*    prepareMode,
  IImageDocument**      imageDoc
);
```

Parameters

format

[in] This parameter of the **MemoryImageFormatEnum** type describes the format of the memory image file to be prepared.

width

[in] This parameter specifies the width of the image in pixels.

height

[in] This parameter specifies the height of the image in pixels.

byteWidth

[in] This parameter specifies the width of the line of image in bytes (padding included).

xResolution

[in] This parameter specifies horizontal resolution of the image in dots per inch.

yResolution

[in] This parameter specifies vertical resolution of the image in dots per inch.

rawDataPtr

[in] This parameter is treated as a pointer to memory buffer containing image data. See Memory image format description section for details.

prepareMode

[in] This parameter refers to the **PrepareImageMode** object that defines the mode of image preparation. It may be 0, in which case default values for the **PrepareImageMode** properties will be used, or, if a profile has been loaded, the parameters set by this profile are used.

imageDoc

[out, retval] A pointer to **IImageDocument*** pointer variable that receives the interface pointer to the resulting **ImageDocument** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IEngine::OpenMemoryImage
IEngine::PrepareAndOpenMemoryImage
MemoryImageFormatEnum
Memory image format description
Working with Profiles

PrepareBitmap Method of the Engine Object

This method creates a copy of the specified bitmap image in ABBYY FineReader Engine internal format in the specified folder on the disk.

Visual Basic Syntax

```
Method PrepareBitmap(
  bitmapHandle As Long,
  destFileName As String,
  xResolution  As Long,
  yResolution  As Long,
  prepareMode  As PrepareImageMode
)
```

C++ Syntax

```
HRESULT PrepareBitmap(
  long               bitmapHandle,
  BSTR               destFileName,
  long               xResolution,
  long               yResolution,
  IPrepareImageMode* prepareMode
);
```

Parameters

bitmapHandle

[in] This variable contains the handle of the GDI object (HBITMAP), statically casted to the **Long** type. This handle should be available to the process that operates ABBYY FineReader Engine.

destFileName

[in] The full path to the destination folder.

xResolution

[in] Horizontal resolution of the bitmap.

yResolution

[in] Vertical resolution of the bitmap.

prepareMode

[in] This variable refers to the **PrepareImageMode** object that stores parameters for bitmap conversion in internal format. This parameter may be 0 in which case default parameters of the image preparation mode are used, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- This method allows you to convert a bitmap image into ABBYY FineReader Engine internal format. It provides a way to get read-write **ImageDocument** object corresponding to the bitmap by applying this method first, and then opening the resulting file via the **IEngine::OpenImage** method.

- This is a developer concern to delete the files with prepared image when they are no longer necessary. These files are not automatically removed from the disk by ABBYY FineReader Engine.

See also

Engine
IEngine::PrepareAndOpenBitmap
IEngine::OpenImage
IEngine::OpenBitmapImage
Working with Profiles

PrepareDib Method of the Engine Object

This method creates a copy of the bitmap image that is specified by its bitmap handle. The created copy is converted from DIB (Device-Independent Bitmap) format into ABBYY FineReader Engine internal format and is placed in the folder you specify.

Note: DIB must be created using Windows API.

Visual Basic Syntax

```
Method PrepareDib(
  dibHglobal   As Long,
  destFileName As String,
  xResolution  As Long,
  yResolution  As Long,
  prepareMode  As PrepareImageMode
)
```

C++ Syntax

```
HRESULT PrepareDib(
  long               dibHglobal,
  BSTR               destFileName,
  long               xResolution,
  long               yResolution,
  IPrepareImageMode* prepareMode
);
```

Parameters

dibHglobal

[in] This variable contains the handle of the memory block (HGLOBAL) where the bitmap is saved, statically casted to the **Long** type. This handle should be available to the process that uses ABBYY FineReader Engine.

destFileName

[in] The full path to the destination folder.

xResolution

[in] Horizontal resolution of the bitmap.

yResolution

[in] Vertical resolution of the bitmap.

prepareMode

[in] This variable refers to the **PrepareImageMode** object that stores parameters for bitmap conversion in internal format. This parameter may be 0 in which case default parameters of the image preparation mode are used, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- This method provides a way to get read-write **ImageDocument** object corresponding to the bitmap by applying this method first, and then opening the resulting image in internal format via the **IEngine::OpenImage** method.

- This is a developer concern to delete the files with prepared image when they are no longer necessary. These files are not automatically removed from the disk by ABBYY FineReader Engine.

See also

Engine
IEngine::PrepareAndOpenDib
IEngine::OpenImage
IEngine::OpenDib
Working with Profiles

PrepareImage Method of the Engine Object

This method creates a copy of the specified image file in ABBYY FineReader Engine internal format, in the specified folder on the disk. The output images received from this method may then be opened using the **IEngine::OpenImage** method.

Visual Basic Syntax

```
Method PrepareImage(
  fileName          As String,
  destinationFolder As String,
  prepareMode       As PrepareImageMode,
  pageNumber        As Long,
  passwordCallback  As ImagePasswordCallback,
  documentInfo      As DocumentInfo
) As StringsCollection
```

C++ Syntax

```
HRESULT PrepareImage(
  BSTR                   fileName,
  BSTR                   destinationFolder,
  IPrepareImageMode*     prepareMode,
  long                   pageNumber,
  IImagePasswordCallback* passwordCallback,
  IDocumentInfo*         documentInfo,
  IStringsCollection**   result
);
```

Parameters

fileName

[in] This string contains the full path to the image file that is to be prepared. For example, "C:\MyPictures\MyPicture.bmp".

destinationFolder

[in] This string contains the full path to the folder on disk where the destination files are created. This folder must exist, otherwise an error code is returned.

prepareMode

[in] This variable refers to the **PrepareImageMode** object that stores parameters for image conversion in internal format. This parameter may be 0 in which case default parameters of the image preparation mode are used, or, if a profile has been loaded, the parameters set by this profile are used.

pageNumber

[in] This parameter contains the number of page to process (zero–based). This parameter is optional and may be –1, in which case all pages of the image file are extracted.

passwordCallback

[in] This variable refers to the interface of the user–implemented object of the type **ImagePasswordCallback** which is used to handle possible password requests for accessing images in PDF format. This parameter is optional and may be 0 in which case password–protected files cannot be processed.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being opened.

result

[out, retval] A pointer to **IStringsCollection*** pointer variable that receives the interface pointer of the **StringsCollection** object. This object contains a list of full paths to the images in internal format. These images are located in the destination folder specified by the *destinationFolder* input parameter.

Remarks

- Several output images may appear as the result of preparing a multipage image. In this case, for each image page of the source file a separate output image in internal format is created. If an error occurs while preparing a multipage image, no output image files are generated.

- If the source image was already in ABBYY FineReader Engine internal format and was prepared using the same prepare image mode, it is simply copied into destination directory.

- This is a developer concern to delete the files with prepared image when they are no longer necessary. These files are not automatically removed from the disk by ABBYY FineReader Engine.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Engine
IEngine::PrepareAndOpenImage
IEngine::OpenImage
IEngine::GetNumberOfPagesInImageFile
IImagePasswordCallback
Working with Images
Working with Profiles

PrepareMemoryImage Method of the Engine Object

This method creates a copy of a memory image in the specified folder on disk in ABBYY FineReader Engine internal format.

Visual Basic Syntax

```
Method PrepareMemoryImage(
  format       As MemoryImageFormatEnum,
  width        As Long,
  height       As Long,
  byteWidth    As Long,
  xResolution  As Long,
  yResolution  As Long,
  rawDataPtr   As Long,
  destFileName As String,
  prepareMode  As PrepareImageMode
)
```

C++ Syntax

```
HRESULT PrepareMemoryImage(
  MemoryImageFormatEnum format,
  long                  width,
  long                  height,
  long                  byteWidth,
  long                  xResolution,
  long                  yResolution,
  long                  rawDataPtr,
  BSTR                  destFileName,
  IPrepareImageMode*    prepareMode
);
```

Parameters

format

[in] This parameter of **MemoryImageFormatEnum** type specifies the format of the memory image file that is to be prepared.

width

[in] This parameter specifies the width of the image in pixels.

height

[in] This parameter specifies the height of the image in pixels.

byteWidth

[in] This parameter specifies the width of the line of image in bytes. It should account for justification.

xResolution

[in] This parameter specifies horizontal resolution of the image in dots per inch.

yResolution

[in] This parameter specifies vertical resolution of the image in dots per inch.

rawDataPtr

[in] This parameter is treated as a pointer to memory buffer containing image data. See Memory image format description section for details.

destFileName

[in] This parameter contains the full path to folder where the image in ABBYY FineReader Engine internal format will be saved.

prepareMode

[in] This parameter refers to the **PrepareImageMode** object that defines the mode of image preparation. It may be 0, in which case default values for the **PrepareImageMode** properties will be used, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- This method provides a way to get read-write **ImageDocument** object corresponding to the memory image by applying this method first, and then opening the resulting image in internal format via the **IEngine::OpenImage** method.
- This is a developer concern to delete the files with prepared image when they are no longer necessary. These files are not automatically removed from the disk by ABBYY FineReader Engine.

See also

Engine
IEngine::PrepareAndOpenMemoryImage
IEngine::OpenImage
IEngine::OpenMemoryImage
MemoryImageFormatEnum
Memory image format description
Working with Profiles

AnalyzeAndRecognizePage Method of the Engine Object

This method performs layout analysis, recognition, and page synthesis of the image specified.

Visual Basic Syntax

```
Method AnalyzeAndRecognizePage(
  imageDocument    As ImageDocument,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage,
  layout           As Layout,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognizePage(
  IImageDocument*          imageDocument,
  IPageProcessingParams*   processingParams,
  ISynthesisParamsForPage* synthesisParams,
  ILayout*                 layout,
  IDocumentInfo*           documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be recognized.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores the parameters of the layout analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters have

their properties set to default values, and the recognition language is English), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. After this method is done, it contains the results of the layout analysis and recognition.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets the size and resolution of the layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- All existing blocks are deleted from the *layout*.
- This method call is not equivalent to successive calls to **IEngine::AnalyzePage** and **IEngine::RecognizePage** methods, as recognition information is used for more accurate layout analysis.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
Engine->PrepareAndOpenImage( "D:\\Demo.tif", 0, 0, 0 );
// Create Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
// Create page processing parameters
FREngine::IPageProcessingParamsPtr pPageProcessingParams =
      Engine->CreatePageProcessingParams();
// Now tune it
pPageProcessingParams->DetectBarcodes = VARIANT_TRUE;
// Perform layout analysis, recognition, and page synthesis
Engine->AnalyzeAndRecognizePage( pImageDoc, pPageProcessingParams, 0, pLayout, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")

' Create Layout object
```

```
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()

' Create page processing parameters
Dim PageProcessingParams As FREngine.PageProcessingParams
Set PageProcessingParams = Engine.CreatePageProcessingParams

' Now tune it
PageProcessingParams.DetectBarcodes = True

' Perform layout analysis, recognition, and page synthesis
Engine.AnalyzeAndRecognizePage ImageDoc, PageProcessingParams, Nothing, Layout
```

See also

Engine
IFRPage::AnalyzeAndRecognize
IDocumentAnalyzer::AnalyzeAndRecognizePage
IEngine::AnalyzeAndRecognizePages
Working with Profiles

AnalyzeAndRecognizePages Method of the Engine Object

This method performs layout analysis, recognition, and page synthesis of a collection of images.

Visual Basic Syntax

```
Method AnalyzeAndRecognizePages(
  imageDocuments   As ImageDocumentsCollection,
  layouts          As LayoutsCollection,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognizePages(
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  IPageProcessingParams*     processingParams,
  ISynthesisParamsForPage*   synthesisParams,
  IDocumentInfo*             documentInfo
);
```

Parameters

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be recognized. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layouts

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. After this method is done, it contains the results of layout analysis and recognition.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores the parameters of the layout analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters have their properties set to default values, and the recognition language is English), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

Engine
IFRDocument::AnalyzeAndRecognizePages
IDocumentAnalyzer::AnalyzeAndRecognizePages
IEngine::AnalyzeAndRecognizePage
Working with Profiles

AnalyzePage Method of the Engine Object

This method performs the layout analysis of an image. During layout analysis ABBYY FineReader Engine detects blocks on the image. The blocks determine how and in what order the image areas should be recognized.

Visual Basic Syntax

```
Method AnalyzePage(
  imageDocument    As ImageDocument,
  processingParams As PageProcessingParams,
  layout           As Layout
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzePage(
  IImageDocument*        imageDocument,
  IPageProcessingParams* processingParams,
  ILayout*               layout,
  IDocumentInfo*         documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be analyzed.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. After analysis it contains the results of layout analysis.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is

optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets these parameters for layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- All existing blocks are deleted from *layout.*

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
Engine->PrepareAndOpenImage( "D:\\Demo.tif", 0, 0, 0 );
// Create Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
// Perform page layout analysis
Engine->AnalyzePage( pImageDoc, 0, pLayout, 0 );
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")
' Create Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()
' Perform page layout analysis
Engine.AnalyzePage ImageDoc, Nothing, Layout
...
```

See also

Engine
IFRPage::Analyze
IDocumentAnalyzer::AnalyzePage
IEngine::AnalyzePages
Working with Profiles

AnalyzePages Method of the Engine Object

This method performs the layout analysis of a collection of images. During layout analysis ABBYY FineReader Engine detects blocks on the image. The blocks determine how and in what order the image areas should be recognized.

Visual Basic Syntax

```
Method AnalyzePages(
  imageDocuments   As ImageDocumentsCollection,
  layouts          As LayoutsCollection,
  processingParams As PageProcessingParams,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzePages(
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  IPageProcessingParams*     processingParams,
  IDocumentInfo*             documentInfo
);
```

Parameters

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be recognized. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layouts

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. After this method is done, it contains the results of layout analysis.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

Engine
IFRDocument::AnalyzePages
IDocumentAnalyzer::AnalyzePages
IEngine::AnalyzePage
IEngine::AnalyzeAndRecognizePages
Working with Profiles

RecognizeImageAsPlainText Method of the Engine Object

This method opens an image file, recognizes it and returns recognized text in a special "plain text" format. This format only contains information about recognized text symbols, recognition confidence and positions of these symbols as relative to the recognized image. The resulting plain text is formatted with spaces.

Visual Basic Syntax

```
Method RecognizeImageAsPlainText(
  imageFileName    As String,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage
  passwordCallback As ImagePasswordCallback
) As PlainText
```

C++ Syntax

```
HRESULT RecognizeImageAsPlainText(
  BSTR                     imageFileName,
  IPageProcessingParams*   processingParams,
  ISynthesisParamsForPage* synthesisParams,
  IImagePasswordCallback*  passwordCallback,
  IPlainText**             results
);
```

Parameters

imageFileName

[in] This variable contains the full path to an image file that is to be recognized. If this file is not in ABBYY FineReader Engine internal format, it is prepared with the default values of the **PrepareImageMode** properties, or, if a profile has been loaded, the parameters set by this profile are used.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores analysis and recognition parameters. This parameter is optional and may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters are set to default values, and the recognition language is English), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParamsForPage

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

passwordCallback

[in] This variable refers to the interface of the user-implemented object of the type **ImagePasswordCallback** which is used to handle possible password requests for accessing images in PDF format. This parameter is optional and may be 0 in which case password-protected files cannot be processed.

results

[out, retval] A pointer to **IPlainText*** pointer variable that receives the interface pointer of the **PlainText** output object. This object provides information about recognized symbols and positions of these symbols as relative to the recognized image.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Analyze and recognize the image
 FREngine::IPlainTextPtr text = Engine->RecognizeImageAsPlainText( "D:\\Demo.tif", 0,
0, 0 );

 // Save results as Unicode
 text->SaveToTextFile( "D:\\sample.txt", FREngine::TET_UTF8, FREngine::CP_Null );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Analyze and recognize the image
 Dim Text As FREngine.PlainText
 Set Text = Engine.RecognizeImageAsPlainText("D:\Demo.tif")

 ' Save results as Unicode
 Text.SaveToTextFile "D:\sample.txt", TET_UTF8, CP_Null
```

See also

Engine
IEngine::RecognizeImageDocumentAsPlainText
PlainText
IImagePasswordCallback
Working with Profiles

RecognizeImageDocumentAsPlainText Method of the Engine Object

This method recognizes an image and returns recognized text in a special "plain text" format. This format only contains information about recognized text symbols, recognition confidence and positions of these symbols as relative to the recognized image. The resulting plain text is formatted with spaces.

Visual Basic Syntax

```
Method RecognizeImageDocumentAsPlainText(
  image            As ImageDocument,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage,
  documentInfo     As DocumentInfo
) As PlainText
```

C++ Syntax

```
HRESULT RecognizeImageDocumentAsPlainText(
  IImageDocument*          image,
  IPageProcessingParams*   processingParams,
  ISynthesisParamsForPage* synthesisParams,
  IDocumentInfo*           documentInfo,
  IPlainText**             results
);
```

Parameters

image

[in] This variable refers to the **ImageDocument** object corresponding to the image to be recognized

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores analysis and recognition parameters. This parameter is optional and may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters are set to default values, and the recognition language is English), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

results

[out] A pointer to **IPlainText*** pointer variable that receives the interface pointer of the **PlainText** output object. This object provides information about recognized symbols and positions of these symbols as relative to the recognized image.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

See also

Engine
IEngine::RecognizeImageAsPlainText
PlainText
Working with Profiles

RecognizeImageFile Method of the Engine Object

This method performs layout analysis and recognition of the image file specified, and exports the recognized text into the selected output format. The **RecognizeImageFile** method can process multipage images and export the recognition results to a single file.

Visual Basic Syntax

```
Method RecognizeImageFile(
   imageFileName           As String,
   pageProcessingParams    As PageProcessingParams,
   synthesisParams         As SynthesisParamsForPage
   documentSynthesisParams As SynthesisParamsForDocument,
   exportFormat            As FileExportFormatEnum,
   exportParams            As Unknown,
   passwordCallback        As ImagePasswordCallback,
   outputFileName          As String
 )
```

C++ Syntax

```
HRESULT RecognizeImageFile(
   BSTR                          imageFileName,
   IPageProcessingParams*        pageProcessingParams,
   ISynthesisParamsForPage*      synthesisParams,
   ISynthesisParamsForDocument*  documentSynthesisParams,
   FileExportFormatEnum          exportFormat,
   IUnknown*                     exportParams,
   IImagePasswordCallback*       passwordCallback,
   BSTR                          outputFileName
 );
```

Parameters

imageFileName

[in] This variable contains the full path to an image file that is to be recognized. If this file is not in ABBYY FineReader Engine internal format, it is prepared using the default values of the **PrepareImageMode** properties, or, if a profile has been loaded, the parameters set by this profile are used.

pageProcessingParams

[in] This variable refers to the **PageProcessingParams** object that stores analysis and recognition parameters. This parameter may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters are set to default values, and the recognition language is English), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentSynthesisParams

[in] This variable refers to the **SynthesisParamsForDocument** object that stores parameters of document synthesis. This parameter may be 0. In this case the document is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

exportFormat

[in] This variable of the **FileExportFormatEnum** type specifies the format of the output file into which the recognized text is exported.

exportParams

[in] Pass the export parameters object of type corresponding to your file format through this input parameter. For example, if you are creating an RTF file, create the **RTFExportParams** object, set necessary parameters in it, and pass to this method as the *exporterParams* input parameter. This parameter is optional and may be 0, in which case the parameters of export have default values, or, if a profile has been loaded, the parameters set by this profile are used.

passwordCallback

[in] This variable refers to the interface of the user-implemented object of the type **ImagePasswordCallback** which is used to handle possible password requests for accessing images in PDF format. This parameter is optional and may be 0 in which case password-protected files cannot be processed.

outputFileName

[in] This variable contains the full path to the output file where the recognized text should be exported. If a file at this location already exists, it is overwritten without prompt, or recognized text is append to its end, depending on the mode of export.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method is equivalent to successive calls to the **IEngine::PrepareAndOpenImage** with default parameters, **IEngine::AnalyzeAndRecognizePage** and **IEngine::ExportPage** functions with the specified parameters.

See also

Engine
IEngine::PrepareAndOpenImage
IEngine::AnalyzeAndRecognizePage
IEngine::ExportPage
IImagePasswordCallback
Working with Profiles

RecognizePage Method of the Engine Object

This method recognizes those parts of the specified image that lay inside the blocks of the specified **Layout**.

Visual Basic Syntax

```
Method RecognizePage(
  imageDoc         As ImageDocument,
  synthesisParams  As SynthesisParamsForPage
  extractionParams As ObjectsExtractionParams,
  layout           As Layout,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT RecognizePage(
  IImageDocument*           imageDoc,
  ISynthesisParamsForPage*  synthesisParams,
  IObjectsExtractionParams* extractionParams,
  ILayout*                  layout,
  IDocumentInfo*            documentInfo
);
```

Parameters

imageDoc

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be recognized.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. The blocks in the layout should be created before calling the method. After recognition these blocks will contain the recognized text.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets the size and resolution of the layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- Call this method after you have analyzed or created the layout of the page manually. The old text from blocks, if there is any, is deleted. If the layout contains any table blocks with non-analyzed structure, they will be recognized as containing a single cell corresponding to the whole table.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open the image file
FREngine::IImageDocumentPtr pImageDoc = Engine->PrepareAndOpenImage( "D:\\Demo.tif", 0,
0, 0 );
// Create the Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
// Perform page layout analysis
Engine->AnalyzePage( pImageDoc, 0, pLayout, 0 );
...
// Recognizing
Engine->RecognizePage( pImageDoc, 0, 0, pLayout, 0 );
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open the image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")
' Create the Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()
' Perform page layout analysis
Engine.AnalyzePage ImageDoc, Nothing, Layout
...
' Recognize the image
Engine.RecognizePage ImageDoc, Nothing, Nothing, Layout
...
```

See also

Engine
IFRPage::Recognize
IDocumentAnalyzer::RecognizePage
Working with Profiles

RecognizePages Method of the Engine Object

This method recognizes those parts of the images from the collection that lay inside the blocks of the specified layout collection.

Visual Basic Syntax

```
Method RecognizePages(
  imageDocuments   As ImageDocumentsCollection,
  layouts          As LayoutsCollection,
  synthesisParams  As SynthesisParamsForPage,
  extractionParams As ObjectsExtractionParams,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT RecognizePages(
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  ISynthesisParamsForPage*   synthesisParams,
  IObjectsExtractionParams*  extractionParams,
  IDocumentInfo*             documentInfo
);
```

Parameters

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be recognized. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layouts

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. After this method is done, it contains the results of layout analysis and recognition.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

Engine
IDocumentAnalyzer::RecognizePages
IFRDocument::Recognize

IFRDocument::RecognizePages
Working with Profiles

SynthesizePages Method of the Engine Object

This method performs the document synthesis of a collection of recognized images.

Visual Basic Syntax

```
Method SynthesizePages(
  imageDocumentsCollection   As ImageDocumentsCollection,
  layoutsCollection          As LayoutsCollection,
  synthesisParamsForDocument As SynthesisParamsForDocument,
  documentInfo               As DocumentInfo
)
```

C++ Syntax

```
HRESULT SynthesizePages(
  IImageDocumentsCollection*   imageDocumentsCollection,
  ILayoutsCollection*          layoutsCollection,
  ISynthesisParamsForDocument* synthesisParamsForDocument,
  IDocumentInfo*               documentInfo
);
```

Parameters

imageDocumentsCollection

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be synthesized. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layoutsCollection

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. After this method is done, it contains the results of synthesis.

synthesisParamsForDocument

[in] This variable refers to the **SynthesisParamsForDocument** object that stores parameters of document synthesis. This parameter may be 0. In this case, the document is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object. Then use this object during export, in order the text attributes which were detected during synthesis are available during export.

Return Values

This method returns the standard return codes of the ABBYY FineReader Engine functions.

See also

Engine
IEngine::SynthesizePagesEx
IFRDocument::SynthesizePages
IFRDocument::Synthesize
Working with Profiles

SynthesizePagesEx Method of the Engine Object

This method performs the document synthesis of a collection of recognized images. It requires interface of user-implemented object of type **RecognizedPages**, as its input parameter. This object allows you to pass pages one-by-one rather than as the batch, and thus requires memory for only one recognized page at a time.

Visual Basic Syntax

```
Method SynthesizePagesEx(
  RecognizedPages            As RecognizedPages,
  synthesisParamsForDocument As SynthesisParamsForDocument,
  documentInfo               As DocumentInfo
)
```

C++ Syntax

```
HRESULT SynthesizePagesEx(
  IRecognizedPages*            RecognizedPages,
  ISynthesisParamsForDocument* synthesisParamsForDocument,
  IDocumentInfo*               documentInfo
);
```

Parameters

RecognizedPages

[in] This variable refers to the interface of the user-implemented object of the type **RecognizedPages** which is used to pass recognized texts and images of the synthesized pages one-by-one.

synthesisParamsForDocument

[in] This variable refers to the **SynthesisParamsForDocument** object that stores parameters of document synthesis. This parameter may be 0. In this case the document is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object. Then use this object during export, in order the text attributes which were detected during synthesis are available during export.

Return Values

This method returns the standard return codes of the ABBYY FineReader Engine functions.

See also

Engine
IEngine::SynthesizePages
IFRDocument::SynthesizePages
IFRDocument::Synthesize
Working with Profiles

ExportPage Method of the Engine Object

This method saves recognized text into a file in an external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants.

Visual Basic Syntax

```
Method ExportPage(
  format       As FileExportFormatEnum,
  fileName     As String,
  imageDoc     As ImageDocument,
  layout       As Layout,
  exportParams As Unknown
  documentInfo As DocumentInfo
)
```

C++ Syntax

```
HRESULT ExportPage(
  FileExportFormatEnum format,
  BSTR                 fileName,
  IImageDocument*      imageDoc,
  ILayout*             layout,
  IUnknown*            exportParams
  IDocumentInfo*       documentInfo
);
```

Parameters

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description for the supported file formats.

fileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

imageDoc

[in] This variable refers to the **ImageDocument** that corresponds to the exported page. This parameter must not be 0.

layout

[in] This variable refers to the **Layout** object that contains blocks and recognized text corresponding to the exported page. This parameter may be 0 when exporting a page to PDF (PDF/A) format using **PEM_ImageOnly** mode.

exportParams

[in] Pass the export parameters object of the type corresponding to your file format through this input parameter. For example, if you are saving the text into an RTF file, create an **RTFExportParams** object, set the necessary parameters in it, and pass it to this method as the *exportParams* input parameter. This parameter may be 0, in which case the default values for the export parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object. You should use the same **DocumentInfo** object, which was used as a parameter in the **SynthesizePages** or **SynthesizePagesEx** methods of the **Engine** object. In this case, all the information about document which was received during synthesis is used during export. This parameter may be 0, in which case the text attributes which were detected during synthesis are not available.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Open the image file
 FREngine::IImageDocumentPtr pImageDoc =
 Engine->PrepareAndOpenImage( "D:\\Demo.tif", 0, 0, 0 );

 // Create the Layout object
 FREngine::ILayoutPtr pLayout = Engine->CreateLayout();

 // Analyze and recognize the image
 Engine->AnalyzeAndRecognizePage( pImageDoc, 0, 0, pLayout, 0 );

 // Save the results
 Engine->ExportPage( FREngine::FEF_RTF, "D:\\sample.rtf", pImageDoc, pLayout, 0, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Open the image file
 Dim ImageDoc As FREngine.ImageDocument
 Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")

 ' Create the Layout object
 Dim Layout As FREngine.Layout
 Set Layout = Engine.CreateLayout()

 ' Analyze and recognize the image
 Engine.AnalyzeAndRecognizePage ImageDoc, Nothing, Nothing, Layout

 ' Save the results
 Engine.ExportPage FEF_RTF, "D:\sample.rtf", ImageDoc, Layout
```

See also

Engine
IFRPage::Export
IEngine::ExportPages
Working with Profiles

ExportPages Method of the Engine Object

This method saves recognized text from several pages into a file in an external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants.

Visual Basic Syntax

```
Method ExportPages(
  format         As FileExportFormatEnum,
  fileName       As String,
  imageDocuments As ImageDocumentsCollection,
  layouts        As LayoutsCollection,
  exportParams   As Unknown,
  documentInfo   As DocumentInfo
)
```

C++ Syntax

```
HRESULT ExportPages(
  FileExportFormatEnum       format,
  BSTR                       fileName,
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  IUnknown*                  exportParams
  IDocumentInfo*             documentInfo,
);
```

Parameters

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description for the supported file formats.

fileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object that corresponds to the set of images that belong to the exported pages. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the exported layouts. This parameter must not be 0.

layouts

[in] This variable refers to the **LayoutsCollection** object containing the set of layouts that belong to the exported pages. This parameter may be 0 when exporting pages to PDF (PDF/A) format using **PEM_ImageOnly** mode.

exportParams

[in] Pass the export parameters object of the type corresponding to your file format through this input parameter. For example, if you are saving your text into an RTF file, create a **RTFExportParams** object, set necessary parameters in it, and pass it to this method as the *exportParams* input parameter. This parameter may be 0, in which case the default values for the export parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object. You should use the same **DocumentInfo** object, which was used as a parameter in the **SynthesizePages** or **SynthesizePagesEx** methods of the **Engine** object. In this case, all the information about document which was received during synthesis is used during export. This parameter may be 0, in which case the text attributes which were detected during synthesis are not available.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- This method is similar to **IEngine::ExportPage**, except that it performs export of several pages into a single file.

- To analyze and recognize pages that will be exported into a single file, specify identical values for all the properties of the **SynthesisParamsForPage** object except for the properties **CorrectDynamicRange**, **DetectBackgroundColor**, **DetectTextColor**.

See also

Engine
IFRDocument::ExportPages
IExporter::ExportPages
IExporter::ExportPagesEx
IEngine::ExportPage
Working with Profiles

Image–Related Objects

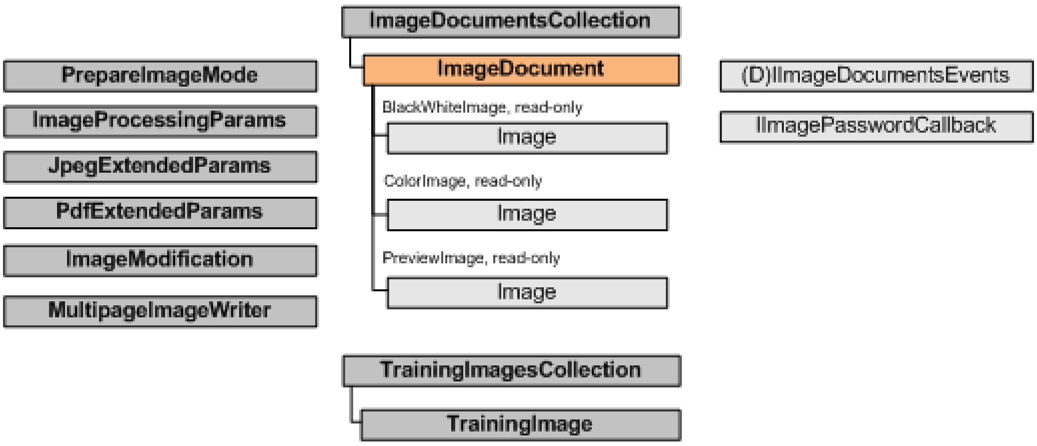
An open image file is represented by an object of the **ImageDocument** type. This object contains a number of image planes, represented by a respective number of the **Image** objects. These planes are: full-size black-and-white copy of the deskewed image, full-size color copy of the deskewed image, and a small color preview, which is optional. ABBYY FineReader Engine also provides several objects, which allow you to modify an image, prepare it for recognition, etc.

This section contains descriptions of the following image-related objects:

- **ImageDocument**
- **IImageDocumentEvents**
- **ImageDocumentsCollection**
- **Image**
- **ImageProcessingParams**
- **PrepareImageMode**
- **JpegExtendedParams**
- **PdfExtendedParams**
- **ImageModification**
- **MultipageImageWriter**
- **IImagePasswordCallback**
- **TrainingImagesCollection**
- **TrainingImage**

You can find additional information about how to work with images in the Working with Images section.

The image-related objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

ImageDocument Object (IImageDocument Interface)

This object corresponds to an open image. Its attributes reflect the attributes of an image. **ImageDocument** object is a root for a collection of **Image** objects, or "image planes". Each image document includes 3 "image planes": black-and-white, color and preview. They are accessible via corresponding properties.

You can set whether **ImageDocument** objects should be created in memory or saved to file on disk with the help of the **CreateImageDocumentsInMemory** property of the **Engine** object. This property may be useful when processing a lot of **ImageDocument** objects simultaneously.

This object supports the **IConnectionPointContainer** interface, which means that it may report events to listeners attached to it, and implementing the **IImageDocumentEvents** interface. It may be declared *WithEvents* in Visual Basic.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BlackWhiteImage	**Image**, read-only	Provides access to the black-and-white image plane of the current **ImageDocument** object.
ColorImage	**Image**, read-only	Provides access to the color image plane of the current **ImageDocument** object.
Id	**Long**, read-only	Stores the ID of the image document.
ImageColorType	**ImageColorTypeEnum**, read-only	Each color plane of the image document is characterized by its own color type. This property specifies the color type for the whole image document as the maximum of the corresponding values for its color planes (black-and-white, gray, color).
ImageRotation	**RotationTypeEnum**, read-only	Specifies rotation that was performed upon the current **ImageDocument** object from the opening or from the last call to the **SaveModified** method.
ImageWasInverted	**Boolean**, read-only	This property set to TRUE specifies that the image colors were inverted after opening or after the last call to **SaveModified** method.
ImageWasMirrored	**Boolean**, read-only	This property set to TRUE specifies that the image was mirrored around the vertical axis after opening or after the last call to **SaveModified** method.
IsInMemory	**Boolean**, read-only	Specifies if the image document is stored in memory only or it is also represented as a folder on disk.
IsModified	**Boolean**, read-only	Specifies if any modifications were made upon the **ImageDocument** object since it was obtained from the image file or from the last call of the **SaveModified** method. Information about modifications is available through the **ImageWasInverted,**

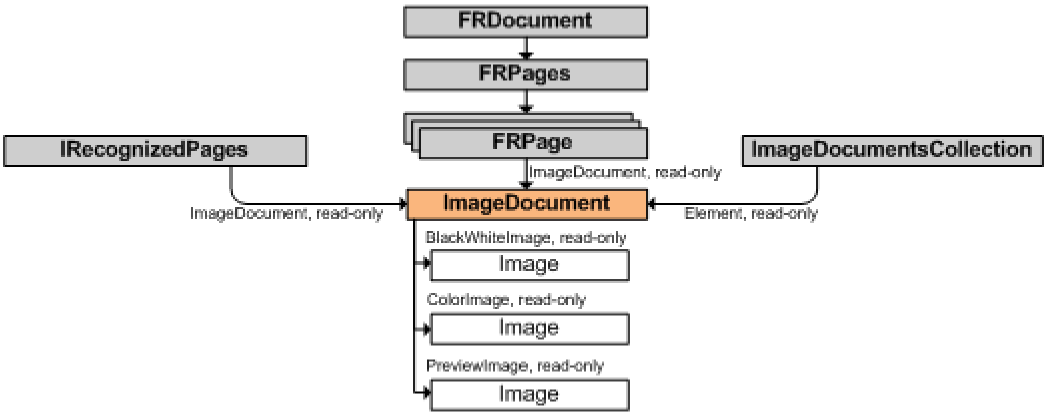
		ImageWasMirrored and **ImageRotation** properties.
IsSkewCorrected	**Boolean**, read-only	This property provides information about whether the skew of the image was fully corrected. This property is TRUE if the image does not need any skew correction (the **CorrectSkewMode** property of the **PrepareImageMode** object is 0) or if the skew was fully corrected during the preparation process as defined by the **CorrectSkewMode** property of the **PrepareImageMode** object. If the value of this property is FALSE, an attempt to correct the skew of the image failed.
Path	**String**, read-only	Stores the path to the folder with object's internal representation on disk. The property contains an empty string, if the value of the **IsInMemory** property is TRUE.
PreviewImage	**Image**, read-only	Provides access to the preview image plane of the current **ImageDocument** object. An open image contains this image plane, only if **IPrepareImageMode::CreatePreview** property was set to TRUE during image preparation. Otherwise the object accessed through this property is NULL.
SkewAngle	**Double**, read-only	Stores the tangent of skew angle that was detected for the image and corrected during opening. If the skew angle is negative the image is rotated clockwise; if the angle is positive the image is rotated counter-clockwise. The image can be rotated around any point. The size of the deskewed image is always larger than the size of the original image.
SourceImageFileFormat	**ImageFileFormatEnum**, read-only	Provides information about format of the source image file of the current **ImageDocument** object. If this information is not available or image was received from scanner, the value of this property is **IFF_UnknownFormat**.
SourceImageScannerInfo	**String**, read-only	Provides information about scanner used to acquire the image. If information about source image parameters is not available or image was received from file, the value of this property is empty string.
SourceImageScanThreshold	**Long**, read-only	Provides information about scanning intensity threshold for the current **ImageDocument** object. If information about source image parameters is not available or image was obtained from file, the value of this property is -1.
SourceImageXResolution	**Long**, read-only	Provides information about horizontal resolution of the source image of the current **ImageDocument** object. If information about source image parameters is not available, the value of this property is 0.
SourceImageYResolution	**Long**, read-only	Provides information about vertical resolution of the source image of the current **ImageDocument** object. If information about source image parameters is not available, the value of this property is 0.

Methods

Name	Description
ChangeResolution	Changes resolution of the image.
ConvertCoordinates	Converts coordinates of a pixel between image planes of the **ImageDocument**.
CorrectSkew	Corrects a skew of the image.
GetTextBackgroundColor	Detects colors of text and background in the specified rectangle on image.
Modify	Allows you to modify the image. This method provides advanced modifications as compared with the **Transform** method.
RemoveCameraBlur	Removes motion blur from the image.
RemoveCameraNoise	Removes ISO noise from the image.

RemoveColorObjects	Removes specified color objects from the whole image or its parts.
RemoveGarbage	Removes garbage (excess dots that are smaller than a certain size) from the image.
SaveImageRegionTo	Saves the parts of the image into a folder on disk. The saved image is in ABBYY FineReader Engine internal format.
SaveTo	Saves the contents of the **ImageDocument** object into a folder on disk. The image is saved in ABBYY FineReader Engine internal format.
SaveToFile	Saves the contents of the **ImageDocument** object into a file.
SaveToMemory	Saves the contents of the **ImageDocument** object into the global memory.
SaveModified	Saves all modifications that were performed upon the current **ImageDocument** object into the files on disk.
SmoothImage	Allows you to smooth the image. This method can be used for gray and color images only.
SubtractColor	Removes the color with the specified hue and saturation from the image.
Transform	Applies a limited set of transformations to the image.

Related objects



Output parameter

This object is the output parameter of the following methods LoadImageDocFromFile, LoadImageDocFromMemory, OpenBitmapImage, OpenDib, OpenImage, OpenMemoryImage, PrepareAndOpenBitmap, PrepareAndOpenDib, PrepareAndOpenImage, PrepareAndOpenMemoryImage of the Engine object.

Input parameter

This object is the input parameter of the following methods:

- **AnalyzeAndRecognizePage**, **AnalyzePage**, **RecognizeImageDocumentAsPlainText**, **RecognizePage**, **ExportPage** of the **Engine** object.

- **AnalyzeAndRecognizePage**, **AnalyzePage**, **AnalyzeRegion**, **AnalyzeTable**, **DetectOrientation**, **ExtractBarcodes**, **FindPageSplitPosition**, **RecognizeBlocks**, **RecognizeImageDocumentAsPlainText**, **RecognizePage**, **RemoveGeometricalDistortions** of the **DocumentAnalyzer** object.

- **AddImage** of the **FRDocument** object.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Open the image file
 FREngine::IImageDocumentPtr pImageDoc =
 Engine->PrepareAndOpenImage( L"D:\\Demo.tif", 0, 0, 0 );

 // Extract image dimensions
 FREngine::IImagePtr image = pImageDoc->ColorImage;
 long width = image->Width;
 long height = image->Height;

 // Obtain text and background colors
 long textColor, backgroundColor;
 pImageDoc->GetTextBackgroundColor( 0, 0, width, height, 0, &textColor,
&backgroundColor);
 ...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Open the image file
 Dim ImageDoc As FREngine.ImageDocument
 Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")

 ' Extract image dimensions
 Dim Image As FREngine.Image
 Set Image = ImageDoc.ColorImage
 Dim Width As Long, Height As Long
 Width = Image.Width
 Height = Image.Height

 ' Obtain text and background colors
 Dim TextColor As Long, BackgroundColor As Long
 ImageDoc.GetTextBackgroundColor 0, 0, Width, Height, 0, TextColor, BackgroundColor
 ...
```

See also

IImageDocumentEvents,
Working with Images
Working with Connectable Objects
Working with Properties

ChangeResolution Method of the ImageDocument Object

This method changes the resolution of the image. The size of the image in pixels remains unchanged.

Visual Basic Syntax

```
Method ChangeResolution(
  newResolution As Long
)
```

C++ Syntax

```
HRESULT ChangeResolution(
  long newResolution,
);
```

Parameters

newResolution

[in] Variable containing the new value for the resolution. Must be greater than 20.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Image resolution can also be changed when opening images if the **OverwriteResolution** or **AutoOverwriteResolution** properties of the **PrepareImageMode** object are set to TRUE. In this case the new resolution will be used for image preprocessing (i.e. for binarization, deskewing, etc.). The **ChangeResolution** method allows you to change the resolution of the existing image.

See also

ImageDocument
PrepareImageMode

ConvertCoordinates Method of the ImageDocument Object

This method converts coordinates of a pixel defined on an image plane to coordinates defined on another image plane. Conversion between all possible pairs of image planes is permitted.

Visual Basic Syntax

```
Method ConvertCoordinates(
  fromPage As ImageTypeEnum,
  toPage   As ImageTypeEnum,
  x        As Long,
  y        As Long
)
```

C++ Syntax

```
HRESULT ConvertCoordinates(
  ImageTypeEnum fromPage,
  ImageTypeEnum toPage,
  long*         x,
  long*         y
);
```

Parameters

fromPage

[in] This variable of the **ImageTypeEnum** type specifies the image page from which the coordinates of pixel are to be converted.

toPage

[in] This variable of the **ImageTypeEnum** type specifies the image page to which the coordinates of pixel are to be converted.

x

[out] This variable stores the horizontal coordinate of the pixel relative to the source image page before the method call, and the horizontal coordinate of the pixel relative to the target image page after the method call.

y

[out] This variable stores the vertical coordinate of the pixel relative to the source image page before the method call, and the vertical coordinate of the pixel relative to the target image page after the method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The coordinates of pixels on the black-and-white image page and the color image page are the same.

See also

ImageTypeEnum

CorrectSkew Method of the ImageDocument Object

This method corrects a skew of the image.

Visual Basic Syntax

```
Method CorrectSkew(
  CorrectSkewFlags As Long
)
```

C++ Syntax

```
HRESULT CorrectSkew(
  long CorrectSkewFlags,
);
```

Parameters

CorrectSkewFlags

[in] The variable may contain any combination of the **CorrectSkewModeEnum** constants. In the case of a combination of several flags, the following order is used: skew correction by squares, skew correction by lines, skew correction by text lines.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

You can also correct image skew when opening images by using the **CorrectSkewMode** property of the **PrepareImageMode** object. In this case the skew will be corrected during image preprocessing.

See also

IImageDocument::CorrectSkew
PrepareImageMode

GetTextBackgroundColor Method of the ImageDocument Object

This method detects colors of text and background in the specified rectangle on image. This rectangle should be specified in the coordinates against the deskewed black-and-white page of the **ImageDocument**.

Visual Basic Syntax

```
Method GetTextBackgroundColor(
  left            As Long,
  top             As Long,
  right           As Long,
  bottom          As Long,
  flags           As Long,
  textColor       As Long,
  backgroundColor As Long
)
```

C++ Syntax

```
HRESULT GetTextBackgroundColor(
  long  left,
  long  top,
  long  right,
  long  bottom,
  long  flags,
  long* textColor,
  long* backgroundColor
);
```

Parameters

left

[in] This parameter contains coordinate of the left border of the rectangle.

top

[in] This parameter contains coordinate of the top border of the rectangle.

right

[in] This parameter contains coordinate of the right border of the rectangle.

bottom

[in] This parameter contains coordinate of the bottom border of the rectangle.

flags

[in] This parameter may either be 0 or DCR_Invert. If DCR_Invert is passed, then the rectangle is considered to be inverted (white text against the black background).

textColor

[out] This parameter receives the value of the text color in rectangle.

backgroundColor

[out] This parameter receives the value of the background color in rectangle.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Text and background colors are detected using information from the deskewed black-and-white page of the **ImageDocument**. But the colors are returned as they are on the color pages of the **ImageDocument**. A pixel of the deskewed black-and-white image plane that lays inside the specified rectangle is considered to belong to text if it is black (white) if the rectangle is not inverted (inverted). If this method fails to detect the colors of text and/or background, it returns the undefined color value (0xFFFFFFFF).

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open the image file
FREngine::IImageDocumentPtr pImageDoc =
Engine->PrepareAndOpenImage( L"D:\\Demo.tif", 0, 0, 0 );
// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Obtain text and background colors
long textColor, backgroundColor;
pImageDoc->GetTextBackgroundColor( 0, 0, width, height, 0, &textColor,
&backgroundColor);
...
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open the image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")
' Extract image dimensions
Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Obtain text and background colors
Dim TextColor As Long, BackgroundColor As Long
ImageDoc.GetTextBackgroundColor 0, 0, Width, Height, 0, TextColor, BackgroundColor
...
```

See also

ImageDocument

Modify Method of the ImageDocument Object

This method modifies the current **ImageDocument**. All modifications defined by the **ImageModification** object are possible.

Visual Basic Syntax

```
Method Modify(
  modification As ImageModification
)
```

C++ Syntax

```
HRESULT Modify(
  ImageModification modification
);
```

Parameters

modification

[in] This variable of the **ImageModification** type specifies the transformations that are to be performed upon the **ImageDocument** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

All the information about the initial image will be lost after the method call.

This method applies modifications to the basic black-and-white and base color image planes of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

See also

ImageModification

RemoveCameraBlur Method of the ImageDocument Object

This method removes motion blur from the specified region of the image. The method is primary designed for preprocessing the images obtained by a digital camera.

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method RemoveCameraBlur(
  region As Region
)
```

C++ Syntax

```
HRESULT RemoveCameraBlur(
  IRegion* region
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the set of rectangles to remove motion blur from. The coordinates of rectangles are related to the deskewed black-and-white page of the **ImageDocument**. This parameter may be 0. In this case the motion blur is removed from the whole image.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method removes blur from the black-and-white and color image planes of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

See also

ImageDocument

RemoveCameraNoise Method of the ImageDocument Object

This method removes ISO noise from the specified region of the image. The method is primary designed for preprocessing the images obtained by a digital camera.

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method RemoveCameraNoise(
  region As Region
)
```

C++ Syntax

```
HRESULT RemoveCameraNoise(
  IRegion* region
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the set of rectangles to remove noise from. The coordinates of rectangles are related to the deskewed black-and-white page of the **ImageDocument**. This parameter may be 0. In this case the noise is removed from the whole image.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method removes noise from the black-and-white and color image planes of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

See also

ImageDocument

RemoveColorObjects Method of the ImageDocument Object

This method allows you to remove color objects from the image. You can remove red, green, blue, or yellow objects from the whole image, or only from some parts of the image: specified region, its background, or only stamps and signatures in this region.

⚠**Important**! This method can be used for color images only.

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method RemoveColorObjects(
  region As Region,
  color  As ObjectsColorEnum,
  mode   As ObjectsTypeEnum
)
```

C++ Syntax

```
HRESULT RemoveColorObjects(
  IRegion*         region,
  ObjectsColorEnum color,
  ObjectsTypeEnum  mode
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the set of rectangles to remove objects from. The coordinates of rectangles are related to the deskewed black-and-white page of the **ImageDocument**. This parameter may be 0. In this case color objects are removed from the whole image.

color

[in] This variable of the **ObjectsColorEnum** type defines the color of the object.

mode

[in] This variable of the **ObjectsTypeEnum** type defines the type of the objects to be removed: objects on the whole image, only background objects, or only color stamps and signatures.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method applies color filtering to the color image plane of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

See also

ImageDocument

RemoveGarbage Method of the ImageDocument Object

This method removes garbage (excess dots that are smaller than a certain size) from the image.

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method RemoveGarbage(
  region      As Region,
  garbageSize As Long
)
```

C++ Syntax

```
HRESULT RemoveGarbage(
  IRegion* region,
  long     garbageSize
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the set of rectangles to remove garbage from. The coordinates of rectangles are related to the deskewed black-and-white page of the **ImageDocument**. This parameter may be 0. In this case the garbage is removed from the whole image.

garbageSize

[in] This variable specifies the maximum area of black dots that are to be considered garbage (in pixels). The value of -1 for this input parameter tells ABBYY FineReader Engine to automatically calculate the size of garbage.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method removes garbage from the black-and-white image plane of the **ImageDocument**.

See also

ImageDocument

SaveTo Method of the ImageDocument Object

This method saves the contents of the **ImageDocument** object into a folder on disk. The image is saved in ABBYY FineReader Engine internal format.

Visual Basic Syntax

```
Method SaveTo(
  folderName As String
)
```

C++ Syntax

```
HRESULT SaveTo(
   BSTR folderName
);
```

Parameters

folderName

[in] This parameter stores the full path to the folder. For example, "C:\MyPic".

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ImageDocument

SaveToFile Method of the ImageDocument Object

This method saves the contents of the **ImageDocument** object into a file. The file is saved in a format witch cannot be viewed in any external program. The **ImageDocument** object saved using this method can be opened with the help of the **IEngine::LoadImageDocFromFile** method only.

Visual Basic Syntax

```
Method SaveToFile(
  fileName As String
)
```

C++ Syntax

```
HRESULT SaveToFile(
   BSTR fileName
);
```

Parameters

fileName

[in] This parameter stores the full path to the file. For example, "C:\MyPic.imageDoc". If this file already exists, it is overwritten without prompt.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method is not suitable for saving images, as it saves the contents of the **ImageDocument** object in a format not suitable for viewing. It is designed for situations when it is not possible to use the **IImageDocument::SaveTo** method which saves the object's contents into a folder on disk.

To save an image in a format suitable for viewing, use the **IImage::WriteToFile** method.

See also

ImageDocument
IEngine::LoadImageDocFromFile

SaveToMemory Method of the ImageDocument Object

This method saves the contents of the **ImageDocument** object into the global memory and returns an HGLOBAL handle — casted to the **Long** type, of the memory area allocated for the object. It is user's responsibility to free this memory when it is no longer needed. As the memory is allocated by the **GlobalAlloc** API function, it should be freed by the **GlobalFree** function. The size of the memory area that the object allocates can be obtained by calling the **GlobalSize** function.

Visual Basic Syntax

```
Method SaveToMemory() As Long
```

C++ Syntax

```
HRESULT SaveToMemory(
  long* hGlobal
);
```

Parameters

hGlobal

[out] A pointer to a **long** variable that receives the HGLOBAL handle — casted to **long** — of the memory area allocated for the **ImageDocument** object. Should not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ImageDocument
IEngine::LoadImageDocFromMemory

SaveModified Method of the ImageDocument Object

This method saves all the modifications that were performed upon the current **ImageDocument** object into the files on disk. The method can be used only if the image document is represented as a folder on disk (the **IImageDocument::IsInMemory** property is FALSE). It does not overwrite the source image file. It sets the value of the **IImageDocument::IsModified** property to FALSE.

Visual Basic Syntax

```
Method SaveModified()
```

C++ Syntax

```
HRESULT SaveModified();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ImageDocument

SmoothImage Method of the ImageDocument Object

This method allows you to smooth the image by averaging over the square neighborhood.

⚠**Important**! This method can be used for gray and color images only.

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method SmoothImage(
  region   As Region,
  areaSize As Long
)
```

C++ Syntax

```
HRESULT SmoothImage(
  IRegion* region,
  long     areaSize
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the set of rectangles to be smoothed. The coordinates of rectangles are related to the deskewed black-and-white page of the **ImageDocument**. This parameter may be 0. In this case the whole image is smoothed.

areaSize

[in] This variable specifies the side of the square neighborhood. Must be an odd number above or equal to 3.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method smoothes the color image plane of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

See also

ImageDocument

SubtractColor Method of the ImageDocument Object

This method removes the color with the specified hue and saturation from the image. The method is primary designed for filtering color on images of passports and certificates. Such preprocessing allows the program to pick out texts on the images.

⚠**Important**! This method can be used for color images only.

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method SubtractColor(
  referenceHue       As Byte,
  saturationBoundary As Long
)
```

C++ Syntax

```
HRESULT SubtractColor(
  byte referenceHue,
  long saturationBoundary
);
```

Parameters

referenceHue

[in] This parameter specifies the hue, which is to be filtered, in HSL representation. The value of this parameter must be in range from 0 to 255. The value 0 corresponds to red color, 43 — to yellow, 85 — to green, 171 — to blue, 213 — to purple.

saturationBoundary

[in] This variable specifies saturation boundary in HSL representation. The value of this parameter must be in range from 1 to 254. If the saturation value is higher than the value of this parameter, the specified hue will be removed from the image. For example, for passports the value in range from 25 to 35 is suitable.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method applies color filtering to the color image plane of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

See also

ImageDocument

Transform Method of the ImageDocument Object

This method transforms the current **ImageDocument**. Among available transformations are inversion of colors, mirroring and rotation by 90, 180 and 270 degrees.

This method reports events to the listeners attached to the **IConnectionPointContainer** interface of the **ImageDocument** object.

Visual Basic Syntax

```
Method Transform(
  rotation As RotationTypeEnum,
  mirror   As Boolean,
  invert   As Boolean
)
```

C++ Syntax

```
HRESULT Transform(
  RotationTypeEnum rotation,
  VARIANT_BOOL     mirror,
  VARIANT_BOOL     invert
);
```

Parameters

rotation

[in] This variable of the **RotationTypeEnum** type defines the angle of rotation for the image.

mirror

[in] This parameter specifies whether the image should be mirrored around the vertical axis during transformation.

invert

[in] This parameter specifies if image colors should be inverted during transformation.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method applies transformations to the basic black-and-white and basic color image planes of the **ImageDocument**. All the other pages are deleted from the **ImageDocument**. They will be created upon demand.

The sequence of geometrical transformations is as follows: first the rotation by the specified angle is performed, and then the image is mirrored around the vertical axis.

See also

ImageDocument
RotationTypeEnum

SaveImageRegionTo Method of the ImageDocument Object

This method saves the parts of the image restricted by the specified set of rectangles into a folder on disk. The saved image is in the ABBYY FineReader Engine internal format.

The compression of a color image plane is defined by the **ImageCompression** property of the **PrepareImageMode** object, while a black-and-white image plane is always saved with the CCITT4 compression. This new image has the rectangle that fully bounds the set of rectangles. Parts of image that do not lay inside these rectangles but lay inside the bounding rectangle are filled in with white color.

Visual Basic Syntax

```
Method SaveImageRegionTo(
  folderName  As String,
  rects       As Region,
  prepareMode As PrepareImageMode
)
```

C++ Syntax

```
HRESULT SaveImageRegionTo(
  BSTR              folderName,
  IRegion*          rects,
  IPrepareImageMode* prepareMode
);
```

Parameters

folderName

[in] This parameter stores the full path to the folder. For example, "C:\MyPic".

rects

[in] This parameter of the **Region** type specifies the set of rectangles that are to be copied from the source image into the target one. The coordinates of rectangles are related to the deskewed black-and-white page of the **ImageDocument**.

prepareMode

[in] This parameter of the **PrepareImageMode** type specifies the parameters of image preparation during the rectangles extraction.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
void DoHorizontalSplit( FREngine::IImageDocument* imageDoc, long position1, long
position2 )
 {
  // Obtain the black-and-white image plane
  FREngine::IImagePtr image = imageDoc->BlackWhiteImage;

  // Create regions for the top and bottom parts of the image
  FREngine::IRegionPtr region1 = Engine->CreateRegion();
  FREngine::IRegionPtr region2 = Engine->CreateRegion();

  // Add rectangles for the image parts into regions
  region1->AddRect( 0, 0, image->Width, position1 );
  region2->AddRect( 0, position2, image->Width, image->Height );

  // Save the image parts into files with unique names
  // and with default PrepareImageMode
  imageDoc->SaveImageRegionTo( L"D:\\MyPic1", region1, 0 );
  imageDoc->SaveImageRegionTo( L"D:\\MyPic2", region2, 0 );
 }
```

Visual Basic code

```
Private Sub DoHorizontalSplit(ImageDoc As FREngine.ImageDocument, Position1 As Long,
Position2 As Long)
       ' Obtain the black-and-white image plane
       Dim Image As FREngine.Image
       Set Image = ImageDoc.BlackWhiteImage

       ' Create regions for the top and bottom parts of the image
       Dim Region1 As FREngine.Region
       Dim Region2 As FREngine.Region
       Set Region1 = Engine.CreateRegion
       Set Region2 = Engine.CreateRegion

       ' Add rectangles for the image parts into regions
       Region1.AddRect 0, 0, Image.Width, Position1
       Region2.AddRect 0, Position2, Image.Width, Image.Height
       ' Save the image parts into files with unique names
       ' and with default PrepareImageMode
       ImageDoc.SaveImageRegionTo "D:\MyPic1", Region1
       ImageDoc.SaveImageRegionTo "D:\MyPic2", Region2
```

```
End Sub
```

See also

ImageDocument
PrepareImageMode

IImageDocumentEvents Interface

This is callback interface that is used for reporting events from the **ImageDocument** object to the listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to receive notifications from the **ImageDocument** object should declare it *WithEvents* and implement the following Sub:

```
Public WithEvents imageDoc As FREngine.ImageDocument
Private Sub imageDoc_TransformationMade(ByVal rotation As RotationTypeEnum,
                                        ByVal wasMirrored As Boolean,
                                        ByVal wasInverted As Boolean)
...
End Sub
```

Methods

Name	Description
TransformationMade	Called by ABBYY FineReader Engine when some transformation was made with the **ImageDocument**.

See also

ImageDocument
Working with Connectable Objects

TransformationMade Method of the IImageDocumentEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine when some transformation was made upon image either explicitly by call of the **IImageDocument::Transform** method, or internally by ABBYY FineReader Engine. The latter situation may occur when an image with wrong orientation is analyzed and the **IPageProcessingParams::DetectOrientation** property is TRUE. Implementation of this method may reload the image to show the changes to the user, as it is done in ABBYY FineReader. Reloading of the image is necessary, if, say, the client application receives events from the **DocumentAnalyzer** object, and fills out recognized parts of the image with color.

Visual Basic Syntax

```
Sub TransformationMade(
  ByVal rotation    As RotationTypeEnum,
  ByVal wasMirrored As Boolean,
  ByVal wasInverted As Boolean
)
```

C++ Syntax

```
HRESULT TransformationMade(
  RotationTypeEnum rotation,
  VARIANT_BOOL     wasMirrored,
  VARIANT_BOOL     wasInverted
);
```

Parameters

rotation

[in] This variable of type **RotationTypeEnum** specifies what kind of rotation was performed upon the image.

wasMirrored

[in] This Boolean variable specifies if the image was inverted.

wasInverted

[in] This Boolean variable specifies whether image colors were inverted.

Return Values

The return value of this method is ignored.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

If the image was transformed, its pixel dimensions may change.

See also

ImageDocument
IImageDocumentEvents

ImageDocumentsCollection Object (IImageDocumentsCollection Interface)

This object represents a collection of **ImageDocument** objects. It serves as a storage to pass various sets of parameters into those ABBYY FineReader Engine functions that require them. It may also be return value of ABBYY FineReader Engine methods.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

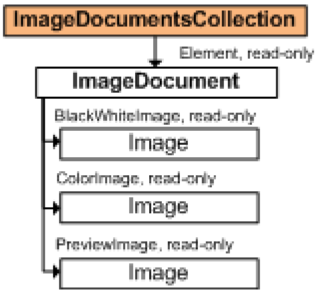
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**ImageDocument**	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



Output parameter

This collection is the output parameter of the **CreateImageDocumentsCollection** method of the **Engine** object.

Input parameter

This collection is the input parameter of the following methods:

- **AnalyzeAndRecognizePages**, **AnalyzePages**, **ExportPages**, **RecognizePages**, **SynthesizePages** methods of the **Engine** object.
- **AnalyzeAndRecognizePages**, **AnalyzePages**, **RecognizePages** methods of the **DocumentAnalyzer** object.

- **ExportPages** method of the **Exporter** object.

See also

Working with Properties

Image Object (IImage Interface)

This object represents a single "image plane" (black-and-white, color or preview) of an open image. It gives a user access to properties of this "image plane", such as its geometrical parameters and resolution. It allows you to get a bitmap handle corresponding to this image.

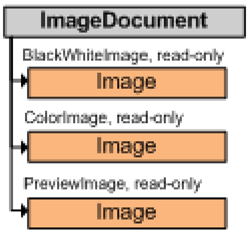
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Height	**Long**, read-only	Stores the height of the current image plane in pixels.The height of the black-and-white image plane of an image is equal to the height of the color image plane of the image.
ImageColorType	**ImageColorTypeEnum**, read-only	Specifies the color type for the current image plane (black-and-white, gray, color). By agreement on the ABBYY FineReader Engine API, if the **ImageDocument** object represents black-and-white image, an image plane received from **IImageDocument::ColorImage** property will actually be of black-and-white color type. This property provides information about actual color type of the current image plane.
ImageDocument	**ImageDocument**, read-only	This property refers to the parent **ImageDocument** object of the current image plane. As **Image** object cannot exist independently of the **ImageDocument** object, this property always refers to a valid **ImageDocument** object.
Width	**Long**, read-only	Stores the width of the current image plane in pixels. The width of the black-and-white image plane of an image is equal to the width of the color image plane of the image.
XResolution	**Long**, read-only	Stores the horizontal resolution of the current image plane in pixels per inch. The resolution of the black-and-white image plane of an image is equal to the resolution of the color image plane of the image.
YResolution	**Long**, read-only	Stores the vertical resolution of the current image plane in pixels per inch. The resolution of the black-and-white image plane of an image is equal to the resolution of the color image plane of the image.

Methods

Name	Description
EstimateBitmapSize	Estimates the size of the bitmap that will be returned by the **GetPicture** method.
GetPicture	Returns a handle of bitmap that corresponds to the current image plane.
WriteToFile	Writes the image of the current image plane into an image file.

Related objects



Input parameter

This object is the input parameter of the **IMultipageImageWriter::AddPage** method.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Open the image file
 FREngine::IImageDocumentPtr pImageDoc = Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
0, 0, 0 );

 // Extract image dimensions
 FREngine::IImagePtr image = pImageDoc->ColorImage;
 long width = image->Width;
 long height = image->Height;

 // Obtain text and background colors
 long textColor, backgroundColor;
 pImageDoc->GetTextBackgroundColor( 0, 0, width, height, 0, &textColor,
&backgroundColor);
 ...
 // Create and initialize the IimageModification object
 FREngine::IImageModificationPtr imageModification =
  Engine->CreateImageModification();
 ...
 // Saving the modified image
 image->WriteToFile( L"D:\\sample.png", FREngine::IFF_PngColorPng, imageModification, 0
);
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Open the image file
 Dim ImageDoc As FREngine.ImageDocument
 Set ImageDoc = Engine.PrepareAndOpenImage( "D:\Demo.tif")

 ' Extract image dimensions
 Dim Image As FREngine.Image
 Set Image = ImageDoc.ColorImage
 Dim Width As Long, Height As Long
 Width = Image.Width
 Height = Image.Height
 ...
 ' Obtain text and background colors
 Dim TextColor As Long, BackgroundColor As Long
 ImageDoc.GetTextBackgroundColor 0, 0, Width, Height, 0, TextColor, BackgroundColor
 ' Create and initialize the IimageModification object
 Dim ImageModification As FREngine.ImageModification
 Set ImageModification = Engine.CreateImageModification
 ...
 ' Save the modified image
 Image.WriteToFile "D:\sample.png", IFF_PngColorPng, ImageModification

 Set Image = Nothing
```

See also

ImageDocument
Working with Images
Working with Properties

EstimateBitmapSize Method of the Image Object

This method estimates the size of memory that is to be allocated for the bitmap returned from the **IImage::GetPicture** method called with the same parameters. Thus, its input parameters are analogous to those of the **IImage::GetPicture.**

Visual Basic Syntax

```
Method EstimateBitmapSize(
  modification As ImageModification,
  mode         As Long
) As Long
```

C++ Syntax

```
HRESULT EstimateBitmapSize(
  ImageModification* modification,
  long               mode,
  long*              size
);
```

Parameters

modification

[in] This parameter of the **ImageModification** type specifies modifications that are performed upon image. It may include clipping rectangles, in which case only specified parts of the image are passed, stretch ratio, painting rectangles for filling up parts of the image with color and so on. This parameter may be 0, and in this case no modifications will be performed upon the image page and it will be passed "as is".

mode

[in] This parameter may be any combination of **GP_** prefixed flags or contain 0.

GP_ flag value	Description
GP_ScaleToGray	For black-and-white image page and the stretch ratio <1, as defined by the **ImageModification** object, a gray bitmap will be created and returned by this method.
GP_ReduceToHighColor	For color image page the presence of this flag reduces the number of its colors to 65536, that corresponds to high color.

size

[out] A pointer to **long** variable that receives the return value of this method. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ImageModification
IImage::GetPicture

GetPicture Method of the Image Object

This method returns a part of image in the Device Independent Bitmap (DIB) format. The return value of this method may be directly casted to the HBITMAP type.

Visual Basic Syntax

```
Method GetPicture(
  modification As ImageModification,
  mode         As Long
) As Long
```

C++ Syntax

```
HRESULT GetPicture(
  ImageModification* modification,
  long               mode,
  long*              bitmapHandle
);
```

Parameters

modification

[in] This parameter of type **ImageModification** defines modifications that are performed upon image. It may include clipping rectangles, in which case only specified parts of the image are passed, stretch ratio, painting rectangles for filling up parts of the image with color and so on. This parameter may be 0, and in this case no modifications will be performed upon the image page and it will be passed "as is".

mode

[in] This parameter may be any combination of **GP_** prefixed flags or contain 0.

GP_ flag value	**Description**
GP_ScaleToGray	For black-and-white image page and the stretch ratio <1, as defined by the **ImageModification** object, a gray bitmap will be created and returned by this method.
GP_ReduceToHighColor	For color image page the presence of this flag reduces the number of its colors to 65536, that corresponds to high color.

bitmapHandle

[out] A pointer to **long** variable that receives the return value of this method. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The returned bitmap is created via the **CreateDIBSection** method and passed under client's ownership. Thus after using this bitmap, it is necessary to call the **DeleteObject** method for it.

See also

ImageModification
IImage::EstimateBitmapSize

WriteToFile Method of the Image Object

This method saves a copy of the current image plane into an image file in the specified format.

Visual Basic Syntax

```
Method WriteToFile(
  fileName     As String,
  fileFormat   As ImageFileFormatEnum,
  modification As ImageModification,
  parameters   As Object
)
```

C++ Syntax

```
HRESULT WriteToFile(
  BSTR                 fileName,
  ImageFileFormatEnum  fileFormat,
  ImageModification*   modification,
  IUnknown*            parameters
);
```

Parameters

fileName

[in] This parameter contains the full path to the image file where the image is saved. For example, "C:\MyPic.bmp". If a file in this path already exists, it is overwritten without prompt.

fileFormat

[in] A variable of the **ImageFileFormatEnum** type that specifies the format of the output file. Not all formats defined by this enumeration are supported for writing.

modification

[in] This parameter of the **ImageModification** type defines modifications that are performed upon image before writing it into file. This parameter is optional and may be 0, in which case no modifications are performed upon image.

parameters

[in] This variable may refer to the **JpegExtendedParams** object that defines parameters for saving the image to JPEG format, or **PdfExtendedParams** object that defines parameters for saving the image to PDF format. This parameter is optional and may be 0. In this case the image is saved with lossless JPEG 2000 compression.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open the image file
FREngine::IImageDocumentPtr pImageDoc = Engine->PrepareAndOpenImage(
       L"D:\\Demo.tif", 0, 0, 0 );
// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Obtain text and background colors
long textColor, backgroundColor;
pImageDoc->GetTextBackgroundColor( 0, 0, width, height, 0, &textColor,
&backgroundColor);
...
// Create and initialize the IimageModification object
FREngine::IImageModificationPtr imageModification =
       Engine->CreateImageModification();
...
// Saving the modified image
image->WriteToFile( L"D:\\sample.png",
       FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open the image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")
' Extract image dimensions
Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
...
' Obtain text and background colors
Dim TextColor As Long, BackgroundColor As Long
ImageDoc.GetTextBackgroundColor 0, 0, Width, Height, 0, TextColor, BackgroundColor
' Create and initialize the IimageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
...
' Save the modified image
Image.WriteToFile "D:\sample.png", _
       IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

Working with Images
ImageFileFormatEnum
ImageModification

ImageProcessingParams Object (IImageProcessingParams Interface)

This object specifies how an image will be preprocessed before analysis and recognition. Most types of blocks have child objects of the **ImageProcessingParams** type that are available through the corresponding properties. Image processing parameters should be set for each block that is to be recognized, if any non-trivial image preprocessing upon that block is needed. The **ImageProcessingParams** object may specify image rotation and its mirroring around the vertical axis. Rotation is the first operation in sequence of geometrical transformation, and mirroring is the second one. All properties of a newly created object of this type are set to reasonable defaults. To get info on the default value of this or that property see its description.

The **ImageProcessingParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

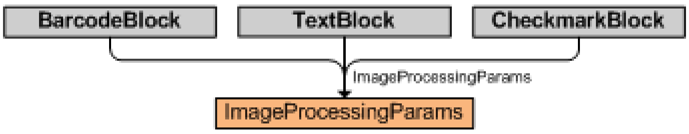
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
InvertImage	**Boolean**	Specifies if the image colors are inverted during preprocessing. This property is FALSE by default.
MirrorImage	**Boolean**	Specifies if the image is mirrored around the vertical axis during preprocessing. This property is FALSE by default.
RotationType	**RotationTypeEnum**	Specifies what type of rotation will be performed upon the image during its preprocessing. This property is RT_NoRotation by default, which means that image is not rotated.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreateImageProcessingParams** method of the **Engine** object.

See also

BarcodeBlock
TextBlock
CheckmarkBlock
Tuning Analysis, Recognition, and Synthesis Parameters
Working with Properties

PrepareImageMode Object (IPrepareImageMode Interface)

This object contains different attributes specifying how an image will be prepared during conversion to the internal format by the **IEngine::PrepareImage** and **IEngine::PrepareBitmap** methods and other similar methods. All properties of a newly created object of this type are set to reasonable defaults. To know about the default value of this or that property, see its description. The sequence of the transformations upon the prepared image is the following: first the rotation is performed, and then the image is mirrored.

The **PrepareImageMode** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re–created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read–only	Returns the **Engine** object.
AutoOverwriteResolution	**Boolean**	Specifies whether resolution of the prepared image should be automatically overwritten. The property is only available, if the value of the **OverwriteResolution** property is FALSE. If the value of the **AutoOverwriteResolution** property is TRUE, ABBYY FineReader Engine will automatically detect and overwrite image resolution. By default, the value of the property is TRUE.
CorrectSkewByBlackSquaresHorizontally	**Boolean**	This property is obsolete. Use the **CorrectSkewMode** property instead.
CorrectSkewByBlackSquaresVertically	**Boolean**	This property is obsolete. Use the **CorrectSkewMode** property instead.
CorrectSkewByHorizontalLines	**Boolean**	This property is obsolete. Use the **CorrectSkewMode** property instead.
CorrectSkewByHorizontalText	**Boolean**	This property is obsolete. Use the **CorrectSkewMode** property instead.
CorrectSkewByVerticalLines	**Boolean**	This property is obsolete. Use the **CorrectSkewMode** property instead.
CorrectSkewByVerticalText	**Boolean**	This property is obsolete. Use the **CorrectSkewMode** property instead.
CorrectSkewMode	**Long**	Specifies the mode of skew correction. The value of this property is an OR superposition of the **CorrectSkewModeEnum** enumeration constants which denote the types of skew correction. 0 means do not correct skew. By default, this property is set to CSM_CorrectSkewByHorizontalText | CSM_CorrectSkewByVerticalText.
CreatePreview	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to create preview page for the prepared image. By default, this property is set to FALSE.
DiscardColorImage	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to leave only black–and–white planes in the prepared image. By default, this property is set to FALSE.
ImageCompression	**ImageCompressionEnum**	This property specifies how an image should be compressed during conversion to the internal format. By default, this property is set to IC_Auto.
InvertImage	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to invert colors of the prepared image. By default, this property is set to FALSE.

MirrorImage	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to mirror the prepared image around its vertical axis. By default, this property is set to FALSE.
OverwriteResolution	**Boolean**	Allows you to overwrite resolution of the prepared image. The resolution is overwritten depending on the values of the **XResolutionToOverwrite** and **YResolutionToOverwrite** properties. In this case the new resolution will be used for image preprocessing (i.e. for binarization, deskewing, etc.). Image resolution can be automatically overwritten (see the description of the **AutoOverwriteResolution** property). By default, this property is set to FALSE. See also **IImageDocument::ChangeResolution**.
PreviewHeight	**Long**	Specifies the height in pixels of the preview page. This property is valid only if the **CreatePreview** property is TRUE, otherwise it is ignored. By default, this property is set to 90.
PreviewWidth	**Long**	Specifies the width in pixels of the preview page. This property is valid only if the **CreatePreview** property is TRUE, otherwise it is ignored. By default, this property is set 64.
Rotation	**RotationTypeEnum**	This property specifies the rotation angle to apply to the image during preparation. It specifies no rotation by default.
XResolutionToOverwrite	**Long**	Specifies the horizontal resolution of the original image in dpi. This value is used to overwrite resolution of the prepared image when resolution of the original image is not specified or incorrect and only if the **OverwriteResolution** property is TRUE. ABBYY FineReader Engine works with the prepared image which horizontal and vertical resolutions are equal, therefore the program stretches the image so that the horizontal and vertical resolutions of the prepared image are identical and equal to the maximum of **XResolutionToOverwrite** and **YResolutionToOverwrite**. By default, this property is set to 300.
YResolutionToOverwrite	**Long**	Specifies the vertical resolution of the original image in dpi. This value is used to overwrite resolution of the prepared image when resolution of the original image is not specified or incorrect and only if the **OverwriteResolution** property is TRUE. ABBYY FineReader Engine works with the prepared image which horizontal and vertical resolutions are equal, therefore the program stretches the image so that the horizontal and vertical resolutions of the prepared image are identical and equal to the maximum of **XResolutionToOverwrite** and **YResolutionToOverwrite**. By default, this property is set to 300.

Methods

Name	**Description**
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.

SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Output parameter

This object is the output parameter of the **CreatePrepareImageMode** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **AddImageFile**, **AddImageFileWithPassword**, **AddImageFileWithPasswordCallback** of the **FRDocument** object,
- **CreateFRDocumentFromImage**, **PrepareImage**, **PrepareMemoryImage**, **PrepareAndOpenImage**, **PrepareAndOpenMemoryImage**, **PrepareBitmap**, **PrepareAndOpenBitmap**, **PrepareDib**, **PrepareAndOpenDib** of the **Engine** object,
- **SaveImageRegionTo** of the **ImageDocument** object.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Create and customize image loading parameters
FREngine::IPrepareImageModePtr prepareImageMode = Engine->CreatePrepareImageMode();
// Turn on mirroring
prepareImageMode->MirrorImage = VARIANT_TRUE;
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
      Engine->PrepareAndOpenImage( Engine->Path +
"\\..\\Samples\\SampleImages\\Demo.tif",
      prepareImageMode, 0, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Create and customize image loading parameters
Dim PrepareImageMode As FREngine.PrepareImageMode
Set PrepareImageMode = Engine.CreatePrepareImageMode
' Turn on mirroring
PrepareImageMode.MirrorImage = True
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
"D:\Demo.tif", PrepareImageMode)
```

See also

Working with Images
Working with Properties

JpegExtendedParams Object (IJpegExtendedParams Interface)

This object provides functionality for tuning the parameters of saving an image to JPEG format (**IFF_JpegGrayJfif, IFF_JpegColorJfif, IFF_Jpeg2kGray, IFF_Jpeg2kColor, IFF_TiffGrayJpegJfif** and **IFF_TiffColorJpegJfif** image format types) using the **IImage::WriteToFile** function. A pointer to this object is passed into the **IImage::WriteToFile** function as an input parameter, and thus affects the size and quality of the resulting image. All properties of a newly created object of this type are set to reasonable defaults. See the description of particular property for its default value.

The **JpegExtendedParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
UseJpeg6Compression	**Boolean**	The property is obsolete. The value of this property is ignored.
Quality	**Long**	Stores the value of the JPEG quality in percent. The default value for this property is 50.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.
SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Output parameter

This object is the output parameter of the **CreateJpegExtendedParams** method of the **Engine** object.

Input parameter

This object is the input parameter of the **WriteToFile** method of the **Image** object.

See also

ImageFileFormatEnum
Working with Images
Working with Properties

PdfExtendedParams Object (IPdfExtendedParams Interface)

This object provides functionality for tuning the parameters of saving an image to PDF format (**IFF_PDF** image format type) using the **IImage::WriteToFile** function. A pointer to this object is passed into the **IImage::WriteToFile** function as an input parameter, and thus affects the size and quality of the resulting image. All properties of a newly created object of this type are set to reasonable defaults. See the description of particular property for its default value.

Note: The earliest version of the PDF file which matches the specified properties of the **PDFEncryptionInfo** object is selected as the version of the PDF file.

- The earliest file version available is the version **1.3**.
- If at least one of the **PermissionFillFormFields**, **PermissionExtractTextAndGraphicsExt**, **PermissionAssembleDoc**, **PermissionPrintExt** properties of the **PDFEncryptionInfo** object, or the encryption key length exceeds 40 bits, the PDF file version will be **1.4**.
- If the **IPDFEncryptionInfo::UseAES** property is TRUE, the version will be **1.6**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Author	**String**	Specifies the author of the PDF file. The default value is an empty string.
Creator	**String**	Specifies the creator of the PDF file. The default value is "ABBYY FineReader Engine 10".

EncryptionInfo	**PDFEncryptionInfo**	Specifies encryption parameters of the PDF file. The property returns a copy of the **PDFEncryptionInfo** object but not a reference to it. To modify the value of the property, you must assign the value of this property to the **PDFEncryptionInfo** object, change the necessary encryption parameters, and then assign this object back to the property.
Keywords	**String**	Specifies the keywords of the PDF file. The default value is an empty string.
PDFVersion	**PDFVersionEnum**	Specifies the version of the PDF file. The version should not conflict with the specified export parameters (see the note above for details). The default value for this property is PVN_Auto which specifies that the version is detected automatically.
Producer	**String**	Specifies the producer of the PDF file. The default value is an empty string.
Subject	**String**	Specifies the subject of the PDF file. The default value is an empty string.
Title	**String**	Specifies the title of the PDF file. The default value is an empty string.

Related objects

PdfExtendedParams
PDFEncryptionInfo
PDFEncryptionInfo

Output parameter

This object is the output parameter of the **CreatePdfExtendedParams** method of the **Engine** object

Input parameter

This object is the input parameter of the **WriteToFile** method of the **Image** object.

See also

Working with Images
Working with Properties

ImageModification Object (IImageModification Interface)

This object is used to store parameters of image modification. Method **IImageDocument::Modify** that is used to modify an image, together with some other methods, takes a reference to this object as an input parameter. The **ImageModification** allows a wide range of operations upon an image such as stretching, setting clip regions, inversion regions, paint regions, replace pixels regions, erase text regions, remove garbage regions. The image is modified as follows:

- The color of text and the size of garbage in regions is determined.
- Image part inside the clipping regions is cut.
- "Paint" regions are filled in with the corresponding color.
- Colors inside the "invert" regions are inverted.
- Black dots inside the "replace black pixels" regions are replaced with the dots of the corresponding color, and the black text from "erase text" regions is erased at the same time.
- White dots inside the "replace white pixels" regions are replaced with the dots of the corresponding color, and the white text from "erase text" regions is erased at the same time.
- The garbage inside the "remove garbage" regions is cleaned up. This modification can be applied only to the black-and-white image plane.
- Image is stretched with the stretch ratio defined by the **StretchRatio** property.

All regions that are added inside this object should not exceed the bounds of the image rectangle.

The **ImageModification** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
StretchRatio	**Double**	Specifies the stretch ratio to apply to the image. By default this property is 1.0 that corresponds to no stretch.

Methods

Name	Description
AddClipRegion	Adds new clipping region to the internal array of clipping regions.
AddInvertRegion	Adds new inversion region to the internal array of inversion regions.
AddPaintRegion	Adds new paint region to the internal array of paint regions.
AddRemoveGarbageRegion	Adds new "remove garbage" region to the internal array of "remove garbage" regions.
AddReplaceBlackPixelsRegion	Adds new "replace black pixels" region to the internal array of "replace black pixels" regions.
AddReplaceWhitePixelsRegion	Adds new "replace white pixels" region to the internal array of "replace white pixels" regions.
ClearClipRegions	Clears the internal array of clipping regions.
ClearInvertRegions	Clears the internal array of inversion regions.
ClearPaintRegions	Clears the internal array of paint regions.
ClearRemoveGarbageRegions	Clears the internal array of "remove garbage" regions.
ClearReplaceBlackPixelsRegions	Clears the internal array of "replace black pixels" regions.
ClearReplaceWhitePixelsRegions	Clears the internal array of "replace white pixels" regions.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the **CreateImageModification** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **EstimateBitmapSize**, **GetPicture**, **WriteToFile** of the **Image** object,
- **Modify** of the **ImageDocument** object.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
       Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
       prepareImageMode, 0, 0 );

// Extract image dimensions
```

```
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Create and initialize the ImageModification object
FREngine::IImageModificationPtr imageModification =
       Engine->CreateImageModification();
// Set clipping region (1/12 of image width from left and right
// and 1/6 of image height from top and bottom)
FREngine::IRegionPtr region = Engine->CreateRegion();
region->AddRect( width / 12, height / 6, 11 * width / 12, 5 * height / 6 );
FREngine::IRegionPtr region = Engine->CreateRegion
imageModification->AddClipRegion( region );
// Save modified image
image->WriteToFile( L"D:\\sample.png",
       FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
       "D:\Demo.tif", PrepareImageMode)

Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Create and initialize the ImageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
' Set clipping region (1/12 of image width from left and right
' and 1/6 of image height from top and bottom)
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect Width / 12, Height / 6, 11 * Width / 12, 5 * Height / 6
' Clip margins (1/12 of image width from left and right
' and 1/6 of image height from top and bottom)
ImageModification.AddClipRegion Region
' Save modified image
Image.WriteToFile "D:\sample.png", _
       IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

Working with Images
Working with Properties

AddClipRegion Method of the ImageModification Object

This method adds a new clipping region to the internal array of clipping regions of the **ImageModification** object. To remove all the clipping regions previously added call the **IImageModification::ClearClipRegions** method. In case the modification is applied to a single color plane of the image, coordinates of the region should be specified on this color plane. In case the modification is applied to the whole **ImageModification**, the coordinates should be specified on the deskewed black-and-white image plane.

Visual Basic Syntax

```
Method AddClipRegion(
  region As Region
)
```

C++ Syntax

```
HRESULT AddClipRegion(
  IRegion* region
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the clipping region to be added.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Clipping regions specify what part of the image will be affected during modification. Only part of image inside the bounding region of the clipping regions is processed, and the part of image inside the bounding region that does not belong to any of the clipping regions, is filled in with the white color.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
       Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
       prepareImageMode, 0, 0 );

// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Create and initialize the ImageModification object
FREngine::IImageModificationPtr imageModification =
       Engine->CreateImageModification();
// Set clip region (1/12 of image width from left and right
// and 1/6 of image height from top and bottom)
FREngine::IRegionPtr region = Engine->CreateRegion();
region->AddRect( width / 12, height / 6, 11 * width / 12, 5 * height / 6 );
// Clip margins
imageModification->AddClipRegion( region );
// Save modified image
image->WriteToFile( L"D:\\sample.png",
       FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
       "D:\Demo.tif", PrepareImageMode)

Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Create and initialize the ImageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
' Set clip region (1/12 of image width from left and right
' and 1/6 of image height from top and bottom)
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect Width / 12, Height / 6, 11 * Width / 12, 5 * Height / 6
' Clip margins
ImageModification.AddClipRegion Region
' Save modified image
Image.WriteToFile "D:\sample.png", _
       IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

ImageModification
IImageModification::ClearClipRegions

AddInvertRegion Method of the ImageModification Object

This method adds a new inversion region to the internal array of inversion regions of the **ImageModification** object. To remove all the inversion regions previously added call the **IImageModification::ClearInvertRegions** method**.** In case the modification is applied to a single color plane of the image, coordinates of the region should be specified on this color plane. In case the modification is applied to the whole **ImageModification**, the coordinates should be specified on the deskewed black-and-white image plane.

Visual Basic Syntax

```
Method AddInvertRegion(
  region As Region
)
```

C++ Syntax

```
HRESULT AddInvertRegion(
  IRegion* region
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the inversion region to be added.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Colors of the image inside the inversion regions will be inverted when **IImageDocument::Modify** method is applied.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
```

```
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
       Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
       prepareImageMode, 0, 0 );

// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Create and initialize the ImageModification object
FREngine::IImageModificationPtr imageModification =
       Engine->CreateImageModification();
// Set invert region
FREngine::IRegionPtr region = Engine->CreateRegion();
region->AddRect( width / 3, height / 3, 2 * width / 3, 2 * height / 3 );
imageModification->AddInvertRegion( region );
// Save modified image
image->WriteToFile( L"D:\\sample.png",
       FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
       "D:\Demo.tif", PrepareImageMode)

Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Create and initialize the ImageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
' Set invert region
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect Width / 3, Height / 3, 2 * Width / 3, 2 * Height / 3
ImageModification.AddInvertRegion Region
' Save modified image
Image.WriteToFile "D:\sample.png", _
       IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

ImageModification
IImageModification::ClearInvertRegions

AddPaintRegion Method of the ImageModification Object

This method adds a new paint region to the internal array of paint regions of the **ImageModification** object. To remove all the paint regions previously added call the **IImageModification::ClearPaintRegions** method**.** In case the modification is applied to a single color plane of the image, coordinates of the region should be specified on this color plane. In case the modification is applied to the whole **ImageModification**, the coordinates should be specified on the deskewed black-and-white image plane.

Visual Basic Syntax

```
Method AddPaintRegion(
  region As Region,
  color  As Long
)
```

C++ Syntax

```
HRESULT AddPaintRegion(
  IRegion* region,
  long     color
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the paint region to be added.

color

[in] This variable specifies the color with which the image inside the region is filled in.
Note: The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. For example, the Long value of the color white equals 16777215.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Parts of the image inside the paint regions will be filled in with the specified color when **IImageDocument::Modify** method is applied.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
       Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
       prepareImageMode, 0, 0 );

// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Create and initialize the ImageModification object
FREngine::IImageModificationPtr imageModification =
       Engine->CreateImageModification();
// Paint white box
FREngine::IRegionPtr region = Engine->CreateRegion();
region->AddRect( width / 4, height / 4, 3 * width / 4, 3 * height / 4 );
imageModification->AddPaintRegion( region, 16777215 );
// Save modified image
image->WriteToFile( L"D:\\sample.png",
       FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
       "D:\Demo.tif", PrepareImageMode)

Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Create and initialize the ImageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
' Paint white box
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect Width / 4, Height / 4, 3 * Width / 4, 3 * Height / 4
ImageModification.AddPaintRegion Region, 16777215
' Save modified image
Image.WriteToFile "D:\sample.png", _
       IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

ImageModification
IImageModification::ClearPaintRegions

AddRemoveGarbageRegion Method of the ImageModification Object

This method adds a new "remove garbage" region to the internal array of "remove garbage" regions of the **ImageModification** object. To remove all the "remove garbage" regions previously added call the **IImageModification::ClearRemoveGarbageRegions** method. The coordinates of the region should be specified on the deskewed black-and-white image plane.

Visual Basic Syntax

```
Method AddRemoveGarbageRegion(
  region       As Region
  attributes   As Long,
  [garbageSize As Long = -1]
)
```

C++ Syntax

```
HRESULT AddRemoveGarbageRegion(
  IRegion* region
  long     attributes,
  long     garbageSize = -1
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the "remove garbage" region to be added.

attributes

[in] This variable may either contain 0 or RGR_Invert. If RGR_Invert is passed, then white garbage on black background will be removed.

garbageSize

[in] This variable specifies the maximum size of black dots that are to be considered garbage, in pixels. The -1 value for this input parameter tells ABBYY FineReader Engine to automatically calculate the size of garbage.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The parts of the image inside the "remove garbage" regions will be cleaned up when **IImageDocument::Modify** method is applied.

See also

ImageModification
IImageModification::ClearRemoveGarbageRegions

AddReplaceBlackPixelsRegion Method of the ImageModification Object

This method adds a new "replace black pixels" region to the internal array of "replace black pixels" regions of the **ImageModification** object. To remove all the "replace black pixels" regions previously added call the **IImageModification::ClearReplaceBlackPixelsRegions** method**.** In case the modification is applied to a single color plane of the image, coordinates of the region should be specified on this color plane. In case the modification is applied to the whole **ImageModification** object, the coordinates should be specified on the deskewed black-and-white image plane.

Visual Basic Syntax

```
Method AddReplaceBlackPixelsRegion(
  region As Region,
  color  As Long,
  [strokesExpansion  As Long = 0]
)
```

C++ Syntax

```
HRESULT AddReplaceBlackPixelsRegion(
  IRegion* region
  long     color,
  long     strokesExpansion = 0
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the "replace black pixels" region to be added.

color

[in] This variable specifies the color with which the black pixels are replaced.
Note: The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. For example, the Long value of the color white equals 16777215.

strokesExpansion

[in] This variable specifies the expansion (in pixels) of white areas on the black-and-white image plane before replacing.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Black pixels on the image inside the "replace black pixels" regions will be replaced with the pixels of the specified color when **IImageDocument::Modify** method is applied. This operation is performed on the black-and-white image plane rather than color image plane but results will be applied to the color image plane too.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
```

```
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
       Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
       prepareImageMode, 0, 0 );

// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Create and initialize the ImageModification object
FREngine::IImageModificationPtr imageModification =
       Engine->CreateImageModification();
// Replace black pixels
FREngine::IRegionPtr region = Engine->CreateRegion();
region->AddRect( width / 3, height / 3, 2 * width / 3, 2 * height / 3 );
imageModification->AddReplaceBlackPixelsRegion( region, 16777215, 0 );
// Save modified image
image->WriteToFile( L"D:\\sample.png",
       FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
       "D:\Demo.tif", PrepareImageMode)

Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Create and initialize the ImageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
' Replace black pixels
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect Width / 3, Height / 3, 2 * Width / 3, 2 * Height / 3
ImageModification.AddReplaceBlackPixelsRegion Region, 16777215
' Save modified image
Image.WriteToFile "D:\sample.png", _
       IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

ImageModification
IImageModification::ClearReplaceBlackPixelsRegions

AddReplaceWhitePixelsRegion Method of the ImageModification Object

This method adds a new "replace white pixels" region to the internal array of "replace white pixels" regions of the **ImageModification** object. To remove all the "replace white pixels" regions previously added call the **IImageModification::ClearReplaceWhitePixelsRegions** method**.** In case the modification is applied to a single color plane of the image, coordinates of the region should be specified on this color plane. In case the modification is applied to the whole **ImageModification**, the coordinates should be specified on the deskewed black-and-white image plane.

Visual Basic Syntax

```
Method AddReplaceWhitePixelsRegion(
  region As Region,
  color  As Long,
  [strokesExpansion  As Long = 0]
)
```

C++ Syntax

```
HRESULT AddReplaceWhitePixelsRegion(
  IRegion* region,
  long     color,
  long     strokesExpansion
);
```

Parameters

region

[in] This parameter of the **Region** type specifies the "replace white pixels" region to be added.

color

[in] This variable specifies the color with which the white pixels are replaced.
Note: The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. For example, the Long value of the color white equals 16777215.

strokesExpansion

[in] This variable specifies the expansion (in pixels) of black areas on the image before replacing.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

White pixels on the image inside the "replace white pixels" regions will be replaced with the pixels of the specified color when **IImageDocument::Modify** method is applied. This operation is performed on the black-and-white image plane rather than color image plane but results will be applied to the color image plane too.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Open image file
FREngine::IImageDocumentPtr pImageDoc =
       Engine->PrepareAndOpenImage( L"D:\\Demo.tif",
       prepareImageMode, 0, 0 );

// Extract image dimensions
FREngine::IImagePtr image = pImageDoc->ColorImage;
long width = image->Width;
long height = image->Height;
// Create and initialize the ImageModification object
FREngine::IImageModificationPtr imageModification =
      Engine->CreateImageModification();
// Replace white pixels
FREngine::IRegionPtr region = Engine->CreateRegion();
region->AddRect( width / 3, height / 3, 2 * width / 3, 2 * height / 3 );
imageModification->AddReplaceWhitePixelsRegion( region, 0, 0 );
// Save modified image
image->WriteToFile( L"D:\\sample.png",
      FREngine::IFF_PngColorPng, imageModification, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
...
' Open image file
Dim ImageDoc As FREngine.ImageDocument
Set ImageDoc = Engine.PrepareAndOpenImage( _
      "D:\Demo.tif", PrepareImageMode)

Dim Image As FREngine.Image
Set Image = ImageDoc.ColorImage
Dim Width As Long, Height As Long
Width = Image.Width
Height = Image.Height
' Create and initialize the ImageModification object
Dim ImageModification As FREngine.ImageModification
Set ImageModification = Engine.CreateImageModification
' Replace white pixels
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect Width / 3, Height / 3, 2 * Width / 3, 2 * Height / 3
ImageModification.AddReplaceWhitePixelsRegion Region, 0
' Save modified image
Image.WriteToFile "D:\sample.png", _
      IFF_PngColorPng, ImageModification
Set Image = Nothing
```

See also

ImageModification
IImageModification::ClearReplaceWhitePixelsRegions

ClearClipRegions Method of the ImageModification Object

This method clears the internal array of clipping regions. By default the internal array of clipping regions is empty. This method may be called to remove all the clipping regions that were added previously. To add a new region to this internal array call the **IImageModification::AddClipRegion** method.

Visual Basic Syntax

```
Method ClearClipRegions()
```

C++ Syntax

```
HRESULT ClearClipRegions();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Clipping regions define what part of the image will be affected during modification. Only part of image inside the bounding region of the clipping regions is processed, and the part of image inside the bounding region that does not belong to any of the clipping regions, is filled in with the white color.

See also

ImageModification
IImageModification::AddClipRegion

ClearInvertRegions Method of the ImageModification Object

This method clears the internal array of inversion regions. By default the internal array of inversion regions is empty, and this method may be called to remove all the inversion regions that were added previously. To add a new region to this internal array call the **IImageModification::AddInvertRegion** method.

Visual Basic Syntax

```
Method ClearInvertRegions()
```

C++ Syntax

```
HRESULT ClearInvertRegions();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Colors of the image inside the inversion regions will be inverted when **IImageDocument::Modify** method is applied.

See also

ImageModification
IImageModification::AddInvertRegion

ClearPaintRegions Method of the ImageModification Object

This method clears the internal array of paint regions. By default the internal array of paint regions is empty, and this method may be called to remove all the paint regions that were added previously. To add a new region to this internal array call the **IImageModification::AddPaintRegion** method.

Visual Basic Syntax

```
Method ClearPaintRegions()
```

C++ Syntax

```
HRESULT ClearPaintRegions();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Parts of the image inside the paint regions will be filled up with the specified color when **IImageDocument::Modify** method is applied.

See also

ImageModification
IImageModification::AddPaintRegion

ClearRemoveGarbageRegions Method of the ImageModification Object

This method clears the internal array of "remove garbage" regions. By default the internal array of these regions is empty, and this method may be called to remove all the "remove garbage" regions that were added previously. To add a new region to this internal array call the **IImageModification::AddRemoveGarbageRegion** method.

Visual Basic Syntax

```
Method ClearRemoveGarbageRegions()
```

C++ Syntax

```
HRESULT ClearRemoveGarbageRegions();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The parts of the image inside the "remove garbage" regions will be cleaned up when **IImageDocument::Modify** method is applied.

See also

ImageModification
IImageModification::AddRemoveGarbageRegion

ClearReplaceBlackPixelsRegions Method of the ImageModification Object

This method clears the internal array of "replace black pixels" regions. By default the internal array of these regions is empty, and this method may be called to remove all the "replace black pixels" regions that were added previously. To add a new region to this internal array, call the **IImageModification::AddReplaceBlackPixelsRegion** method.

Visual Basic Syntax

```
Method ClearReplaceBlackPixelsRegions()
```

C++ Syntax

```
HRESULT ClearReplaceBlackPixelsRegions();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Black pixels on the image inside the "replace black pixels" regions will be replaced with the pixels of the specified color when **IImageDocument::Modify** method is applied. This operation is performed on the black-and-white image plane rather than color image plane but results will be applied to the color image plane too.

See also

ImageModification
IImageModification::AddReplaceBlackPixelsRegion

ClearReplaceWhitePixelsRegions Method of the ImageModification Object

This method clears the internal array of "replace white pixels" regions. By default the internal array of these regions is empty, and this method may be called to remove all the "replace white pixels" regions that were added previously. To add a new region to this internal array call the **IImageModification::AddReplaceWhitePixelsRegion** method.

Visual Basic Syntax

```
Method ClearReplaceWhitePixelsRegions()
```

C++ Syntax

```
HRESULT ClearReplaceWhitePixelsRegions();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

White pixels on the image inside the "replace white pixels" regions will be replaced with the pixels of the specified color when **IImageDocument::Modify** method is applied. This operation is performed on the black-and-white image plane rather than color image plane but results will be applied to the color image plane too.

See also

ImageModification
IImageModification::AddReplaceWhitePixelsRegion

MultipageImageWriter Object (IMultipageImageWriter Interface)

This object is used for saving several images into a single image file.

To write a multipage image file:

1. Create a **MultipageImageWriter** object using the **CreateMultipageImageWriter** method of the **Engine** object.
2. Add images to the end of the multipage image file using the **AddPage** method of the **MultipageImageWriter** object. Each image is added as a single page.
3. Before the newly created image file can be used, all the references to the **MultipageImageWriter** object must be released.

Note: A **MultipageImageWriter** object can be created for one-page formats, but in this case only one page can be added to the file.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.

Methods

Name	**Description**
AddPage	Appends an image to the end of the multipage image file.

Output parameter

This object is the output parameter of the **CreateMultipageImageWriter** method of the **Engine** object.

See also

Working with Images

AddPage Method of the MultipageImageWriter Object

This method appends an image to the end of the multipage image file.

Visual Basic Syntax

```
Method AddPage(
  image As Image
)
```

C++ Syntax

```
HRESULT AddPage(
  IImage* image
);
```

Parameters

image

[in] This variable refers to the **Image** object corresponding to the image to be appended.

Return Values

This method returns E_INVALIDARG if the saving format is black-and-white while the format of the image being appended is gray or color. It may also return standard return values of ABBYY FineReader Engine functions.

Remark

If you create the **MultipageImageWriter** object for an one-page format, you can add no more than one page to the resulting file.

See also

MultipageImageWriter
IEngine::CreateMultipageImageWriter

IImagePasswordCallback Interface

This interface is to be implemented on the client side. It contains a method which can return a password when it is needed to access the image file. Currently, only files in PDF format can be protected with passwords.

The sequence of usage for this interface is as follows:

1. The user of ABBYY FineReader Engine implements an object with the **IImagePasswordCallback** interface. For C++, this object should be derived from this interface and implement its **raw_GetPassword** method. This object should also implement the methods of the **IUnknown** interface.

2. The user then passes a pointer to this object's interface into any of the **IFRDocument::AddImageFileWithPasswordCallback**, **IEngine::GetNumberOfPagesInImageFile**, **IEngine::PrepareImage**, **IEngine::PrepareAndOpenImage**, **IEngine::RecognizeImageFile**, **IEngine::RecognizeImageAsPlainText** methods as one of input parameters. ABBYY FineReader Engine will call the **GetPassword** method of this object to get the password if necessary.

In the case when the user does not expect to deal with password-protected image files or does not want to handle password requests, the "Null" (C++) or "Nothing" (Visual Basic) pointer may be passed instead of the pointer to **IImagePasswordCallback** interface. The only disadvantage of this approach is that password-protected image files will not be opened with ABBYY FineReader Engine.

Method

Name	Description
GetPassword	Returns the password.

Input parameter

This object is the input parameter of the following methods:

- **PrepareImage**, **PrepareAndOpenImage**, **RecognizeImageFile**, **RecognizeImageAsPlainText**, **GetNumberOfPagesInImageFile** methods of the **Engine** object.

- **AddImageFileWithPasswordCallback** method of the **FRDocument** object.

See also

Working with Connectable Objects

GetPassword Method of the IImagePasswordCallback Interface

This method is implemented by the user. ABBYY FineReader Engine can use a pointer to the **IImagePasswordCallback** interface in methods that open image files to request passwords for protected files, actually PDFs. Typical implementation of this method could show a dialog box where the user can provide a password necessary to access the image file.

Visual Basic Syntax

```
Sub IImagePasswordCallback_GetPassword(
  ByVal ownerNeeded   As Boolean,
  ByRef isResultValid As Boolean
) As String
```

C++ Syntax

```
HRESULT raw_GetPassword(
  VARIANT_BOOL  ownerNeeded,
  VARIANT_BOOL* isResultValid,
  BSTR*         password
);
```

Parameters

ownerNeeded

[in] This parameters indicates whether "user" (*ownerNeeded*=**False**) or "owner" (*ownerNeeded*=**True**) password is requested for the image file in PDF format. "Owner" password provides highest access level to the document's contents and properties.

isResultValid

[out] This parameter should be set to TRUE if result value in the *password* should be used by ABBYY FineReader Engine. When it is set to FALSE, ABBYY FineReader Engine will act as if no password available and will not open the image file. *The default value of this parameter is FALSE.*

password

[out] This parameter allows you to return the string to be used as a password for the image file.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

This method may be called by ABBYY FineReader Engine possibly more than once, until the correct password is returned or *isResultValid* parameter is set to FALSE, which means "the user cannot (or does not want to) enter the password".

See also

IImagePasswordCallback

TrainingImagesCollection Object (ITrainingImagesCollection Interface)

This object represents a collection of **TrainingImage** objects. It serves as a storage to pass various sets of parameters into those ABBYY FineReader Engine functions that require them. It may also be return value of ABBYY FineReader Engine methods.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

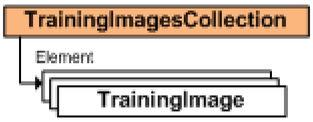
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**TrainingImage**	Provides access to a single element of the collection.

Methods

Name	**Description**
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



Output parameter

This collection is the output parameter of the **CreateTrainingImagesCollection** method of the **Engine** object.

Input parameter

This collection is the input parameter of the **TrainUserPattern** method of the **Engine** object.

See also

Recognizing with Training
Training User Patterns
Working with Properties

TrainingImage Object (ITrainingImage Interface)

This object represents a single training image. It contains character image which can be used during user pattern training.

⚠**Important!**

- You must specify the values of the **Height** and **Width** properties before you call the **SetImageData** method.

- The **SmallCharsHeight** property must be set to the correct value.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BaseLine	**Long**	Contains the distance from the base line to the top edge of the image. The base line is the line on which the characters are located. The top edge of the image is determined by the character orientation. By default, the value of this property is 0.
Height	**Long**	Specifies the height of the training image in pixels. By default, the value of this property is 0.
SmallCharsHeight	**Long**	Specifies the height of small characters in pixels on the source image. By default, the value of this property is 0.
Width	**Long**	Specifies the width of the training image in pixels. By default, the value of this property is 0.

Methods

Name	Description
SetImageData	Sets the training image data from the buffer in memory. The image should be isotropic (that is its horizontal resolution should equal the vertical one), black-and-white with 1 bit per pixel encoding.

Related objects

TrainingImagesCollection
Element
TrainingImage

Output parameter

This object is the output parameter of the following methods:

- **CreateTrainingImage** method of the **Engine** object.
- **Item** method of the **TrainingImagesCollection** object.

Input parameter

This object is the input parameter of the **Insert**, **Add** method of the **TrainingImagesCollection** object.

See also

TrainingImagesCollection
Recognizing with Training
Training User Patterns
Working with Properties

SetImageData Method of the TrainingImage Object

This method sets the training image data from the buffer in memory.

The image should be isotropic (that is its horizontal resolution should equal the vertical one), black-and-white with 1 bit per pixel encoding.

Image is stored in buffer continuously, line-by-line, from top to bottom. One line of black-and-white image is stored as a sequence of at least *N* = ceil (**ITrainingImage::Width** / 8) bytes. Each byte encodes colors of 8 adjacent pixels, most significant bit of the first byte corresponds to leftmost pixel of the line. Bit value of 0 denotes white pixel, value if 1 denotes black pixel. If **ITrainingImage::Width** is not a multiple of 8, least significant bits of *N*-th byte are ignored.

Visual Basic Syntax

```
Method SetImageData(
  rawDataPointer As Long
) As ImageDocument
```

C++ Syntax

```
HRESULT SetImageData(
  long rawDataPointer
);
```

Parameters

rawDataPointer

[in] This parameter is treated as a pointer to memory buffer containing image data.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The image data should exist while the **TrainingImage** object received from this method exists.

See also

TrainingImage

Layout-Related Objects

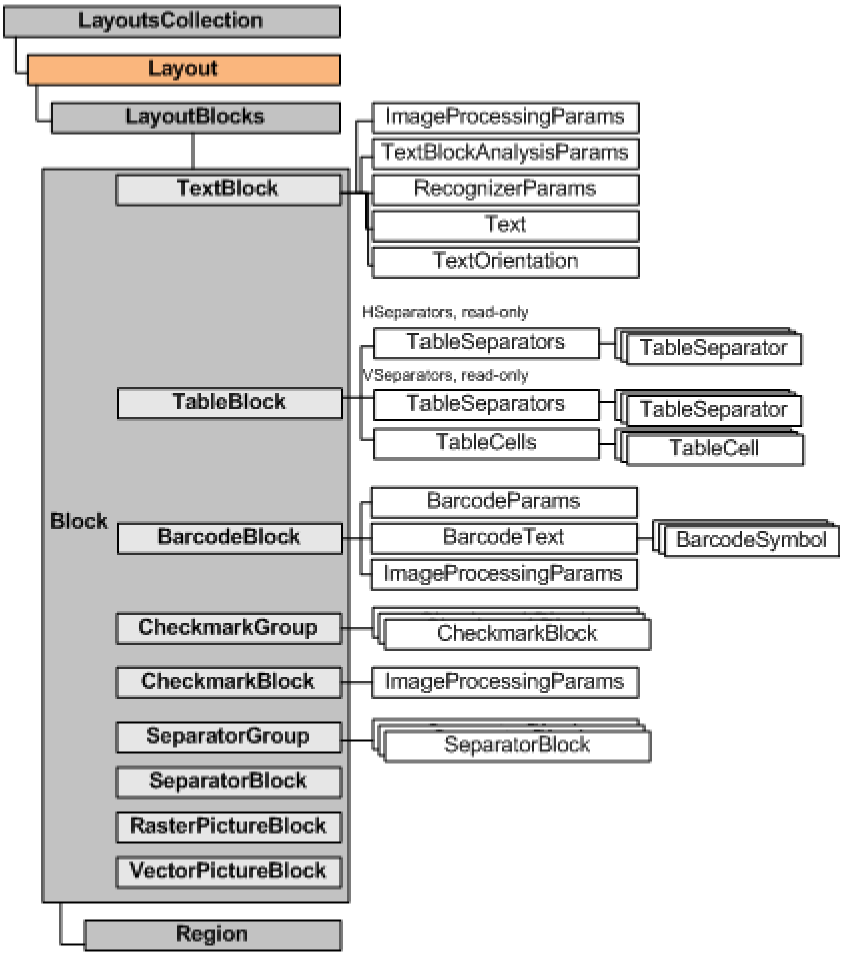
Layout object is at the top of hierarchy of objects that represent ABBYY FineReader Engine blocks. Layout object exposes a collection of blocks. A block is an object that defines a zone on image and specifies the way in which this zone is recognized. It also contains the recognized text that corresponds to the image zone the block defines.

This section contains descriptions of the following layout-related objects:

- **Layout**
- **LayoutsCollection**
- **LayoutBlocks**
- **Block**
- **TextBlock**
- **TextBlockAnalysisParams**
- **TableBlock**
- **TableCells**
- **TableCell**
- **TableSeparators**
- **TableSeparator**
- **BarcodeBlock**
- **BarcodeText**
- **BarcodeSymbol**
- **RasterPictureBlock**
- **CheckmarkGroup**
- **CheckmarkBlock**
- **SeparatorGroup**
- **SeparatorBlock**

You can find additional information about how to work with layout and blocks in the Working with Layout and Blocks section.

The layout-related objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

Layout Object (ILayout Interface)

This object exposes methods and properties for working with the image layout. The **Layout** object serves as a root for blocks. Its attributes are width and height. These parameters are set equal to the corresponding parameters of the black-and-white page of the image for which the **Layout** object is defined. This is done automatically when the **Layout** object is analyzed or recognized. It is not recommended to change the geometrical parameters of the **Layout** object, as it may unpredictably affect the results of the recognized text export.

The **Layout** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

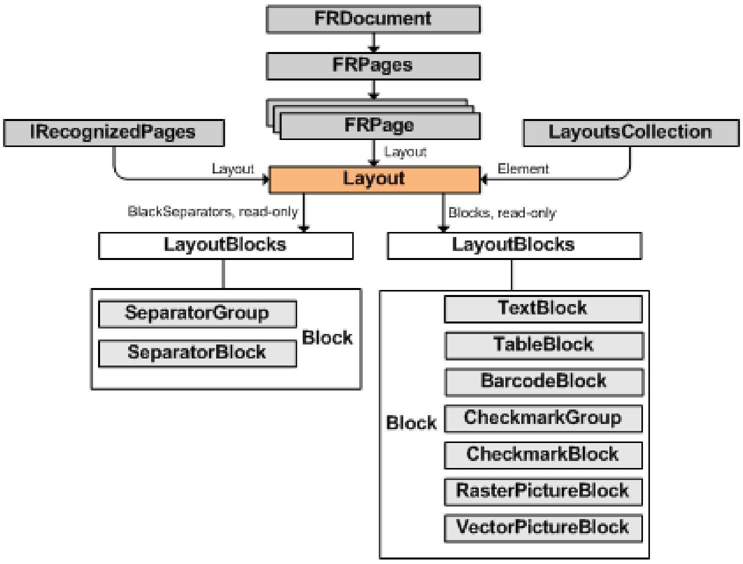
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BlackSeparators	LayoutBlocks, read-only	Provides access to the collection of separator and separator group blocks of the layout. This property refers to a valid object independently of whether there are any separator blocks in the **Layout** or not. In case there are no separators in the **Layout**, the **BlackSeparators** property is empty.
Blocks	LayoutBlocks, read-only	Provides access to the collection of blocks of the layout. This collection does not contain the separator and separator group blocks. To access these blocks, use the **BlackSeparator** property. This property refers to a valid object independently of whether there are any blocks in

		Layout or not. In case there are no blocks in **Layout**, the **Blocks** property is empty. See also **Working with read-only object properties in raw C++.**
Height	**Long**	Stores the layout height in pixels. Usually the height of **Layout** is the same as the height of the image to which this **Layout** corresponds, although you may assign it any positive value. Blocks may exceed the bounds of the layout, but they will be trimmed during recognition.
Name	**String**	Stores the layout name.
TextAsString	**String**, read-only	Writes the values of all blocks, except for Picture blocks, to one line. The **Cells** object determines the order in which text from table cells is written. So the order may not coincide with the table cells order as they go in the image.
UserProperty	**VARIANT**	Allows you to associate any user-defined information with an object of the **Layout** type.
Width	**Long**	Stores the layout width in pixels. Usually the width of **Layout** is the same as the width of the image to which this **Layout** corresponds, although you may assign it any positive value. Blocks may exceed the bounds of the layout, but they will be trimmed during recognition.

Methods

Name	**Description**
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
AddBlock	Creates the **Block** object of the type specified and adds it to the collection of the layout blocks.
InsertBlock	Creates the **Block** object of the type specified and inserts it into the specified position in the collection of the layout blocks.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **CreateLayout** method of the **Engine** object.
- **Item** method of the **LayoutsCollection** object.

Input parameter

This object is the input parameter of the following methods:

- **AnalyzeAndRecognizePage**, **AnalyzePage**, **ExportPage**, **RecognizePage**, **CreateLayoutBlocks** methods of the **Engine** object.
- **AnalyzeAndRecognizePage**, **AnalyzePage**, **AnalyzeTable**, **ExtractBarcodes**, **RecognizeBlocks**, **RecognizePage** methods of the **DocumentAnalyzer** object.
- **Add**, **Insert** methods of the **LayoutsCollection** object.

See also

Working with Layout and Blocks
LayoutBlocks
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

AddBlock Method of the Layout Object

This method creates a **Block** object of the type specified and adds it to the collection of the layout blocks.

Visual Basic Syntax

```
Method AddBlock(
  blockType As BlockTypeEnum,
  region    As Region
) As Block
```

C++ Syntax

```
HRESULT AddBlock(
  BlockTypeEnum blockType,
  IRegion*      region,
  IBlock**      block
);
```

Parameters

blockType

[in] This variable of the **BlockTypeEnum** type specifies the type of the newly created block.

region

[in] This variable refers to the **Region** object that specifies the region of the newly created block. This parameter may be 0, in which case the region of the new block will be set to the region of the layout.

block

[out, retval] A pointer to **IBlock*** pointer variable that receives the interface pointer of the created block.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
...
// Create a Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
// Set block region
FREngine::IRegionPtr pRegion = Engine->CreateRegion();
pRegion->AddRect( 0, 0, 100, 50 );
// Create a block of the "checkmark" type and add into the layout
FREngine::IBlockPtr pCheckmark = pLayout->AddBlock( FREngine::BT_Checkmark, pRegion );
...
```

Visual Basic code

```
...
```

```
' Create a Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()
' Set block region
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect 0, 0, 100, 50
' Create a block of the "checkmark" type and add it into the layout
Dim Checkmark As FREngine.block
Set Checkmark = Layout.AddBlock(BT_Checkmark, Region)
...
```

See also

Layout
Block
ILayout::InsertBlock

InsertBlock Method of the Layout Object

This method creates a **Block** object of the type specified and inserts it into the specified position in the collection of the layout blocks.

Visual Basic Syntax

```
Method InsertBlock(
  index     As Long,
  blockType As BlockTypeEnum,
  region    As Region
) As Block
```

C++ Syntax

```
HRESULT InsertBlock(
  long          index,
  BlockTypeEnum blockType,
  IRegion*      region,
  IBlock**      block
);
```

Parameters

index

[in] This parameter specifies the index of the newly inserted block in the collection of the layout blocks. The value of the parameter must be in range from 0 to the value of the **ILayoutBlocks::Count** property. If the block with this index already exists in the group, the elements of the collection are shifted to the right. The element may also be inserted at the end of collection, in which case the value of this parameter must be equal to the value of the **ILayoutBlocks::Count** property.

blockType

[in] This variable of the **BlockTypeEnum** type specifies the type of the newly created block.

region

[in] This variable refers to the **Region** object that specifies the region of the newly created block. This parameter may be 0, in which case the region of the new block will be set to the region of the layout.

block

[out, retval] A pointer to **IBlock*** pointer variable that receives the interface pointer of the created block.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Layout
Block

LayoutBlocks
ILayout::AddBlock

LayoutsCollection Object (ILayoutsCollection Interface)

This object represents a collection of **Layout** objects. It serves as a storage to pass various sets of parameters into those ABBYY FineReader Engine functions that require them. It may also be return value of ABBYY FineReader Engine methods.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

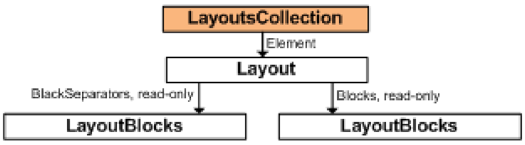
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**Layout**	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



Output parameter

This collection is the output parameter of the **CreateLayoutsCollection** method of the **Engine** object.

Input parameter

This collection is the input parameter of the following methods:

- **AnalyzeAndRecognizePages**, **AnalyzePages**, **ExportPages**, **RecognizePages**, **SynthesizePages** of the **Engine** object.
- **AnalyzeAndRecognizePages**, **AnalyzePages**, **RecognizePages** methods of the **DocumentAnalyzer** object.
- **ExportPages** method of the **Exporter** object.

See also

Working with Layout and Blocks
Layout
Working with Properties

LayoutBlocks Object (ILayoutBlocks Interface)

This object represents a collection of layout blocks. It may exist either as independent object or as a sub-object of a **Layout** object, and it serves as a mean to pass a collection of blocks to a method or as a storage of **Layout** blocks respectively.

In the first case, the collection is empty after creating with the **IEngine::CreateLayoutBlocks** method and blocks can be added to this collection using the **Add** and **Insert** methods, and deleted using the **Remove**, **RemoveAll** methods.

In the second case, the blocks collection is received using the **ILayout::Blocks** property and already contains all the blocks of the layout. You cannot call the **Add** and **Insert** methods for such object. To add or insert a block into the collection, use the **AddBlock** or **InsertBlock** methods of the corresponding **Layout** object, which creates a new block and adds it into the layout. The **Remove** and **RemoveAll** methods delete blocks from the collection and from the layout.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

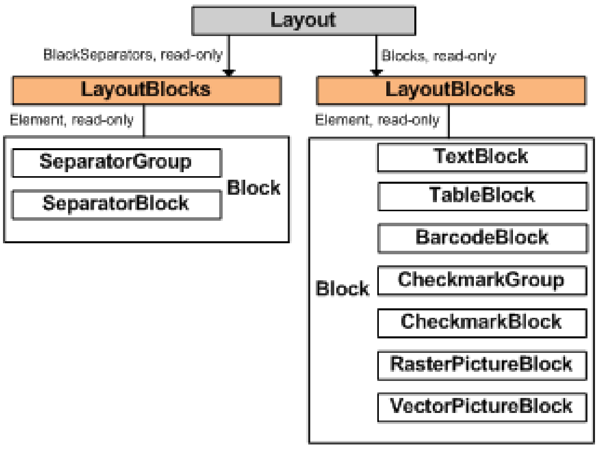
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**Block**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a **Block** object at the end of the collection. The method is not available for the collection received using the **ILayout::Blocks** property.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a **Block** object into the specified position in the collection. The method is not available for the collection received using the **ILayout::Blocks** property.
Item	Provides access to a **Block** object in a collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



Output parameter

This object is the output parameter of the **CreateLayoutBlocks** method of the **Engine** object.

See also

Block
Layout
Working with Layout and Blocks
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

Block Object (IBlock Interface)

This object represents a single block. When recognizing a page, ABBYY FineReader Engine first analyzes its layout and detects blocks of various types on the page. Blocks determine how the image areas are recognized and contain recognized information after recognition.

Each block on the page belongs to one of the nine types: text, table, raster picture, vector picture, barcode, checkmark, checkmarks group, separator, and separators group. The type of the block is defined by the **Type** property. The **Block** object exposes methods which typecast it to the one of its child objects and thereby provide access to the extended attributes of a block of specific type.

The position of the block on an image is defined by its region (the **Region** property) and the layer to which the block belongs (the **BlockLayerType** property).

Properties

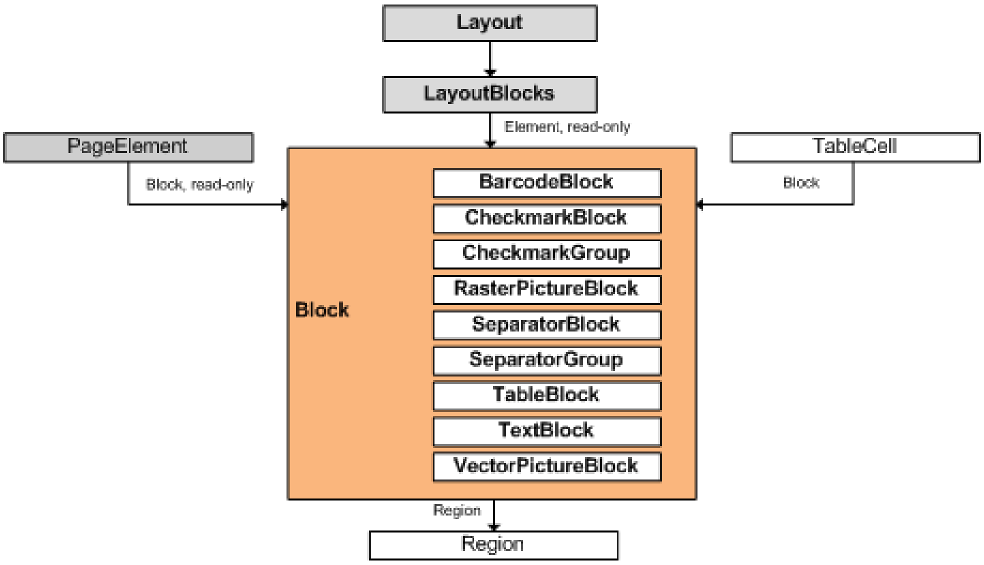
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColor	**Long**	Specifies the background color of the block. By default, the value of this property is white, or RGB(255,255,255).
BlockLayerType	**BlockLayerTypeEnum**	Specifies the layer of the block: background, foreground, or hidden. Blocks may be overlaid, for example, a text block may lay over a background picture block. By default, the value of this property is BLT_Foreground.
Description	**String**	Stores the description of the block. By default, the value of this property is an empty string.
Name	**String**	Stores the name of the block. It may be an arbitrary string. By default, the value of this property is an empty string.
Region	**Region**	Provides access to the block region. A region is a collection of rectangles placed one under another and having common top/bottom coordinates. That is, the bottom line of the upper rectangle touches the top line of the lower one. Unlike other types of blocks, a table block may have no more than one rectangle in its region, that is why an attempt to assign a region with more that one rectangle to a table block will result in an error. The region is defined by coordinates of its rectangles (in pixels) upon the deskewed black-and-white plane of the corresponding image. **Note:** The property returns a constant object. To change the block region, you must first receive an intermediate **Region** object with the help of the **IEngine::CreateRegion** method, change the necessary parameters, and then assign this object to the property.
Type	BlockTypeEnum, read-only	ABBYY FineReader Engine uses the following nine types of blocks: text, table, raster picture, vector picture, barcode, checkmark, checkmarks group, separator, and separators group. Each type of block has its own specific properties. Block type is defined at the time of its creation. It can only be changed by the following procedure: 1. Delete this block from layout by calling the **ILayoutBlocks::Remove** method. 2. Create a new block of the desired type and add it into the desired layout by calling the **AddBlock** or **InsertBlock** method of the **Layout** object.
UserProperty	**VARIANT**	Allows you to associate some user-defined information of any type with an object of the **Block** type.

Methods

Name	Description
GetAsBarcodeBlock	Returns the block as the **BarcodeBlock** object. If the block is not a barcode block, NULL is returned.
GetAsCheckmarkBlock	Returns the block as the **CheckmarkBlock** object. If the block is not a checkmark block, NULL is returned.
GetAsCheckmarkGroup	Returns the block as the **CheckmarkGroup** object. If the block is not a checkmark group block, NULL is returned.
GetAsRasterPictureBlock	Returns the block as the **RasterPictureBlock** object. If the block is not a raster picture block, NULL is returned.
GetAsSeparatorBlock	Returns the block as the **SeparatorBlock** object. If the block is not a separator block, NULL is returned.

GetAsSeparatorGroup	Returns the block as the **SeparatorGroup** object. If the block is not a separator group block, NULL is returned.
GetAsTableBlock	Returns the block as the **TableBlock** object. If the block is not a table block, NULL is returned.
GetAsTextBlock	Returns the block as the **TextBlock** object. If the block is not a text block, NULL is returned.
GetAsVectorPictureBlock	Returns the block as the **VectorPictureBlock** object. If the block is not a vector picture block, NULL is returned.
Move	Offsets block region by some vector.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **AddBlock** and **InsertBlock** methods of the **Layout** object
- **Item** method of the **LayoutBlocks** object

Input parameter

This object is the input parameter of the **Insert**, **Add** methods of the **LayoutBlocks** object.

See also

LayoutBlocks
Working with Layout and Blocks
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

GetAsBarcodeBlock Method of the Block Object

This method returns the block as the **BarcodeBlock** object. If the block is not a barcode block, NULL is returned.

Visual Basic Syntax

```
Method GetAsBarcodeBlock() As BarcodeBlock
```

C++ Syntax

```
HRESULT GetAsBarcodeBlock(
  IBarcodeBlock** result
);
```

Parameters

result

[out] A pointer to **IBarcodeBlock*** pointer variable that receives the interface pointer to the returned **BarcodeBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
BarcodeBlock

GetAsCheckmarkBlock Method of the Block Object

This method returns the block as the **CheckmarkBlock** object. If the block is not a checkmark block, NULL is returned.

Visual Basic Syntax

```
Method GetAsCheckmarkBlock() As CheckmarkBlock
```

C++ Syntax

```
HRESULT GetAsCheckmarkBlock(
  ICheckmarkBlock** result
);
```

Parameters

result

[out] A pointer to **ICheckmarkBlock*** pointer variable that receives the interface pointer to the returned **CheckmarkBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
CheckmarkBlock

GetAsCheckmarkGroup Method of the Block Object

This method returns the block as the **CheckmarkGroup** object. If the block is not a checkmark group block, NULL is returned.

Visual Basic Syntax

```
Method GetAsCheckmarkGroup() As CheckmarkGroup
```

C++ Syntax

```
HRESULT GetAsCheckmarkGroup(
  ICheckmarkGroup** result
);
```

Parameters

result

[out] A pointer to **ICheckmarkGroup*** pointer variable that receives the interface pointer to the returned **CheckmarkGroup** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
CheckmarkGroup

GetAsRasterPictureBlock Method of the Block Object

This method returns the block as the **RasterPictureBlock** object. If the block is not a raster picture block, NULL is returned.

Visual Basic Syntax

```
Method GetAsRasterPictureBlock() As RasterPictureBlock
```

C++ Syntax

```
HRESULT GetAsRasterPictureBlock(
  IRasterPictureBlock** result
);
```

Parameters

result

[out] A pointer to **IRasterPictureBlock*** pointer variable that receives the interface pointer to the returned **RasterPictureBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
RasterPictureBlock

GetAsSeparatorBlock Method of the Block Object

This method returns the block as the **SeparatorBlock** object. If the block is not a separator block, NULL is returned.

Visual Basic Syntax

```
Method GetAsSeparatorBlock() As SeparatorBlock
```

C++ Syntax

```
HRESULT GetAsSeparatorBlock(
  ISeparatorBlock** result
);
```

Parameters

result

[out] A pointer to **ISeparatorBlock*** pointer variable that receives the interface pointer to the returned **SeparatorBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
SeparatorBlock

GetAsSeparatorGroup Method of the Block Object

This method returns the block as the **SeparatorGroup** object. If the block is not a separator group block, NULL is returned.

Visual Basic Syntax

```
Method GetAsSeparatorGroup() As SeparatorGroup
```

C++ Syntax

```
HRESULT GetAsSeparatorGroup(
  ISeparatorGroup** result
);
```

Parameters

result

[out] A pointer to **ISeparatorGroup*** pointer variable that receives the interface pointer to the returned **SeparatorGroup** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
SeparatorGroup

GetAsTableBlock Method of the Block Object

This method returns the block as the **TableBlock** object. If the block is not a table block, NULL is returned.

Visual Basic Syntax

```
Method GetAsTableBlock() As TableBlock
```

C++ Syntax

```
HRESULT GetAsTableBlock(
  ITableBlock** result
);
```

Parameters

result

[out] A pointer to **ITableBlock*** pointer variable that receives the interface pointer to the returned **TableBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
TableBlock

GetAsTextBlock Method of the Block Object

This method returns the block as the **TextBlock** object. If the block is not a text block, NULL is returned.

Visual Basic Syntax

```
Method GetAsTextBlock() As TextBlock
```

C++ Syntax

```
HRESULT GetAsTextBlock(
  ITextBlock** result
);
```

Parameters

result

[out] A pointer to **ITextBlock*** pointer variable that receives the interface pointer to the returned **TextBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
TextBlock

GetAsVectorPictureBlock Method of the Block Object

This method returns the block as the **VectorPictureBlock** object. If the block is not a vector picture block, NULL is returned.

Visual Basic Syntax

```
Method GetAsVectorPictureBlock() As VectorPictureBlock
```

C++ Syntax

```
HRESULT GetAsVectorPictureBlock(
  IVectorPictureBlock** result
);
```

Parameters

result

[out] A pointer to **IVectorPictureBlock*** pointer variable that receives the interface pointer to the returned **VectorPictureBlock** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Block
VectorPictureBlock

Move Method of the Block Object

This method allows you to move the block region. The region is defined by coordinates of its rectangles (in pixels) upon the deskewed black-and-white plane of the corresponding image.

Visual Basic Syntax

```
Method Move(
  deltaX As Long,
  deltaY As Long
)
```

C++ Syntax

```
HRESULT Move(
  long deltaX,
  long deltaY
);
```

Parameters

deltaX

[in] Horizontal offset in pixels.

deltaY

[in] Vertical offset in pixels.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Block

DeleteAllWords Method of the Dictionary Object

This method deletes all words from the dictionary.

Visual Basic Syntax

```
Method DeleteAllWords()
```

C++ Syntax

```
HRESULT DeleteAllWords();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

To delete a group of words or a single word from the dictionary, use the **IDictionary::DeleteWords** or **IDictionary::DeleteWord** method, respectively.

See also

Dictionary
IDictionary::DeleteWords
IDictionary::DeleteWord

TextBlock Object (ITextBlock Interface)

This object provides access to specific properties of a text block. These blocks correspond to an image zone recognized as formatted text. The recognized text from the part of the image this block encloses is also accessible via this object. The **ITextBlock** interface is derived from the **IBlock** interface and inherits all its properties.

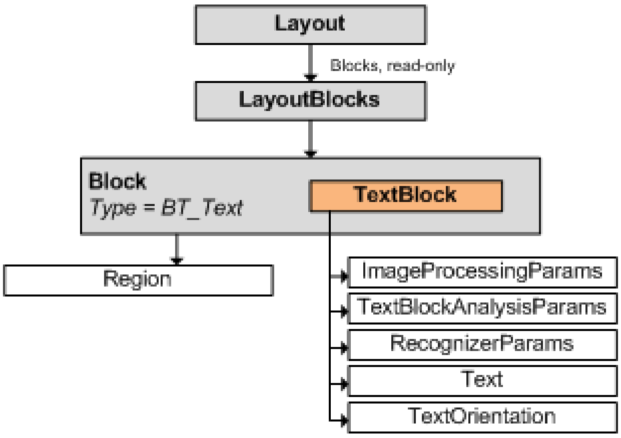
Properties

Name	Type	Description
AnalysisParams	**TextBlockAnalysisParams**	Provides access to the analysis parameters of the text block.
BlockRole	**BlockRoleEnum**	Provides access to the block role of the text block. By default, it is BR_Unknown.
ImageProcessingParams	**ImageProcessingParams**	Provides access to the image preprocessing parameters of the text block.
RecognizerParams	**RecognizerParams**	Provides access to the recognition parameters of the text block.
Text	**Text**, read-only	Provides access to the recognized text of the text block. This text always has the TR_CompoundText role (**IText::TextRole** property).
TextOrientation	**TextOrientation**	Provides access to the parameters of text orientation in the block.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the **GetAsTextBlock** method of the **Block** object.

See also

Block
Working with Layout and Blocks
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

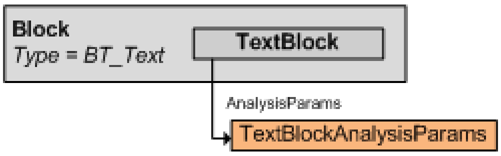
TextBlockAnalysisParams Object (ITextBlockAnalysisParams Interface)

This object specifies how a text block should be analyzed. The object allows you to set analysis parameters for each individual text block, while the **PageAnalysisParams** object affects the process of layout analysis of the whole page.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
AutodetectInversion	**Boolean**	Specifies whether the color inversion (white text on black background) must be automatically detected and normalized during layout analysis. This property is FALSE by default.
SkewCorrectionMode	**SkewCorrectionModeEnum**	Specifies the mode of skew correction during layout analysis. This property is SCM_Never by default.

Related objects



See also

Tuning Analysis, Recognition and Synthesis Parameters,
Working with Properties

TableBlock Object (ITableBlock Interface)

This object provides access to specific properties of a table block. The **ITableBlock** interface is derived from the **IBlock** interface and inherits all its properties.

The region of blocks of this type may consist of one rectangle only. The structure of the table is described by two collections of table separators, horizontal and vertical (the **HSeparators** and **VSeparators** properties), and a collection of table cells (the **Cells** property). Each table cell is treated as a block of some type.

The recognized text is a property of a single cell, not of the entire table. To access the recognized text of a table block, you should do the following:

1. Receive the collection of table cells using the **Cells** property.
2. Select the desired cell. Use the methods of the **TableCells** object.
3. Receive the block object of the cell (the **ITableCell::Block** property).
4. Check that the block is of the type BT_Text (the **IBlock::Type** property) and receive the **TextBlock** object using the **IBlock::GetAsTextBlock** method.
5. Use the **ITextBlock::Text** property.

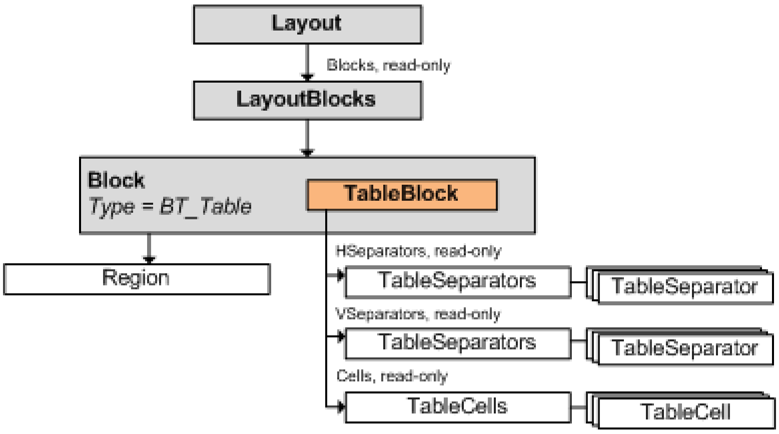
Properties

Name	Type	Description
Cells	**TableCells**, read-only	Provides access to the cells collection of the table block. The collection always contains at least one cell, in case there are no table separators in the table. The cells in the collection are arranged in the logical reading order.
HSeparators	TableSeparators, read-only	Provides access to horizontal separators collection of the table block. This collection always contains at least two separators corresponding to the table block top and bottom.
VSeparators	TableSeparators, read-only	Provides access to vertical separators collection of the table block. This collection always contains at least two separators corresponding to the table block left and right borders.

Methods

Name	Description
FindBaseCellFromPoint	Allows you to find cell position in the base grid from the pixel on image.
InitializeGrid	Initializes table grid for the table block.

Related objects



Output parameter

This object is the output parameter of the **GetAsTableBlock** method of the **Block** object.

See also

Block
Working with Layout and Blocks
Working with Text
Working with Properties

See sample: RecognizedTextProcessing

FindBaseCellFromPoint Method of the TableBlock Object

This method allows you to find cell position in the base grid for a given pixel.

Visual Basic Syntax

```
Method FindBaseCellFromPoint(
  ByVal pointX As Long,
  ByVal pointY As Long,
  ByRef baseX  As Long,
  ByRef baseY  As Long
)
```

C++ Syntax

```
HRESULT FindBaseCellFromPoint(
   long  pointX,
   long  pointY,
   long* baseX,
   long* baseY
);
```

Parameters

pointX

[in] This variable contains the horizontal coordinate of the pixel relative to the image.

pointY

[in] This variable contains the vertical coordinate of the pixel relative to the image.

baseX

[in,out] In this variable the horizontal coordinate of the cell in the base grid is returned.

baseY

[in,out] In this variable the vertical coordinate of the cell in the base grid is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Cell coordinates in a base grid are the coordinates of its left top corner in that grid. By the base grid here we assume the grid formed by table borders and separators. Each vertical separator increments the horizontal coordinate by one, and each horizontal separator increments the vertical coordinate by one. Coordinate axes are oriented from left to right and from top to bottom. Pixel coordinates relative to image must lay inside the table block's region otherwise base coordinate value returned will be -1.

See also

TableBlock

InitializeGrid Method of the TableBlock Object

This method initializes table grid for the table block.

Visual Basic Syntax

```
Method InitializeGrid(
  horzSeparators As LongsCollection,
  vertSeparators As LongsCollection
)
```

C++ Syntax

```
HRESULT InitializeGrid(
  ILongsCollection* horzSeparators,
  ILongsCollection* vertSeparators
);
```

Parameters

horzSeparators

[in] This variable refers to the **LongsCollection** object that contains coordinates of internal horizontal separators for the table block.

vertSeparators

[in] This variable refers to the **LongsCollection** object that contains coordinates of internal vertical separators for the table block.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Table separators are initialized anew by a call to this method. Old table structure is destroyed. All coordinates of table separators must lay between the coordinates of the table borders, otherwise an error code is returned. Note that only coordinates of *internal* separators should be passed in **LongsCollection** objects, that is they should not include coordinates of table borders, although table borders are always present in collections of table separators of the table block itself. All new table cells that appear as the result of a call to this method are initialized with the attributes (for example, recognition parameters) of the cell that was in a left top corner of the previous table structure.

See also

TableBlock

TableCells Object (ITableCells Interface)

All cells of a table block form a single collection represented by **TableCells** object. Besides the standard collection functionality, this object contains methods for merging and splitting groups of table cells and method for finding table cell index in collection by its position in a base table grid. The collection is accessible via the **TableBlock** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

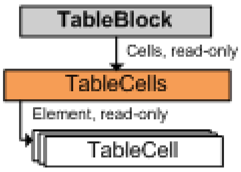
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.

Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**TableCell**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
FindCellIndex	Returns an index of the cell that corresponds to the specified point in base coordinates.
Item	Provides access to a single element of the collection.
Merge	Merges a group of cells inside the specified rectangle.
Split	Splits a group of cells inside the specified rectangle.

Related objects



See also

TableBlock
TableCell
Working with Layout and Blocks
Working with Text
Working with Properties

See sample: RecognizedTextProcessing

FindCellIndex Method of the TableCells Object

This method returns an index of the cell that corresponds to the specified point in base coordinates of the table grid.

Visual Basic Syntax

```
Method FindCellIndex(
  x As Long,
  y As Long
) As Long
```

C++ Syntax

```
HRESULT FindCellIndex(
  long  x,
  long  y,
  long* index
);
```

Parameters

x

[in] This variable specifies horizontal coordinate of the point (defined on vertical separators).

y

[in] This variable specifies vertical coordinate of the point (defined on horizontal separators).

index

[out] A pointer to **long** variable that receives the return value of this method.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The point specified should not exceed the table grid otherwise an error code is returned.

See also

TableCells

Merge Method of the TableCells Object

This method merges a group of cells located in the specified rectangle. This method changes the **TableCells** object — it affects a number of cells. During the merge, recognized text in cells, if any, is also merged and assigned to the newly created cell.

Visual Basic Syntax

```
Method Merge(
  left   As Long,
  top    As Long,
  right  As Long,
  bottom As Long
)
```

C++ Syntax

```
HRESULT Merge(
  long left,
  long top,
  long right,
  long bottom
);
```

Parameters

left

[in] This variable specifies coordinate of the left border of the rectangle in base coordinates.

top

[in] This variable specifies coordinate of the top border of the rectangle in base coordinates.

right

[in] This variable specifies coordinate of the right border of the rectangle in base coordinates.

bottom

[in] This variable specifies coordinate of the bottom border of the rectangle in base coordinates.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The rectangle for merge is specified in base coordinates, not on image. It should not cut existing cells. This means that if table block already contains merged cells — cells having dimensions more than one base unit, a user should take care and not to specify the rectangle that would intersect the interior of such cells. This rectangle may only be drawn at cells borders.

See also

ITableCells::Split

Split Method of the TableCells Object

This method splits any merged cells that are located in the specified rectangle. This method changes the **TableCells** object — it affects a number of cells. After the split, recognized text from the cells is assigned to the left top cell.

Visual Basic Syntax

```
Method Split(
  left   As Long,
  top    As Long,
  right  As Long,
  bottom As Long
)
```

C++ Syntax

```
HRESULT Split(
  long left,
  long top,
  long right,
  long bottom
);
```

Parameters

left

[in] This variable specifies coordinate of the left border of the rectangle in base coordinates.

top

[in] This variable specifies coordinate of the top border of the rectangle in base coordinates.

right

[in] This variable specifies coordinate of the right border of the rectangle in base coordinates.

bottom

[in] This variable specifies coordinate of the bottom border of the rectangle in base coordinates.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The rectangle for split is specified in base coordinates, not on image. It should not cut existing cells. This means that if table block contains merged cells — cells having dimensions more than one base unit, a user should take care and not to specify the rectangle that would intersect the interior of such cells. This rectangle may only be drawn at cells' borders. After this method call a number of new cells is added in the collection instead of the split cells. These new cells have dimensions of one base unit.

See also

ITableCells::Merge

TableCell Object (ITableCell Interface)

This object represents a single table cell of a table block. This is an element of a **TableCells** collection. The object provides access to the name of the cell, its coordinates in a base grid, and contents of the cell.

Each table cell is represented as a separate block. To access contents of the cell you should use the **Block** property. The type of the contents (e.g. text, picture) depends on the **IBlock::Type** property. If the table cell contains text, you can access the text of the table cell and other text properties using the **IBlock::GetAsTextBlock** method.

A cell has four coordinates — the indexes of the left, right, top and bottom separators that enclose it. Cell coordinates are the coordinates in a base grid. By the base grid here we assume the grid formed by table borders and separators. Each vertical separator increments the horizontal coordinate by one, and each horizontal separator increments the vertical coordinate by one. Coordinate axes are oriented from left to right and from top to bottom.

Table cell coordinates cannot be changed directly. They are affected by **ITableCells::Merge** and **ITableCells::Split** methods. But be aware that these operations not only change attributes of a single cell, but affect the cells collection as a whole, adding or removing cells.

Properties

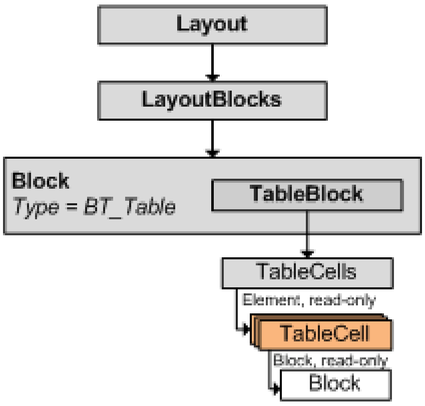
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Block	**Block**, read-only	Provides access to the block of the cell.
Bottom	**Long**, read-only	Stores coordinate of the bottom border of the cell in a base grid.

Left	**Long**, read-only	Stores coordinate of the left border of the cell in a base grid.
Name	**String**	Stores the name of the cell. The default value is an empty string.
Right	**Long**, read-only	Stores coordinate of the right border of the cell in a base grid.
Top	**Long**, read-only	Stores coordinate of the top border of the cell in a base grid.

Methods

Name	Description
ChangeBlockType	Changes the type of the block, which corresponds to the table cell.

Related objects



Output parameter

This object is the output parameter of the **Item** method of the **TableCells** object.

See also

TableBlock
TableCells
Working with Layout and Blocks
Working with Text
Working with Properties

See sample: RecognizedTextProcessing

ChangeBlockType Method of the TableCell Object

This method changes the type of the block, which corresponds to the table cell.

Visual Basic Syntax

```
Method ChangeBlockType(
  value As BlockTypeEnum
)
```

C++ Syntax

```
HRESULT ChangeBlockType(
  BlockTypeEnum  value
);
```

Parameters

value

[in] This variable specifies the new type of the block. See the description of the **BlockTypeEnum** enumeration constants. The block of the cell cannot be of the type BT_Table.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TableCell

TableSeparators Object (ITableSeparators Interface)

This object is a collection of table block separators. Each table block comprises two collections of separators: vertical and horizontal. This object contains methods for getting the number of table separators in collection and accessing a single table separator in collection. Besides, there are methods for adding/removing separators. The collection is accessible via the **TableBlock** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

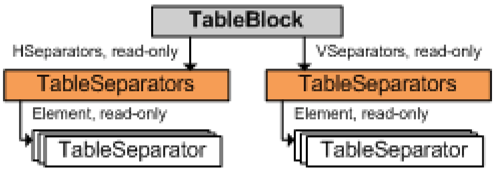
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	TableSeparator, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a new table separator into the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection. The first and the last separators cannot be removed as they correspond to the table block borders. Table cells are changed as the result of removing table separator. If several table cells containing recognized text are merged as the result of removing table separator, the new cell will contain merged text from these cells.

Related objects



See also

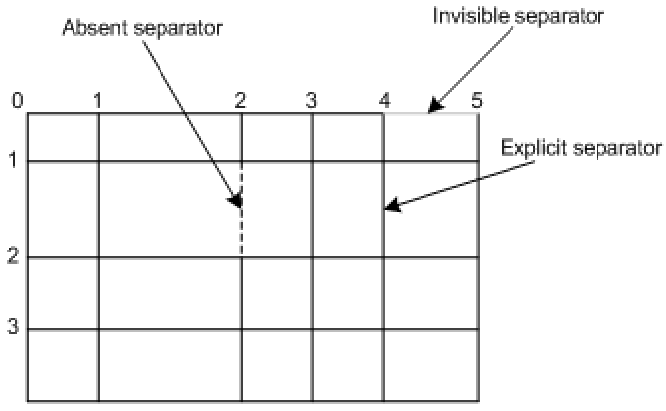
TableBlock
TableSeparator
Working with Layout and Blocks
Working with Properties

TableSeparator Object (ITableSeparator Interface)

This object represents a single table separator in a table block. It contains methods for accessing table separator attributes such as position and type.

The table separators are characterized by their types. A separator type is in fact a property of a separator part lying between its nearest crossings with other separators, and not of the entire separator. The separators may be of the following types:

- **Absent**. This type is assigned to the table separators that cross through the merged cells.
- **Unknown**. This type is assigned by default to every newly added table separator.
- **Invisible**. This type is assigned to an "imaginary" table separator created as a result of table structure analysis at a place where the source table did not have one but where it should logically be.
- **Explicit**. Table separators of this type appear at the place of black lines of the source table.
- **Multiple**. This type of separator may appear as a result of table editing.



Base coordinates in a table and types of table separators

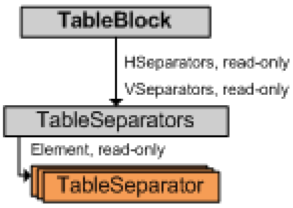
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Position	**Long**	Stores coordinate of the separator (horizontal or vertical coordinate on the image). You can set new position for the separator only between the neighboring two separators. In case new position is out of this range an error code is returned. You cannot change position of the first and the last separators in collection as they correspond to the table block's borders and their coordinates should be changed via **Block** methods.
Type	**TableSeparatorTypeEnum**	Stores separator type.

Methods

Name	Description
SetType	Sets new type for the separator.

Related objects



Output parameter

This object is the output parameter of the **Item** method of the **TableSeparators** object.

See also

TableBlock
TableSeparators
Working with Layout and Blocks
Working with Properties

Type Property of the TableSeparator Object

This property stores separator type. Separators may be of four types as defined by **TableSeparatorTypeEnum** constants. This property can only be changed by calling the **ITableSeparator::SetType** method.

Visual Basic Syntax

```
Property Type(coord As Long) As TableSeparatorTypeEnum
  read-only
```

C++ Syntax

```
HRESULT get_Type(
   long                    coord,
   TableSeparatorTypeEnum* pVal
);
```

Parameters

coord

[in] A variable of the **long** type that contains coordinate of the beginning of the separator segment in a base grid.

pVal

[out] A pointer to **TableSeparatorTypeEnum** variable that receives the value of the property. Must not be NULL.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Separator type is not an attribute of the entire separator but of a single separator segment between the adjacent intersections with perpendicular separators. Therefore this property is indexed by the *coord* parameter. Separator type is an attribute of its segment with coordinates [*coord,coord+1*] in a base grid. Table separators' types are automatically corrected during operations with groups of table cells (merging and splitting).

See also

TableSeparator
TableSeparatorTypeEnum
ITableSeparator::SetType
Working with Properties

SetType Method of the TableSeparator Object

This method sets separator type. Separators may be of four types as defined by **TableSeparatorTypeEnum** enumeration constants.

Visual Basic Syntax

```
Method SetType(
  coord   As Long,
  newType As TableSeparatorTypeEnum,
  [count  As Long = 1]
)
```

C++ Syntax

```
HRESULT SetType(
  long                   coord,
  TableSeparatorTypeEnum newType,
  long                   count
);
```

Parameters

coord

[in] A variable that contains coordinate of the beginning of the separator segment in a base grid.

newType

[in] A variable of type **TableSeparatorTypeEnum** that contains the value for the new separator type.

count

[in] A variable that contains a number of segments for which to set the new type. This is optional parameter. Default value for it is 1.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Separator type is not an attribute of the whole separator but of a single separator segment between the adjacent intersections with perpendicular separators. Therefore one of the input parameters for this method is *coord* parameter. Separator type is set for the segment with coordinates [*coord*,*coord+count*] in a base grid. It is prohibited to change the type of separator inside a merged cells (it should be **TST_Absent**), and it is prohibited to set the type of separator to the **TST_Absent** value.

See also

TableSeparatorTypeEnum
ITableSeparator::Type

BarcodeBlock Object (IBarcodeBlock Interface)

This object provides access to specific properties of the barcode block: parameters of image preprocessing and recognition in the block, type of the barcode, and recognized text of the barcode. The **IBarcodeBlock** interface is derived from the **IBlock** interface and inherits all its properties.

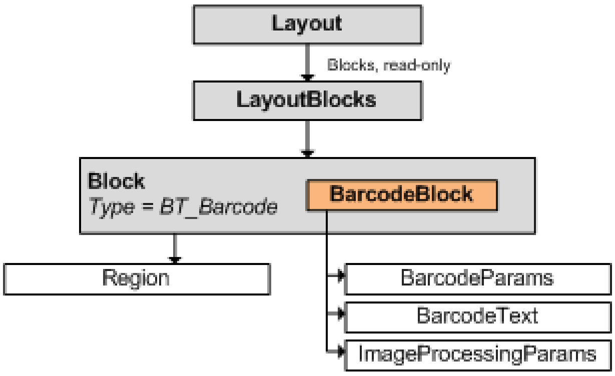
Properties

Name	Type	Description
BarcodeParams	**BarcodeParams**	Provides access to the set of properties affecting the process of barcode recognition.
BarcodeText	**BarcodeText**	Provides access to the recognized text of the barcode. The recognized text is represented as a collection of characters.
BarcodeType	**BarcodeTypeEnum**, read-only	Stores the barcode type detected during the recognition process.
ImageProcessingParams	**ImageProcessingParams**	Provides access to the set of properties affecting image preprocessing inside the barcode block.
SupplementType	**BarcodeSupplementTypeEnum**, read-only	Stores the barcode supplement type detected during the recognition process. This property is only useful for barcodes of type EAN 8, 13, UPC-A, and UPC-E.
SupplementValue	**String**, read-only	Stores the barcode supplement value detected during the recognition process. If the supplement was detected, this property contains 2 or 5 last digits of the recognized text of the barcode. The property is only useful for barcodes of type EAN 8, 13, UPC-A, and UPC-E. To change the value of this property, edit the recognized text in the **BarcodeText** property.
Text	**String**, read-only	Provides access to the recognized text of the barcode. The recognized text is represented as a string. To change the value of this property, edit the recognized text in the **BarcodeText** property.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the **GetAsBarcodeBlock** method of the **Block** object.

See also

Working with Layout and Blocks
Working with Text
Working with Properties

BarcodeText Object (IBarcodeText Interface)

This object represents a text of a recognized barcode as a collection of characters. The object exists as a sub-object of a **BarcodeBlock** object. This object exposes the standard collection functionality and allows you to create the **BarcodeSymbol** object.

Important! The indexing of ABBYY FineReader Engine collections starts with 0.

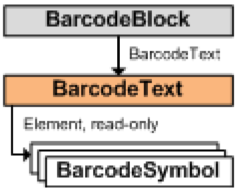
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	BarcodeSymbol, read-only	Provides access to a single element of the collection. The property returns a constant object.

Methods

Name	**Description**
Add	Adds a new element at the end of the collection.
CreateBarcodeSymbol	Creates the **BarcodeSymbol** object.
Item	Provides access to a single element of the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



See also

BarcodeSymbol
BarcodeBlock
Working with Text
Working with Properties

CreateBarcodeSymbol Method of the BarcodeText Object

This method creates the **BarcodeSymbol** object.

Visual Basic Syntax

```
Method CreateBarcodeSymbol(
) As BarcodeSymbol
```

C++ Syntax

```
HRESULT CreateBarcodeSymbol(
  IBarcodeSymbol** result
);
```

Parameters

result

[out, retval] A pointer to **IBarcodeSymbol*** pointer variable that receives the interface pointer of the created object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

BarcodeText
BarcodeSymbol

BarcodeSymbol Object (IBarcodeSymbol Interface)

This object provides access to the properties of one character of a recognized barcode: character itself, rectangle of the character, character confidence and other attributes. The object is an element of the collection of barcode characters represented by the **BarcodeText** object.

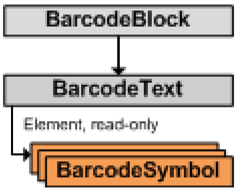
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Bottom	**Long**	Stores the coordinate of the bottom border of the character rectangle. This rectangle is defined on the deskewed black-and-white plane of the image, not accounting for the barcode orientation. It may be undefined in which case all four of its coordinates are zeros.
Character	**String**	Stores the character. The default value of this property is an empty string.
CharConfidence	**Long**	Stores the value of the character confidence. It is in the range from 0 to 100. It represents an estimate of recognition confidence of a character in percentage points. The greater its value, the greater the confidence. Character confidence can be undefined, for example, for characters which were added during barcode editing. In this case, the value of this property is -1. To calculate character confidence more accurately, set the **IRecognizerParams::ExactConfidenceCalculation** property to TRUE.
IsBinaryData	**Boolean**	Specifies whether the character represents binary data in hexadecimal mode. The default value of this property is FALSE.
IsStartStopSymbol	**Boolean**	Specifies whether the character is the barcode start/stop symbol. The property makes sense for barcodes of the Code 39 type (the start/stop symbol is the asterisk "*") and Codabar type (the start/stop symbols are Latin letters "A", "B", "C", "D"). The default value of this property is FALSE.
Left	**Long**	Stores the coordinate of the left border of the character rectangle. This rectangle is defined on the deskewed black-and-white plane of the image, not accounting for the barcode orientation. It may be undefined in which case all four of its coordinates are zeros.
Right	**Long**	Stores the coordinate of the right border of the character rectangle. This rectangle is defined on the deskewed black-and-white plane of the image, not accounting for the barcode orientation. It may be undefined in which case all four of its coordinates are zeros.
Top	**Long**	Stores the coordinate of the top border of the character rectangle. This rectangle is defined on the deskewed black-and-white plane of the image, not accounting for the barcode orientation. It may be undefined in which case all four of its coordinates are zeros.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the **Item**, **CreateBarcodeSymbol** methods the **BarcodeText** object.

Input parameter

This object is the input parameter of the **Add** method of the **BarcodeText** object.

See also

BarcodeText
Working with Text
Working with Properties

RasterPictureBlock Object (IRasterPictureBlock Interface)

This object provides access to specific properties of a raster picture block. The part of the image that this block encloses is not recognized, and the block is exported "as is". The **IRasterPictureBlock** interface is derived from the **IBlock** interface and inherits all its properties.

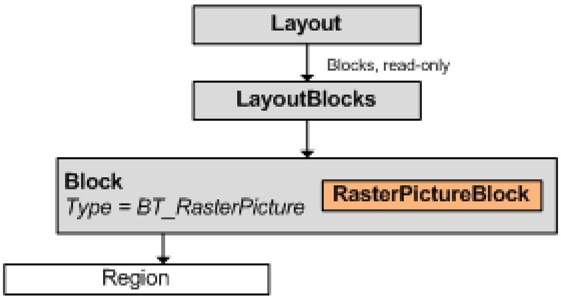
Properties

Name	Type	Description
ColorType	**ImageColorTypeEnum**	Specifies the color type for the whole image as the maximum of the corresponding values for its color planes (black-and-white, gray, color).

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the **GetAsRasterPictureBlock** method of the **Block** object.

See also

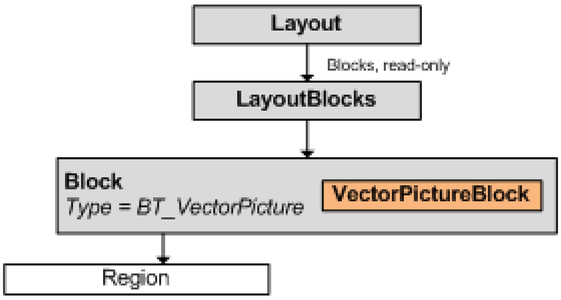
Block
Working with Layout and Blocks
Working with Properties

VectorPictureBlock Object (IVectorPictureBlock Interface)

The object represents a vector picture block. The **IVectorPictureBlock** interface is derived from the **IBlock** interface and inherits all its properties. This object does not provide any specific properties or methods for working with a vector picture block.

Blocks of this type may appear in the layout only if a page has been analyzed with the **IPageAnalysisParams::DetectVectorGraphics** property set to TRUE.

Related objects



Output parameter

This object is the output parameter of the **GetAsVectorPictureBlock** method of the **Block** object.

See also

Block
Working with Layout and Blocks

CheckmarkGroup Object (ICheckmarkGroup Interface)

This object represents a group of checkmark blocks. The **ICheckmarkGroup** interface is derived from the **IBlock** interface and inherits all its properties. Besides the standard collection functionality the **CheckmarkGroup** object allows you to set the maximum and minimum number of selected checkmarks in the group.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

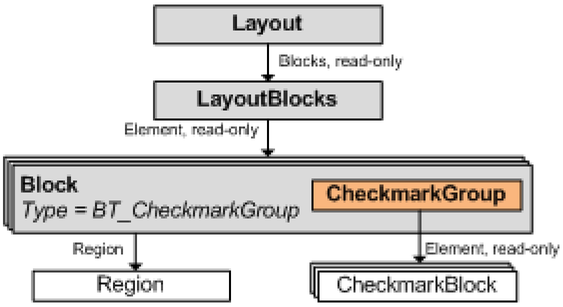
Properties

Name	Type	Description
Count	**Long**, read-only	Stores the number of elements in the group.
Element	CheckmarkBlock, read-only	Provides access to a single element of the group.
MaximumCheckedInGroup	**Long**	Specifies the maximum number of selected checkmark blocks in the group. The default value is -1, which means that all checkmark blocks in the group can be selected.
MinimumCheckedInGroup	**Long**	Specifies the minimum number of selected checkmark blocks in the group. The default value is 0.

Methods

Name	Description
AddCheckmark	Creates a checkmark block and adds it into the group.
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
InsertCheckmark	Creates a checkmark block and inserts it into the specified position in the group.
Item	Provides access to a single element of the group of checkmark blocks.
Remove	Removes an element from the group.
RemoveAll	Removes all the elements from the group.

Related objects



Output parameter

This object is the output parameter of the **GetAsCheckmarkGroup** method of the **Block** object.

See also

Block
CheckmarkBlock
Working with Layout and Blocks
Recognizing Checkmarks
Working with Properties

AddCheckmark Method of the CheckmarkGroup Object

This method creates a checkmark block and adds it into the group.

Visual Basic Syntax

```
Method AddCheckmark(
  region As Region
) As CheckmarkBlock
```

C++ Syntax

```
HRESULT AddCheckmark(
  IRegion*          region,
  ICheckmarkBlock** result
);
```

Parameters

region

[in] This variable refers to the **Region** object that specifies the region of the newly created checkmark block. This parameter may be 0, in which case the region of the new block will be set to the region of the checkmark group.

result

[out, retval] A pointer to **ICheckmarkBlock*** pointer variable that receives the interface pointer of the new checkmark block.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
...
// Create a Layout object
FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
// Set block region
FREngine::IRegionPtr pRegion = Engine->CreateRegion();
pRegion->AddRect( 0, 0, 100, 50 );

// Create a block of the "checkmark group" type and add into the layout
```

```
FREngine::IBlockPtr pCheckmarkGroup = pLayout->AddBlock( FREngine::BT_CheckmarkGroup,
pRegion );

// Create blocks of the "checkmark" type
// and add them to the checkmark group
for( int i = 0; i < 5; i++ ) {
  FREngine::IRegionPtr pCheckmarkRegion = Engine->CreateRegion();
  pRegion->AddRect( 10, 10 + i * 20, 90, 10 + (i + 1) * 20 );
  FREngine::ICheckmarkBlockPtr pCheckmark = pCheckmarkGroup->GetAsCheckmarkGroup()-
>AddCheckmark( pCheckmarkRegion );
}
...
```

Visual Basic code

```
...
' Create a Layout object
Dim Layout As FREngine.Layout
Set Layout = Engine.CreateLayout()

' Set block region
Dim Region As FREngine.Region
Set Region = Engine.CreateRegion()
Region.AddRect 0, 0, 100, 50

' Create a block of the "checkmark group" type and add it into the layout
Dim CheckmarkGroup As FREngine.block
Set CheckmarkGroup = Layout.AddBlock(BT_CheckmarkGroup, Region)

' Create blocks of the "checkmark" type
' and add them to the checkmark group
Dim i As Integer
For i = 0 To 4
Dim CheckmarkRegion As FREngine.Region
Set CheckmarkRegion = Engine.CreateRegion()
CheckmarkRegion.AddRect 10, 10 + i * 20, 90, 10 + (i + 1) * 20
Dim Checkmark As FREngine.block
Set Checkmark = CheckmarkGroup.GetAsCheckmarkGroup.AddCheckmark(CheckmarkRegion)
Next i
...
```

See also

CheckmarkBlock
CheckmarkGroup

InsertCheckmark Method of the CheckmarkGroup Object

This method creates a checkmark block and inserts it into the specified position in the group.

Visual Basic Syntax

```
Method InsertCheckmark(
  index  As Long,
  region As Region
) As CheckmarkBlock
```

C++ Syntax

```
HRESULT InsertCheckmark(
  long              index,
  IRegion*          region,
  ICheckmarkBlock** result
);
```

Parameters

index

[in] This parameter specifies the index of the newly inserted block in the checkmark group. The value of the parameter must be in range from 0 to the value of the **ICheckmarkGroup::Count** property. If the block with this index already exists in the group, the elements of the collection are shifted to the right. The element may also be inserted at the end of collection, in which case the value of this parameter must be equal to the value of the **ICheckmarkGroup::Count** property.

region

[in] This variable refers to the **Region** object that specifies the region of the newly created checkmark block. This parameter may be 0, in which case the region of the new block will be set to the region of the checkmark group.

result

[out, retval] A pointer to **ICheckmarkBlock*** pointer variable that receives the interface pointer of the new checkmark block.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

CheckmarkBlock
CheckmarkGroup

CheckmarkBlock Object (ICheckmarkBlock Interface)

This object provides access to specific properties of a checkmark block. The **ICheckmarkBlock** interface is derived from the **IBlock** interface and inherits all its properties. This object may be an element of the **CheckmarkGroup** collection.

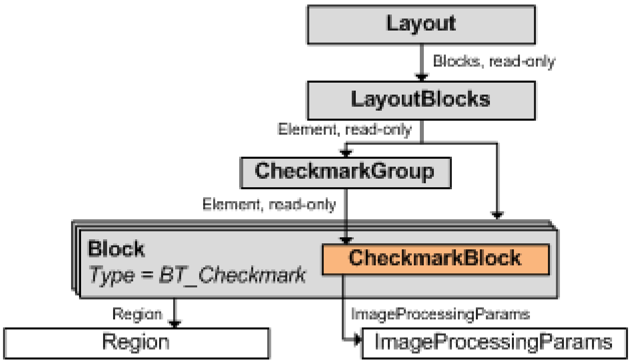
Properties

Name	Type	Description
Bottom	**Long**, read-only	Returns the coordinate of the bottom border of the checkmark rectangle in pixels. The rectangle is defined upon the deskewed black-and-white plane of the corresponding image.
CheckmarkState	**CheckmarkCheckStateEnum**	Specifies the state of the checkmark block.
CheckmarkType	**CheckmarkTypeEnum**	Specifies the checkmark type used for recognition. The default value is CMT_Empty. **Note:** This property must be identical for all checkmarks belonging to a single group.
ImageProcessingParams	**ImageProcessingParams**	Provides access to the set of properties affecting image preprocessing inside the checkmark block.
IsCorrectionEnabled	**Boolean**	This property set to TRUE means that checkmark block can be selected and then corrected. The default value is FALSE. **Note:** This property must be identical for all checkmarks belonging to a single group.
IsSuspicious	**Boolean**	This property set TRUE means that the checkmark was recognized uncertainly.
Left	**Long**, read-only	Returns the coordinate of the left border of the checkmark rectangle in pixels. The rectangle is defined upon the deskewed black-and-white plane of the corresponding image.
Right	**Long**, read-only	Returns the coordinate of the right border of the checkmark rectangle in pixels. The rectangle is defined upon the deskewed black-and-white plane of the corresponding image.
Top	**Long**, read-only	Returns the coordinate of the top border of the checkmark rectangle in pixels. The rectangle is defined upon the deskewed black-and-white plane of the corresponding image.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
SetRect	Sets the new rectangle for the checkmark.

Related objects



Output parameter

This object is the output parameter of the following methods and properties:

- **GetAsCheckmarkBlock** method of the **Block** object
- **Item**, **AddCheckmark** and **InsertCheckmark** methods of the **CheckmarkGroup** object

See also

Block
CheckmarkGroup
Working with Layout and Blocks
Recognizing Checkmarks
Working with Properties

SetRect Method of the CheckmarkBlock Object

This method allows you to set a rectangle for the checkmark block. It affects its **Left**, **Top**, **Right**, and **Bottom** properties. The rectangle is defined using pixel coordinates upon the deskewed black-and-white plane of the corresponding image.

Visual Basic Syntax

```
Method SetRect(
  left   As Long,
  top    As Long,
  right  As Long,
  bottom As Long
)
```

C++ Syntax

```
HRESULT SetRect(
  long left,
  long top,
  long right,
  long bottom
);
```

Parameters

left

[in] The coordinate of the left border of the rectangle.

top

[in] The coordinate of the top border of the rectangle.

right

[in] The coordinate of the right border of the rectangle.

bottom

[in] The coordinate of the bottom border of the rectangle.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

CheckmarkBlock

SeparatorGroup Object (ISeparatorGroup Interface)

This object represents a group of separator blocks. A group of separators usually includes four separators, which form a rectangle. For example, four lines of a table border are recognized as a separators group. The **ISeparatorGroup** interface is derived from the **IBlock** interface and inherits all its properties.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	Type	Description
Count	**Long**, read-only	Stores the number of elements in the group.
Element	SeparatorBlock, read-only	Provides access to a single element of the group.

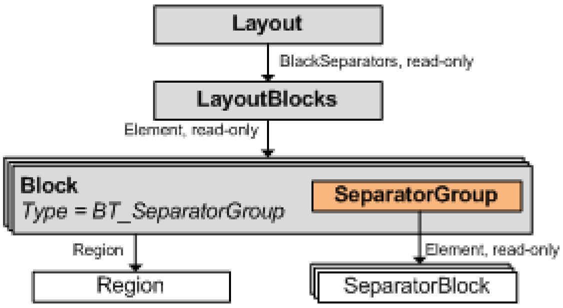
Methods

Name	Description
AddSeparator	Creates a separator block and adds it into the group.
InsertSeparator	Creates a separator block and inserts it into the specified position in the group.
Item	Provides access to a single element of the group of separator blocks.
Remove	Removes an element from the group.
RemoveAll	Removes all the elements from the group.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the **GetAsSeparatorGroup** method of the **Block** object.

See also

Block
SeparatorBlock
Working with Layout and Blocks
Working with Properties

Language–Related Objects

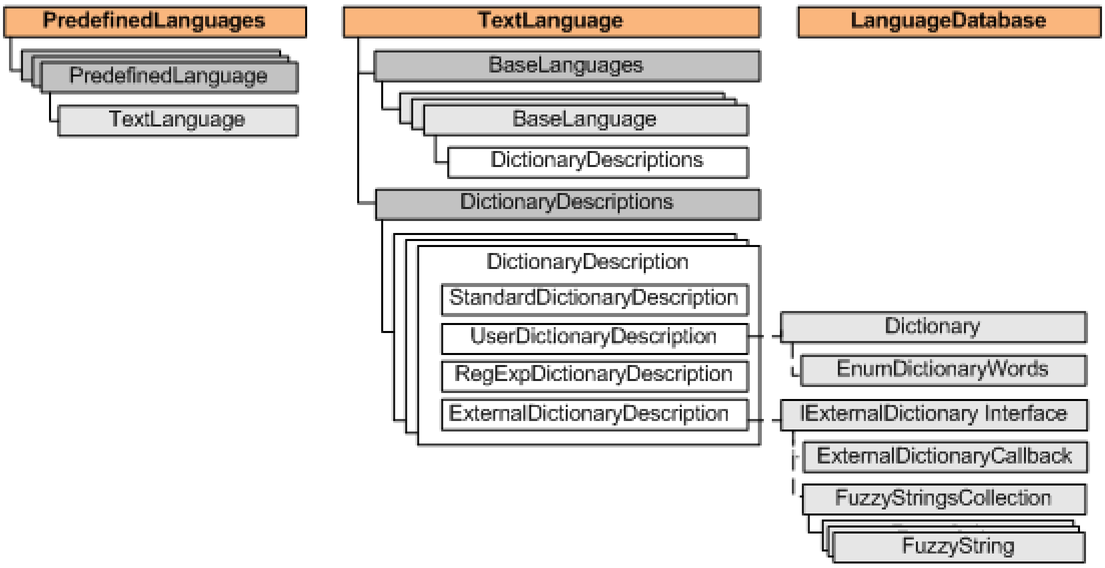
A recognition language for text is represented by the **TextLanguage** object. During the recognition the text is separated into words, and one or several recognition languages correspond to each word. One recognition language is assigned to each character in a word. This recognition language is represented by the **BaseLanguage** objects. Besides, this group of objects includes a collection of predefined languages — the recognition languages that ABBYY FineReader Engine supports by default. These are represented by the **PredefinedLanguages** object. A single predefined language is represented by the **PredefinedLanguage** object, and gives access to the corresponding **TextLanguage** object.

This section contains descriptions of the following language-related objects:

- **TextLanguage**
- **BaseLanguages**
- **BaseLanguage**
- **PredefinedLanguages**
- **PredefinedLanguage**
- **LanguageDatabase**
- **Dictionary**
- **EnumDictionaryWords**
- **DictionaryDescriptions**
- **DictionaryDescription**
- **StandardDictionaryDescription**
- **UserDictionaryDescription**
- **RegExpDictionaryDescription**
- **ExternalDictionaryDescription**
- **ExternalDictionaryCallback**
- **IExternalDictionary**
- **FuzzyStringsCollection**
- **FuzzyString**

You can find additional information in the Working with Languages and Working with Dictionaries sections.

The language-related objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

TextLanguage Object (ITextLanguage Interface)

This object represents the language of recognition for a text. The text language in general case is a set of base languages or languages of a single word. Access to the collection of base languages of a text is provided through the **BaseLanguages** property. Besides, this object exposes methods for accessing different text language attributes such as its internal name, groups of letter sets, etc.

The **TextLanguage** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

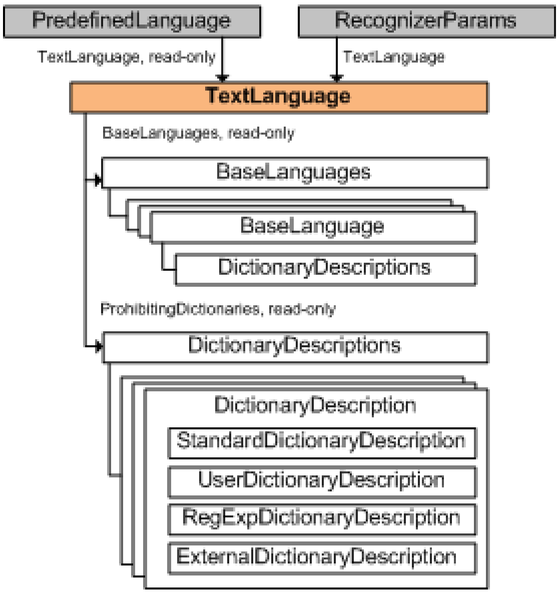
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BaseLanguages	**BaseLanguages**, read-only	Returns a reference to the collection of base languages of the current text language. This collection always exists though contains no elements by default.
ImpliedTextCategory	**TextCategoryEnum**	Specifies the category of text for which the current text language is designed. By default this property contains the **TC_Unknown** value, which means that the text language can be used for recognition of all types of text.
InternalName	**String**	Stores the internal name of the text language. As the internal name may be used to identify the language, it is better be unique. After a new object of the **TextLanguage** type is created, this property stores empty string. You may assign it some unique value to identify your text language among others.
LetterSet	**String**	Sets additional letter sets for the text language.
ProhibitingDictionaries	**DictionaryDescriptions**, read-only	Returns a reference to the collection of prohibiting dictionaries.
UserProperty	**VARIANT**	Allows you to associate any user-defined information with an object of the **TextLanguage** type.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.

LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **CreateTextLanguage**, **CreateCompoundTextLanguage** methods of the **LanguageDatabase** object,
- **CreateTextLanguage** method of the **Engine** object.

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);
   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );

   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage

   ' Copy all attributes from predefined English language
   TextLanguage.CopyFrom _
   Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
   TextLanguage.InternalName = "SampleTL"
   TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

   ' Create new user dictionary description
   Dim UserDic As FREngine.UserDictionaryDescription
   Set UserDic = Engine.CreateUserDictionaryDesc
   UserDic.FileName = DictionaryFile

   ' Bind new dictionary to first and single BaseLanguage object within TextLanguage
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic

 End Sub
```

See also

Working with Languages
Working with Properties

See sample: CustomLanguage

LetterSet Property of the TextLanguage Object

This property sets up additional letter sets for the text language. Every base language included in the collection of base languages of the text language provides its own letter sets. Each word in the text is recognized using a single base language, and thus its letter sets are applied to this word. This property specifies the number of letter sets that are applied to every recognized word irrespectively of the base language assigned to it.

Visual Basic Syntax

```
Property LetterSet(
   type As TextLanguageLetterSetEnum
)As String
```

C++ Syntax

```
HRESULT get_LetterSet(
   TextLanguageLetterSetEnum type,
   BSTR*                     pVal
);
HRESULT put_LetterSet(
   TextLanguageLetterSetEnum type,
   BSTR                      newVal
);
```

Parameters

type

[in] A variable of the **TextLanguageLetterSetEnum** type that describes the type of the letter set that you want to get or set.

pVal

[out] A pointer to **BSTR** variable that receives the value of this property. Must not be NULL.

newVal

[in] A variable of type **BSTR** that contains the new value for the property.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

See the description of **TextLanguageLetterSetEnum** type for the list of available types of letters sets for the text language. By default letter sets of each type are empty.

See also

TextLanguage
TextLanguageLetterSetEnum
Working with Properties

BaseLanguages Object (IBaseLanguages Interface)

This object is a collection of base languages. It contains methods for getting the number of languages in collection, accessing a single element in collection and iterating through a collection. The collection is accessible via the **TextLanguage** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

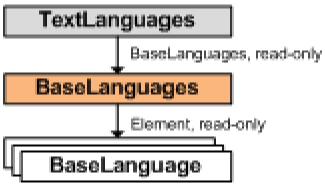
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	BaseLanguage, read-only	Provides access to a single element of the collection.

Methods

Name	**Description**
Add	Adds a new base language into the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



See also

TextLanguage
BaseLanguage
Working with Languages
Working with Properties

See sample: CustomLanguage

BaseLanguage Object (IBaseLanguage Interface)

This object represents a base recognition language. The **TextLanguage** object — a recognition language for a text — contains a collection of base languages. For example English or French languages may be represented by base languages. This object provides access to a base language attributes and allows you to get/set its internal name, letter sets, dictionary type, etc.

The **BaseLanguage** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

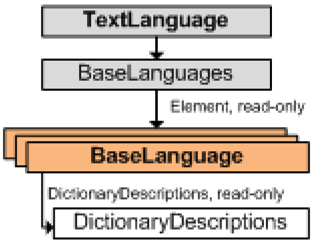
Name	Type	Description
AllowWordsFromDictionaryOnly	**Boolean**	Specifies if only the dictionary words are allowed during recognition in this base language. If this property is TRUE, a word that is not found in the dictionary of the base language can appear in the recognized text only if ABBYY FineReader Engine found no dictionary variants. If no dictionary is associated with the base language, the language will not be used for recognition.
Application	**Engine**, read-only	Returns the **Engine** object.
DictionaryDescriptions	DictionaryDescriptions, read-only	Returns a reference to the dictionary collection.
InternalName	**String**	Specifies the internal name of the base language. This name appears as an attribute of a character in the recognized text, so it is recommended that it were unique.
IsNaturalLanguage	**Boolean**	Specifies if this base language is a natural language. Natural languages are designed for recognizing common texts. Formal languages are not natural ones.
LanguageId	**LanguageIdEnum**	Defines the ID of the language. To convert it to Win32 LCID use the **IEngine::ConvertLanguageIdToLCID** method.
LetterSet	**String**	Provides access to the specified letter set of the base language.
UserProperty	**VARIANT**	Allows you to associate some user-defined information of any type with the **BaseLanguage** object.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.

SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of following methods:

- **CreateBaseLanguage** method of the **Engine** object
- **Item** method of the **BaseLanguages** object

Input parameter

This object is the input parameter of the **Add** method of the **BaseLanguages** object.

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );
   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create a new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to the dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create a new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage

   ' Copy all attributes from the predefined English language
   TextLanguage.CopyFrom _
   Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
   TextLanguage.InternalName = "SampleTL"
   TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

   ' Create a new user dictionary description
   Dim UserDic As FREngine.UserDictionaryDescription
   Set UserDic = Engine.CreateUserDictionaryDesc
   UserDic.FileName = DictionaryFile

   ' Bind the new dictionary to the first and single BaseLanguage object within
TextLanguage
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic
 End Sub
```

See also

BaseLanguages
Working with Languages
Working with Dictionaries
Working with Properties

See sample: CustomLanguage

LetterSet Property of the BaseLanguage Object

Every base language is characterized by a number of letter sets. These types are described by the **BaseLanguageLetterSetEnum** enumeration constants values. This property provides access to these letter sets. It allows you to get and set a specified letter set in a form of a string containing the letter set characters.

Visual Basic Syntax

```
Property LetterSet(
   type As BaseLanguageLetterSetEnum
)As String
```

C++ Syntax

```
HRESULT get_LetterSet(
   BaseLanguageLetterSetEnum type,
   BSTR*                     pVal
);
HRESULT put_LetterSet(
   BaseLanguageLetterSetEnum type,
   BSTR                      newVal
);
```

Parameters

type

[in] A variable of **BaseLanguageLetterSetEnum** type that describes the type of the letter set that you want to get or set.

pVal

[out, retval] A pointer to **BSTR** variable that receives the value of this property. Must not be NULL.

newVal

[in] A variable of **BSTR** type that contains the new value of the property.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

BaseLanguage
BaseLanguageLetterSetEnum
Working with Properties

PredefinedLanguages Object (IPredefinedLanguages Interface)

This object represents a collection of ABBYY FineReader Engine predefined languages. Predefined languages are languages supported by default. The collection of predefined languages is created upon ABBYY FineReader Engine initialization and exists until it is deinitialized. Besides standard collection functionality, this object exposes the **FindLanguage** method that allows you to get a **PredefinedLanguage** object by its internal name. The collection is accessible via the **Engine** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

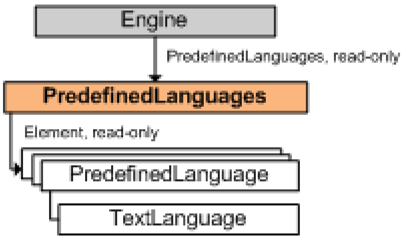
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	PredefinedLanguage, read-only	Provides access to a single element of the collection.

Methods

Name	**Description**
FindLanguage	Finds an element of the collection by its internal name.
Item	Provides access to a single element of the collection.

Related objects



See also

PredefinedLanguage
Working with Languages
List of ABBYY FineReader Engine predefined languages
Working with Properties

See sample: CustomLanguage

FindLanguage Method of the PredefinedLanguages Object

This method finds an element of the **PredefinedLanguages** collection by its internal name and returns a pointer to its interface.

Visual Basic Syntax

```
Method FindLanguage(
  internalName As String
) As PredefinedLanguage
```

C++ Syntax

```
HRESULT FindLanguage(
  BSTR                 internalName,
  IPredefinedLanguage** result
);
```

Parameters

internalName

[in] This variable specifies the internal name of the predefined language. For the list of available predefined languages see the List of ABBYY FineReader Engine predefined languages.

result

[out] A pointer to **IPredefinedLanguage*** pointer variable that receives the interface pointer of the **PredefinedLanguage** object with the specified internal name.

Return Values

If the specified predefined language is not available, this method returns **E_INVALIDARG** error code and NULL pointer to the predefined language. It may also return standard return values of ABBYY FineReader Engine functions.

Remarks

Availability of this or that predefined language depends on the availability of the corresponding module in the set of distributed components of ABBYY FineReader Engine.

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );

   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage

   ' Copy all attributes from predefined English language
   TextLanguage.CopyFrom _
   Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
   TextLanguage.InternalName = "SampleTL"
   TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

   ' Create new user dictionary description
   Dim UserDic As FREngine.UserDictionaryDescription
   Set UserDic = Engine.CreateUserDictionaryDesc
   UserDic.FileName = DictionaryFile

   ' Bind new dictionary to first and single BaseLanguage object within TextLanguage
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic

 End Sub
```

See also

PredefinedLanguage

See sample: CustomLanguage

PredefinedLanguage Object (IPredefinedLanguage Interface)

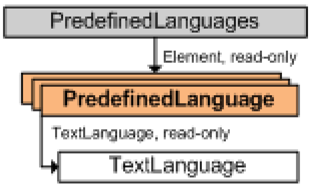
This object represents a single predefined language from a collection of ABBYY FineReader Engine predefined languages. Predefined languages are languages that are supported by default. This object contains properties reflecting predefined language attributes, such as its external name, components and category. Property **TextLanguage** contains the corresponding text language.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
ExternalName	**String**, read-only	Stores the external name of the predefined language. This name is localized and may be used in user interface. The value of this property depends on the current messages language (**IEngine::MessagesLanguage** property). For

		example if the messages language is English, then the name of the predefined language corresponding to the French, will be "French". If the messages language is French, then the name of the same predefined language will be "Français".
InternalName	**String**, read-only	Stores the internal name of the predefined language. It is this name that should be passed to the **IPredefinedLanguages::FindLanguage** method. For the list of available internal names of the predefined languages see List of ABBYY FineReader Engine predefined languages.
LanguageCategory	**LanguageCategoryEnum**, read-only	Indicates the category to which the current predefined language belongs. You may use this property to organize languages in your user interface.
TextLanguage	**TextLanguage**, read-only	Provides access to the **TextLanguage** object corresponding to the current predefined language. The **TextLanguage** object returned by this property is read-only (its modification methods return E_FAIL). Whenever you need to create an editable text language corresponding to a predefined recognition language, do the following two steps: 1. Create an empty **TextLanguage** object. 2. Call its **CopyFrom** method with a pointer to a predefined **TextLanguage** object's interface as its input parameter. A pointer to a predefined text language object's interface may be got from this property. You may want to use this property to initialize the **IRecognizerParams::TextLanguage** property with the value corresponding to the predefined language. The alternative way is to call the **IRecognizerParams::SetPredefinedTextLanguage** method.

Related objects



Output parameter

This object is the output parameter of the **Item**, **FindLanguage** methods of the **PredefinedLanguages** object.

See also

PredefinedLanguages
TextLanguage
List of ABBYY FineReader Engine predefined languages
Working with Languages
Working with Properties

See sample: CustomLanguage

LanguageDatabase Object (ILanguageDatabase Interface)

This object provides means for performing advanced operations with recognition languages. It allows you to work with the whole set of ABBYY FineReader Engine predefined languages, and also to import custom languages created with the use of ABBYY FineReader for using them by ABBYY FineReader Engine. This object allows you to create compound recognition language of several predefined languages and/or imported custom languages.

ABBYY FineReader with its user interface provides relatively simple way to create custom recognition languages (see details in the ABBYY FineReader help file). The procedure of creating and importing recognition languages is as follows:

1. Follow the instruction provided with ABBYY FineReader to create a custom language with the required parameters. The textlang.dat, *.amd files will be created. You may then redistribute them with your custom ABBYY FineReader Engine-based application.
2. Load the created languages with the use of the **ILanguageDatabase::LoadFrom** method.

After doing that, you may combine the loaded custom languages with each other and with predefined languages and use them for text recognition. You may choose not to load any custom languages into the language database. In this case only the predefined languages will be available.

When the languages are in use by ABBYY FineReader Engine, the files from which they were loaded should not be modified from outside (e.g. from ABBYY FineReader application), that is why we recommend unloading of ABBYY FineReader after the language database was created.

You may purchase additional language support applications and fonts at www.paratype.com/shop.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.

Methods

Name	**Description**
CreateCompoundTextLanguage	Creates the **TextLanguage** object of several predefined and/or custom languages included in the language database**.**
CreateTextLanguage	Creates the **TextLanguage** object of one or more predefined and/or custom languages included in the language database**.**
LoadFrom	Loads custom languages into the language database.

Output parameter

This object is the output parameter of the **CreateLanguageDatabase** method of the **Engine** object.

See also

Working with Languages

CreateCompoundTextLanguage Method of the LanguageDatabase Object

This method creates the **TextLanguage** object of several custom and/or predefined languages included in the **LanguageDatabase**. It is the **TextLanguage** object that specifies the recognition language for a text.

Visual Basic Syntax

```
Method CreateCompoundTextLanguage(
  languageNames As StringsCollection
) As TextLanguage
```

C++ Syntax

```
HRESULT CreateCompoundTextLanguage(
  IStringsCollection* languageNames,
  ITextLanguage**     pVal
);
```

Parameters

languageNames

[in] This parameter of the **StringsCollection** type specifies the names of the languages that are included into the language database. When creating custom languages in ABBYY FineReader, please give them names consisting of letters and digits and do not use names that include punctuation makrs (!@#$%^&*(), etc.). In case a language was given a name consisting of letters and digits, it appears for the **LanguageDatabase** object prefixed with the **@** symbol. For example, to retrieve the **TextLanguage** object for a user-defined language named "MyLanguage1", you should pass its name as "@MyLanguage1". Predefined internal languages' names are passed "as is", for example "English", "Russian".

pVal

[out] A pointer to the **ITextLanguage*** pointer variable that receives the interface pointer of the **TextLanguage** object. *pVal* should not be NULL. *pVal* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The language database must contain languages with all internal names that you pass to this function, otherwise an error code is returned. The resulting **TextLanguage** object will have attributes of all the custom languages put together.

See also

LanguageDatabase
ILanguageDatabase::CreateTextLanguage

CreateTextLanguage Method of the LanguageDatabase Object

This method creates the **TextLanguage** object of one or more custom languages included in the **LanguageDatabase**. It is the **TextLanguage** that specifies the recognition language for a text.

Visual Basic Syntax

```
Method CreateTextLanguage(
  languageName As String
) As TextLanguage
```

C++ Syntax

```
HRESULT CreateTextLanguage(
  BSTR            languageName,
  ITextLanguage** pVal
);
```

Parameters

languageName

[in] This parameter specifies the name of the language that is included in the language database. When creating custom languages in ABBYY FineReader please give them names consisting of letters and digits and do not use names that include punctuation marks (!@#$%^&*(), etc.). In case a language was given a name consisting of letters and digits, it appears for the **LanguageDatabase** object prefixed with the **@** symbol. For example, to retrieve the **TextLanguage** for a user-defined language named "MyLanguage1", you should pass here the string "@MyLanguage1". This parameter may contain several languages names divided by commas, for example "@MyLanguage1,@MyLanguage2,English".

pVal

[out] A pointer to the **ITextLanguage*** pointer variable that receives the interface pointer of the **TextLanguage** object. *pVal* should not be NULL. *pVal* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The language database must contain languages with all internal names that you pass to this function, otherwise an error code is returned. If several languages are passed to this method, the resulting **TextLanguage** object will have attributes of all the custom languages put together.

See also

LanguageDatabase
ILanguageDatabase::CreateCompoundTextLanguage

LoadFrom Method of the LanguageDatabase Object

This method loads custom languages that you created in ABBYY FineReader into the **LanguageDatabase** object.

Visual Basic Syntax

```
Method LoadFrom(
  folderPath As String
)
```

C++ Syntax

```
HRESULT LoadFrom(
  BSTR folderPath
);
```

Parameters

folderPath

[in] This parameter specifies the full path to folder where the necessary files are stored. These files are textlang.dat and *.amd.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

You may initialize the **LanguageDatabase** object before using it, otherwise only predefined languages will be available through it.

See also

LanguageDatabase

Dictionary Object (IDictionary Interface)

This object is designed for working with user dictionaries. It cannot be applied to ABBYY FineReader Engine standard dictionary files. User dictionaries are dictionaries that contain word forms of words of a certain language. Each word form in the dictionary has its own weight that defines its priority when there appear several variants for a word during recognition. The weight is a number that may have four discrete values: 25, 50, 100 and 200. User dictionaries may be connected to the **BaseLanguage** object — object representing one base recognition language.

A pointer to the **Dictionary** object interface may be got either from the **IEngine::CreateNewDictionary** or **IEngine::OpenExistingDictionary** methods. The **IEngine::OpenExistingDictionary** method can open dictionaries created with the help of the **IEngine::CreateNewDictionary** method, as well as user dictionaries (*.amd) created in ABBYY FineReader. These dictionaries are created together with user languages and are saved in the folder of the current batch. For more details on the creation of user languages and dictionaries, see the ABBYY FineReader Help file.

The **Edit** method displays the **Dictionary** dialog box that allows a user to edit the dictionary and to import any text file in Windows ANSI- and Unicode-encoding (the only requirement is that words must be separated by spaces or other non-alphabetic characters).

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Name	**String**	Stores the name of the dictionary. It is this name that is displayed at the caption of the **Dictionary** dialog box that is displayed when calling the **Edit** method. After creation of the **Dictionary** object this property stores the name of the dictionary file (without path and extension). You may assign it any other value. This property is not saved into the file associated with the dictionary, and should be initialized every time the dictionary is edited.
WordsCount	**Long**, read-only	Returns the number of words in the dictionary.

Methods

Name	Description
AddWord	Adds a word to the dictionary.
AddWords	Adds a group of words to the dictionary.
DeleteAllWords	Deletes all words from the dictionary.
DeleteWord	Deletes a word from the dictionary.
DeleteWords	Deletes a group of words from the dictionary.
Edit	Displays the **Dictionary** dialog box that allows a user to edit the dictionary.
EnumWords	Returns an object of the **EnumDictionaryWords** type that allows you to iterate through the words in the dictionary.

Output parameter

This object is the output parameter of the **CreateNewDictionary** and **OpenExistingDictionary** methods of the **Engine** object.

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );

   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage
   ' Copy all attributes from predefined English language
   TextLanguage.CopyFrom _
   Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
   TextLanguage.InternalName = "SampleTL"
   TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

   ' Create new user dictionary description
   Dim UserDic As FREngine.UserDictionaryDescription
   Set UserDic = Engine.CreateUserDictionaryDesc
   UserDic.FileName = DictionaryFile

   ' Bind new dictionary to first and single BaseLanguage object within TextLanguage
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic

 End Sub
```

See also

UserDictionaryDescription
Working with Dictionaries
Working with Properties

See sample: CustomLanguage

AddWord Method of the Dictionary Object

This method adds a new word to the dictionary.

Visual Basic Syntax

```
Method AddWord(
  word   As String,
  weight As Long
)
```

C++ Syntax

```
HRESULT AddWord(
  BSTR word,
  long weight
);
```

Parameters

word

[in] This parameter contains the newly added word.

weight

[in] The weight assigned to the word in the dictionary. Must be in the range from 1 to 200. The higher the weight for a word is, the more likely this word will be taken as a variant during recognition. The normal value for this parameter is 100. Visual Basic users see this parameter as having default value of 100. The weight assigned to the word in the dictionary may have a set of discrete values only. These values are 25, 50, 100, 200. The value passed in this parameter is rounded to the nearest of the discrete set of values.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

It is not recommended to use this method for adding a large number of words to the dictionary, because after adding each word the dictionary is rebuilt, and thus the operation takes a rather long time. For adding a group of words into the dictionary, use the **IDictionary::AddWords** method instead.

Sample

⊞**Visual C++ (COM) code**

```
// Global FineReader Engine object.
FREngine::IEnginePtr Engine;
// Create new dictionary
```

```
_bstr_t dictionaryFile = Engine->
      Path + "\\..\\Samples\\SampleImages\\sample.amd";
FREngine::IDictionaryPtr pDictionary =
Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
pDictionary->Name = "Sample";
// Add words to dictionary
pDictionary->AddWord( "the", 100 );
pDictionary->AddWord( "a", 100 );
pDictionary->AddWord( "an", 100 );
```

⊞**Visual Basic code**

```
Public Engine As FREngine.Engine
' Create new dictionary
Dim DictionaryFile As String
DictionaryFile = Engine.Path & "\..\Samples\SampleImages\sample.amd"
Dim Dictionary As FREngine.Dictionary
Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
Dictionary.Name = "Sample"
' Add words to dictionary
Dictionary.AddWord "the"
Dictionary.AddWord "a"
Dictionary.AddWord "an"
```

See also

Dictionary
IDictionary::AddWords

See sample: CustomLanguage

AddWords Method of the Dictionary Object

This method adds a group of words to the dictionary.

Visual Basic Syntax

```
Method AddWords(
  words   As StringsCollection,
  weights As LongsCollection
)
```

C++ Syntax

```
HRESULT AddWords(
  IStringsCollection* words,
  ILongsCollection*   weights
);
```

Parameters

words

[in] This parameter of the **StringsCollection** type contains the collection of the newly added words.

weights

[in] This parameter of the **LongsCollection** type that must have the same size as the collection of words, is used to pass information about the weights for the newly added words. The weights for the words must be in the range from 1 to 200. You may pass 0 for this parameter in which case all the words will be included in the dictionary with default weights of 100. The weight assigned to the word in the dictionary may have a set of discrete values only. These values are 25, 50, 100, 200. The value passed in this parameter is rounded to the nearest of the discrete set of values.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

For more efficient operation it is recommended to pre-sort the added words in alphabetical order.

For adding one word into the dictionary, you can use the **IDictionary::AddWord** method.

See also

Dictionary
IDictionary::AddWord

DeleteWord Method of the Dictionary Object

This method deletes a word from the dictionary.

Visual Basic Syntax

```
Method DeleteWord(
  word As String
)
```

C++ Syntax

```
HRESULT DeleteWord(
  BSTR word
);
```

Parameters

word

[in] This parameter contains the word that is to be deleted. If the deleted word is not present in the dictionary, no error occurs.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

To delete a group of words or all words from the dictionary, use the **IDictionary::DeleteWords** or **IDictionary::DeleteAllWords** method, respectively.

See also

Dictionary
IDictionary::DeleteWords
IDictionary::DeleteAllWords

DeleteWords Method of the Dictionary Object

This method deletes a group of words from the dictionary.

Visual Basic Syntax

```
Method DeleteWords(
  words As StringsCollection
)
```

C++ Syntax

```
HRESULT DeleteWords(
  IStringCollection* words
);
```

Parameters

words

[in] This parameter of the **StringsCollection** type contains the collection of words to be deleted. If any of the deleted words is not present in the dictionary, no error will occur.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

To delete a single word or all words from the dictionary, use the **IDictionary::DeleteWord** or **IDictionary::DeleteAllWords** method, respectively.

See also

Dictionary
IDictionary::DeleteWord
IDictionary::DeleteAllWords

Edit Method of the Dictionary Object

This method displays the **Dictionary** dialog box that allows a user to edit the dictionary. This dialog box allows a user to import any text file in Windows ANSI- and Unicode-encoding (the only requirement is that words must be separated by spaces or other non-alphabetic characters).

Visual Basic Syntax

```
Method Edit()
```

C++ Syntax

```
HRESULT Edit();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The dialog box that this methods displays will have the value of the **IDictionary::Name** property as its caption. For correct operation of this method it is necessary to assign a correct value to the **IEngine::ParentWindow** property.

See also

Dictionary
Engine

The Dictionary Dialog Box

Option	Option Description
The top dialog box field	Enter the word you want to add to the dictionary here.
Word list	Displays all user dictionary words. Select the word you want to delete.
Add (button)	Adds the word you typed in the top line to the dictionary.
View (button)	Displays the paradigm of the selected word. **Note**: Not available now.
Delete (button)	Deletes the selected word from the dictionary.
Import (button)	Imports an already existing dictionary (for example, any text file in Windows ANSI- and Unicode-encoding (the only requirement is that words must be separated by spaces or other non-alphabetic characters).
Export (button)	Exports your ABBYY FineReader user dictionary.

See also

Dictionary
IDictionary::Edit

EnumWords Method of the Dictionary Object

This method returns an object of the **EnumDictionaryWords** type that allows you to iterate through the words in the dictionary. This method makes a copy of the dictionary, and thus all the modifications that are performed upon the dictionary after getting the **EnumDictionaryWords** object do not affect the latter. That is, if a word is added to the dictionary after the **EnumDictionaryWords** object was received for it, this word will not be included into iteration.

Visual Basic Syntax

```
Method EnumWords(
  result As EnumDictionaryWords
)
```

C++ Syntax

```
HRESULT EnumWords(
  IEnumDictionaryWords** result
);
```

Parameters

result

[out, retval] A pointer to **IEnumDictionaryWords*** pointer variable that receives the interface pointer to the returned **EnumDictionaryWords** object. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Dictionary

EnumDictionaryWords Object (IEnumDictionaryWords Interface)

This object serves for iterating words included in a user-defined dictionary. The user-defined dictionary is represented by the **Dictionary** object. The **EnumDictionaryWords** object is got from the **IDictionary::EnumWords** method. All modifications to the parent **Dictionary** object after receiving its enumerator object do not affect the latter. That is, if a new word is added to the dictionary, it will not appear in the iteration.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Dictionary	**Dictionary**, read-only	Refers to the parent object of **Dictionary** type. It is the **IDictionary::EnumWords** method of this object that generated the current one.

Methods

Name	Description
Next	Retrieves the next word from the iteration sequence.
Reset	Restarts the iteration.

Output parameter

This object is the output parameter of the **IDictionary::EnumWords** method.

See also

Dictionary
Working with Dictionaries
Working with Properties

Next Method of the EnumDictionaryWords Object

This method retrieves the next word from the iteration sequence together with the word's weight in the dictionary.

Visual Basic Syntax

```
Method Next(
  confidence As Long
) As String
```

C++ Syntax

```
HRESULT Next(
  long* confidence,
  BSTR* result
);
```

Parameters

confidence

[out] This parameter serves for passing out the confidence of the word. Confidence or weight of the word in the dictionary defines the priority level for a word; this value is used to choose among word variants during text recognition. The higher this value is, the more preferable is this variant for the recognized word. When the iteration is over, 0 will be assigned to this parameter.

result

[out, retval] A pointer to the **BSTR** variable that receives the return value of this method — word from the dictionary. When the iteration is over, 0 is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

EnumDictionaryWords

Reset Method of the EnumDictionaryWords Object

This method restarts the iteration. This method may be called any time during iteration to reset the iteration.

Visual Basic Syntax

```
Method Reset()
```

C++ Syntax

```
HRESULT Reset();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

EnumDictionaryWords

DictionaryDescriptions Object (IDictionaryDescriptions Interface)

This object is a collection of dictionary descriptions. It contains methods for getting the number of dictionaries in a collection, accessing a single element in a collection and iterating through a collection. The collection can include **StandardDictionaryDescription**, **UserDictionaryDescription**, **RegExpDictionaryDescription**, and **ExternalDictionaryDescription** objects, which are the descriptions of different dictionary types. These objects are child objects of the **DictionaryDescription** object. You can add any of these objects to a collection with the help of the **Add** method. If you use the **Item** method, you will only be able to get the **DictionaryDescription** object which can later be cast to any of the abovementioned types.

The **IBaseLanguage::DictionaryDescriptions** property provides access to the dictionary descriptions collection. The collection of prohibiting dictionaries is accessible via the **TextLanguage** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

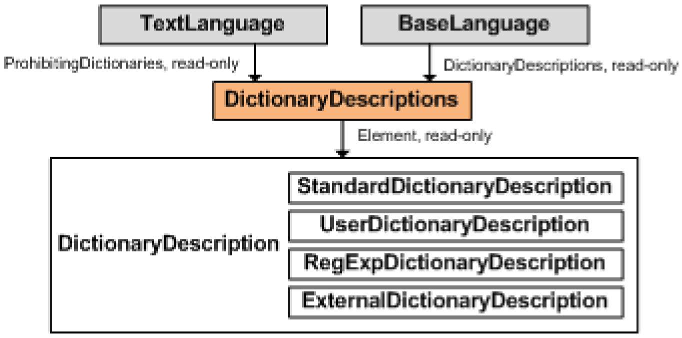
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	DictionaryDescription, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a new dictionary to the collection.

Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects


TextLanguage
BaseLanguage
ProhibitingDictionaries, read-only
DictionaryDescriptions, read-only
DictionaryDescriptions
Element, read-only
DictionaryDescription
StandardDictionaryDescription
UserDictionaryDescription
RegExpDictionaryDescription
ExternalDictionaryDescription

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );

   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage

   ' Copy all attributes from predefined English language
   TextLanguage.CopyFrom _
   Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
   TextLanguage.InternalName = "SampleTL"
   TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

   ' Create new user dictionary description
   Dim UserDic As FREngine.UserDictionaryDescription
   Set UserDic = Engine.CreateUserDictionaryDesc
   UserDic.FileName = DictionaryFile

   ' Bind new dictionary to first and single BaseLanguage object within TextLanguage
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic

 End Sub
```

See also

Working with Dictionaries
DictionaryDescription
StandardDictionaryDescription
UserDictionaryDescription
RegExpDictionaryDescription
ExternalDictionaryDescription
Working with Properties

See sample: CustomLanguage

Add Method of the DictionaryDescriptions Object

This method adds a new dictionary description into the collection.

Visual Basic Syntax

```
Method Add(
  description As DictionaryDescription
)
```

C++ Syntax

```
HRESULT Add(
  IDictionaryDescription* description
);
```

Parameters

description

[in] This parameter refers to the object representing the newly added dictionary description, it may be a **StandardDictionaryDescription**, **UserDictionaryDescription**, **RegExpDictionaryDescription**, or **DictionaryDescription** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );

   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage
```

```
    ' Copy all attributes from predefined English language
    TextLanguage.CopyFrom _
    Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
    TextLanguage.InternalName = "SampleTL"
    TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

    ' Create new user dictionary description
    Dim UserDic As FREngine.UserDictionaryDescription
    Set UserDic = Engine.CreateUserDictionaryDesc
    UserDic.FileName = DictionaryFile

    ' Bind new dictionary to first and single BaseLanguage object within TextLanguage
    TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
    TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic

End Sub
```

See also

DictionaryDescriptions

See sample: CustomLanguage

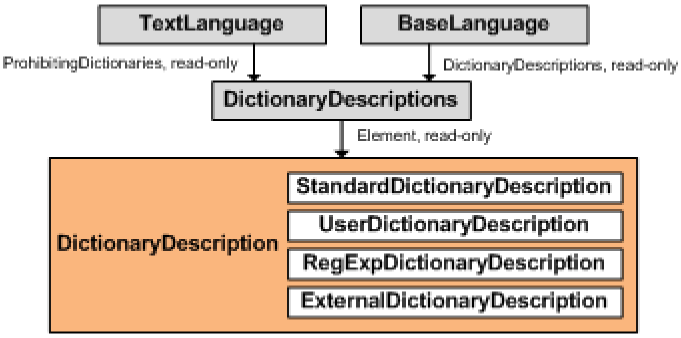
DictionaryDescription Object (IDictionaryDescription Interface)

This object is a dictionary description which may be typecast to one of its child objects: **StandardDictionaryDescription**, **UserDictionaryDescription**, **RegExpDictionaryDescription**, or **ExternalDictionaryDescription**. These objects provide access to descriptions of four different dictionary types and inherit all the properties of the **DictionaryDescription** object. They are also elements of the **DictionaryDescriptions** collection.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Type	DictionaryTypeEnum, read-only	Returns the type of the dictionary.
Weight	**Long**	Stores the dictionary weight in percentage points. This value must be non-negative. By default, this property is set to 100%.

Related objects


TextLanguage
BaseLanguage
ProhibitingDictionaries, read-only
DictionaryDescriptions, read-only
DictionaryDescriptions
Element, read-only
DictionaryDescription
StandardDictionaryDescription
UserDictionaryDescription
RegExpDictionaryDescription
ExternalDictionaryDescription

Output parameter

This object is the output parameter of the **Item** method of the **DictionaryDescriptions** object.

See also

Working with Dictionaries
DictionaryDescriptions
StandardDictionaryDescription
UserDictionaryDescription
RegExpDictionaryDescription
ExternalDictionaryDescription
Working with Properties

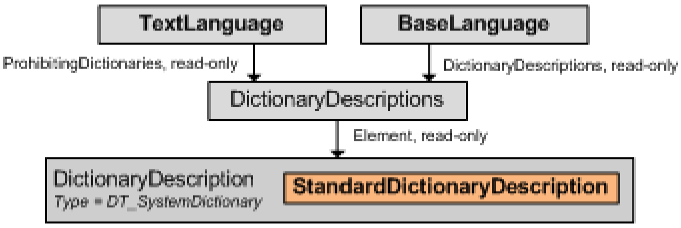
StandardDictionaryDescription Object (IStandardDictionaryDescription Interface)

This object provides access to a standard dictionary. The **IStandardDictionaryDescription** interface is a child object of the **IDictionaryDescription** interface and inherits all its properties.

Properties

Name	Type	Description
CanUseTrigrams	**Boolean**	Allows or forbids the use of dictionary-based trigrams. By default, the value is TRUE.
LanguageId	**LanguageIdEnum**	Defines the ID of the language. To convert it to Win32 LCID, use the **IEngine::ConvertLanguageIdToLCID** method.

Related objects


TextLanguage
BaseLanguage
ProhibitingDictionaries, read-only
DictionaryDescriptions, read-only
DictionaryDescriptions
Element, read-only
DictionaryDescription
Type = DT_SystemDictionary
StandardDictionaryDescription

Output parameter

This object is the output parameter of the IEngine::CreateStandardDictionaryDesc and IDictionaryDescriptions::Item methods.

Input parameter

This object is the input parameter of the **IDictionaryDescriptions::Add** method.

See also

Working with Dictionaries
DictionaryDescription
DictionaryDescriptions
Working with Properties

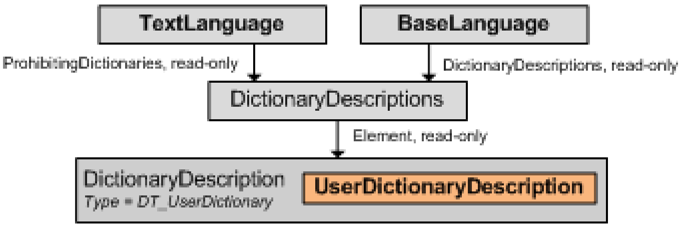
UserDictionaryDescription Object (IUserDictionaryDescription Interface)

This object provides access to a user dictionary. The **IUserDictionaryDescription** interface is a child object of the **IDictionaryDescription** interface and inherits all its properties. A user dictionary can be a dictionary created by the user with the help of the **Dictionary** object or a user dictionary (*.amd) created in ABBYY FineReader (see the ABBYY FineReader help file for more details).

Properties

Name	Type	Description
FileName	**String**	The path to the user dictionary file. This parameter does not check the validity of the dictionary, which will be done later when the dictionary is used.

Related objects



Output parameter

This object is the output parameter of the IEngine::CreateUserDictionaryDesc and IDictionaryDescriptions::Item methods.

Input parameter

This object is the input parameter of the **IDictionaryDescriptions::Add** method.

Sample

Visual C++ (COM) code

```
FREngine::ITextLanguagePtr MakeTextLanguage()
 {
   // Create new dictionary
   _bstr_t dictionaryFile = L"D:\\sample.amd";

   FREngine::IDictionaryPtr pDictionary =
   Engine->CreateNewDictionary( dictionaryFile, FREngine::LI_EnglishUnitedStates );
   pDictionary->Name = "Sample";

   // Add words to dictionary
   pDictionary->AddWord( "the", 100 );
   pDictionary->AddWord( "a", 100 );
   pDictionary->AddWord( "an", 100 );

   // Create new TextLanguage object
   FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();

   // Copy all attributes from predefined English language
   FREngine::ITextLanguagePtr pEnglishLanguage =
             Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
   pTextLanguage->CopyFrom( pEnglishLanguage );
   pTextLanguage->InternalName = "SampleTL";

   // Bind new dictionary to first (and single) BaseLanguage object within TextLanguage
   FREngine::IBaseLanguagePtr pBaseLanguage = pTextLanguage->BaseLanguages->Item(0);

   // Change internal dictionary name to user-defined
   pBaseLanguage->InternalName = "SampleBL";

   // Get collection of dictionary descriptions and remove all items
   FREngine::IDictionaryDescriptionsPtr pDictionaryDescriptions =
   pBaseLanguage->DictionaryDescriptions;
   pDictionaryDescriptions->RemoveAll();

   // Create user dictionary description and add it to the collection
   FREngine::IUserDictionaryDescriptionPtr userDic =
   Engine->CreateUserDictionaryDesc();

   userDic->FileName = dictionaryFile;

   pDictionaryDescriptions->Add( userDic );

   return pTextLanguage;
 }
```

Visual Basic code

```
Private Sub MakeTextLanguage(TextLanguage As FREngine.TextLanguage)
   ' Create new dictionary
   Dim DictionaryFile As String
   DictionaryFile = "D:\sample.amd"

   Dim Dictionary As FREngine.Dictionary
   Set Dictionary = Engine.CreateNewDictionary(DictionaryFile, LI_EnglishUnitedStates)
   Dictionary.Name = "Sample"

   ' Add words to dictionary
   Dictionary.AddWord "the"
   Dictionary.AddWord "a"
   Dictionary.AddWord "an"

   ' Create new TextLanguage object
   Set TextLanguage = Engine.CreateTextLanguage

   ' Copy all attributes from predefined English language
   TextLanguage.CopyFrom _
   Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
   TextLanguage.InternalName = "SampleTL"
   TextLanguage.BaseLanguages(0).InternalName = "SampleBL"

   ' Create new user dictionary description
   Dim UserDic As FREngine.UserDictionaryDescription
   Set UserDic = Engine.CreateUserDictionaryDesc
   UserDic.FileName = DictionaryFile

   ' Bind new dictionary to first and single BaseLanguage object within TextLanguage
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.RemoveAll
   TextLanguage.BaseLanguages(0).DictionaryDescriptions.Add UserDic

 End Sub
```

See also

Working with Dictionaries
Dictionary
DictionaryDescription
DictionaryDescriptions
Working with Properties

See sample: CustomLanguage

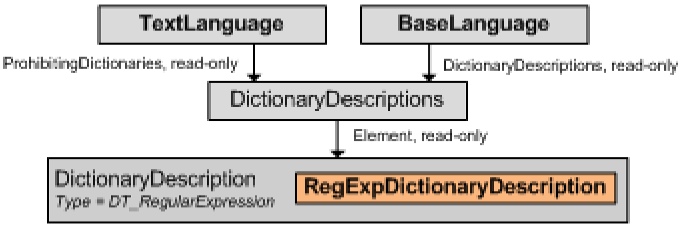
RegExpDictionaryDescription Object (IRegExpDictionaryDescription Interface)

This object provides access to a regular-expression-based dictionary. The **IRegExpDictionaryDescription** interface is a child object of the **IDictionaryDescription** interface and inherits all its properties.

Methods

Name	Description
SetText	Sets a regular expression. See for details Working with ABBYY FineReader Engine Regular Expressions.

Related objects


TextLanguage
BaseLanguage
ProhibitingDictionaries, read-only
DictionaryDescriptions, read-only
DictionaryDescriptions
Element, read-only
DictionaryDescription
Type = DT_RegularExpression
RegExpDictionaryDescription

Output parameter

This object is the output parameter of the IEngine::CreateRegExpDictionaryDesc and IDictionaryDescriptions::Item methods.

Input parameter

This object is the input parameter of the **IDictionaryDescriptions::Add** method.

See also

Working with ABBYY FineReader Engine Regular Expressions
Working with Dictionaries
DictionaryDescription
DictionaryDescriptions
Working with Properties

SetText Method of the RegExpDictionaryDescription Object

This method sets regular expression. The regular expression is passed as an input parameter to this method, then its semantics is checked.

Visual Basic Syntax

```
Method SetText(
  newVal As String
)
```

C++ Syntax

```
HRESULT SetText(
  BSTR newVal
);
```

Parameters

newVal

[in] This variable contains the regular expression. For example, the regular expression [0-9]+ specifies that the dictionary accepts the "words" made up of one or more digits.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RegExpDictionaryDescription
Working with ABBYY FineReader Engine Regular Expressions

ExternalDictionaryDescription Object (IExternalDictionaryDescription Interface)

This object provides access to an external dictionary. The **IExternalDictionaryDescription** interface is a child object of the **IDictionaryDescription** interface and inherits all its properties. The external dictionary is represented as the **IExternalDictionary** interface which is implemented on the client side. This interface allows you to implement your own type of dictionary. You can attach a dictionary with the help of the **SetDictionary** method of the **ExternalDictionaryDescription** object.

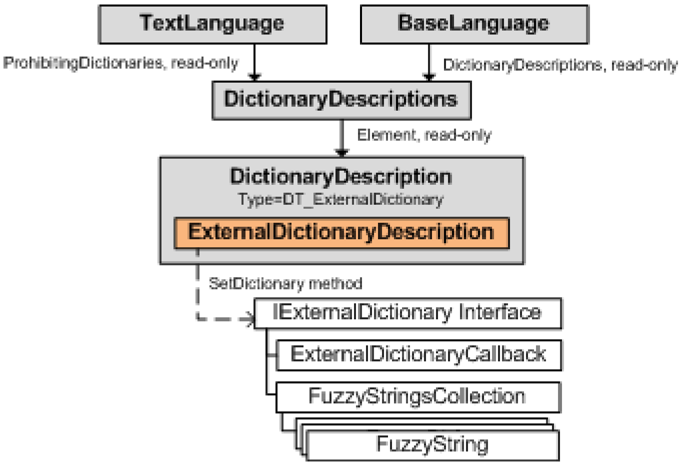
Properties

Name	Type	Description
CheckPrefixes	**Boolean**	If this property is FALSE, the external dictionary will not be used for prefixes check, and the **CheckPrefix** method of the **IExternalDictionary** interface will not be called. The default value is TRUE.
FullFuzzySupport	**Boolean**	If this property is FALSE, the fuzzy string will contain only capital letters as the recognition variant. The default value is TRUE.

Methods

Name	Description
SetDictionary	Attaches an external dictionary.

Related objects


TextLanguage
BaseLanguage
ProhibitingDictionaries, read-only
DictionaryDescriptions, read-only
DictionaryDescriptions
Element, read-only
DictionaryDescription
Type=DT_ExternalDictionary
ExternalDictionaryDescription
SetDictionary method
IExternalDictionary Interface
ExternalDictionaryCallback
FuzzyStringsCollection
FuzzyString

Output parameter

This object is the output parameter of the IEngine::CreateExternalDictionaryDesc and IDictionaryDescriptions::Item methods.

Input parameter

This object is the input parameter of the **IDictionaryDescriptions::Add** method.

See also

Working with Dictionaries
DictionaryDescription
DictionaryDescriptions
Working with Properties

SetDictionary Method of the ExternalDictionaryDescription Object

This method attaches an external dictionary.

Visual Basic Syntax

```
Method SetDictionary(
  Dictionary As ExternalDictionary
)
```

C++ Syntax

```
HRESULT SetDictionary(
  IExternalDictionary* Dictionary
);
```

Parameters

Dictionary

[in] This parameter contains a pointer to the **IExternalDictionar**y interface which represents an external dictionary.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ExternalDictionaryDescription

ExternalDictionaryCallback Object (IExternalDictionaryCallback Interface)

This is a callback interface which is used to deliver information about dictionary words to the recognizer.

Methods

Name	Description
ExternalDictionaryResult	Delivers information about dictionary words to the recognizer. It is called from the **IExternalDictionary::CheckWords** method which is implemented on the client side.

Input parameter

This object is the input parameter of the **IExternalDictionary::CheckWords** method.

See also

IExternalDictionary

ExternalDictionaryResult Method of the ExternalDictionaryCallback Object

This method delivers information about dictionary words to the recognizer. It is called from the **IExternalDictionary::CheckWords** method which is implemented on the client side. The input parameters of this method are: the dictionary word, the word confidence in percentage, and the index of the word in the collection which is passed from the **CheckWords** method of the **IExternalDictionary** interface. The dictionary word must be composed from characters of the corresponding fuzzy string.

Visual Basic Syntax

```
Method ExternalDictionaryResult(
  Word         As String,
  Confidence   As Long,
  RequestIndex As Long
)
```

C++ Syntax

```
HRESULT ExternalDictionaryResult(
  BSTR Word,
  long Confidence,
  long RequestIndex
);
```

Parameters

Word

[in] This parameter contains the word from an external dictionary.

Confidence

[in] This parameter contains the word confidence in percentage.

RequestIndex

[in] This parameter contains the index of the word in the **FuzzyStringsCollection** collection

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ExternalDictionaryCallback

IExternalDictionary Interface

This is the interface for an external dictionary. This interface and all its methods are implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods.

ABBYY FineReader Engine objects allow working with the following dictionary types: standard, user, and regular-expression-based. If these dictionary types are not convenient for you, the **IExternalDictionary** interface allows you to implement your own dictionary type. You can attach your dictionary with the help of the **SetDictionary** method of the **ExternalDictionaryDescription** object. See the Working with Dictionaries section for details.

Methods

Name	Description
CheckPrefix	Determines if the dictionary contains a word with the specified prefix.
CheckWords	Delivers to the recognizer the information about strings in the collection which contains the dictionary words, with the help of the **ExternalDictionaryResult** method of the **ExternalDictionaryCallback** object.

Input parameter

The ExternalDictionary object is the input parameter of the IExternalDictionaryDescription::SetDictionary method.

See also

IExternalDictionaryCallback
ExternalDictionaryDescription
Working with Dictionaries

CheckPrefix Method of the IExternalDictionary Interface

This method is implemented on the client side. This method determines if the dictionary contains a word with the specified prefix. It must return TRUE, if the dictionary contains at least one word with the prefix which is specified as a fuzzy string. This method is called during recognition of difficult cases, if the **CheckPrefixes** property of the **ExternalDictionaryDescription** object is set to TRUE.

Visual Basic Syntax

```
Sub CheckPrefix(
  ByRef Prefix As FuzzyString,
  ByRef Result As Boolean
)
```

C++ Syntax

```
HRESULT CheckPrefix(
  IFuzzyString* Prefix,
  VARIANT_BOOL* Result
);
```

Parameters

Prefix

[in] This parameter contains the fuzzy string.

Result

[out, retval] This parameter is TRUE if the dictionary contains at least one word with the prefix which is specified in the *Prefix* parameter as a fuzzy string.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

- The pointer to the **FuzzyString** object which was used as the *Prefix* parameter is released automatically after the end of the **CheckPrefix** method execution, therefore you do not need to call the **Release** method for this object in the **CheckPrefix** method implementation.

- The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IExternalDictionary

CheckWords Method of the IExternalDictionary Interface

This method is implemented on the client side. It is called during recognition, and it is received as a collection of fuzzy strings. The number of fuzzy strings in the collection may vary, depending on the recognized variants of the word. This method delivers to the recognizer information about strings in the collection which contains the dictionary words, with the help of the **ExternalDictionaryResult** method of the **ExternalDictionaryCallback** object. If the **ExternalDictionaryResult** method is not called for a fuzzy string, the recognizer assumes that proper words have not been found in the dictionary.

Visual Basic Syntax

```
Sub CheckWords(
  ByRef Request  As FuzzyStringsCollection,
  ByRef Callback As ExternalDictionaryCallback
)
```

C++ Syntax

```
HRESULT CheckWords(
  IFuzzyStringsCollection*     Request,
  IExternalDictionaryCallback* Callback
);
```

Parameters

Request

[in] This variable refers to the **FuzzyStringsCollection** object corresponding to the fuzzy strings collection.

Callback

[in] This variable refers to the **ExternalDictionaryCallback** object. The recognizer receives information about dictionary words from this object.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

- The pointers to the **FuzzyStringsCollection** and **ExternalDictionaryCallback** objects which were used as the *Request* and *Callback* parameters are released automatically after the end of the **CheckWords** method execution, therefore you do not need to call the **Release** method for these objects in the **CheckPrefix** method implementation.

- The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IExternalDictionary

FuzzyStringsCollection Object (IFuzzyStringsCollection Interface)

This object represents a collection of **FuzzyString** objects. It is a supplementary interface for external dictionaries.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**FuzzyString**	Provides access to a single element of the collection.

Methods

Name	**Description**
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Input parameter

This collection is the input parameter of the **CheckWords** method of the **IExternalDictionary** interface.

See also

IExternalDictionary
Working with Dictionaries
Working with Properties

FuzzyString Object (IFuzzyString Interface)

This object represents a fuzzy string. A fuzzy string contains recognition variants for each character of a word. One or several fuzzy strings correspond to each recognized word. For example, the following fuzzy strings can correspond to the "hello" word:

<table>
<tr><th>Word</th><th>Position</th><th colspan="4">Fuzzy Strings
(each cell is a value of the CharacterVariants property)</th></tr>
<tr><td>h</td><td>1</td><td>hn</td><td>h</td><td>h</td><td>h</td></tr>
<tr><td>e</td><td>2</td><td>ec</td><td>e</td><td>e</td><td>e</td></tr>
<tr><td>l</td><td>3</td><td>li</td><td>|</td><td>I</td><td>1</td></tr>
<tr><td>l</td><td>4</td><td>li</td><td>|</td><td>b</td><td>1</td></tr>
<tr><td>o</td><td>5</td><td>oc</td><td>oO</td><td></td><td>0</td></tr>
</table>

Length of fuzzy string (the value of the Length property):	5	5	4	5

All fuzzy strings which correspond to one word are grouped into a collection (the **FuzzyStringsCollection** object).

Properties

Name	Type	Description
CharacterVariants	**String**, read-only	Stores the recognition variants of a character in the specified position in the word.
Length	**Long**, read-only	Stores the length of the fuzzy string.

Input parameter

This object is the input parameter of the following methods and properties:

- **Add**, **Insert** methods and **Element** property of the **FuzzyStringsCollection** object.
- **CheckPrefix** method of the **IExternalDictionary** interface.

Output parameter

This object is the output parameter of the **Item** method and **Element** property of the **FuzzyStringsCollection** object.

See also

IExternalDictionary
FuzzyStringsCollection
Working with Dictionaries
Working with Properties

CharacterVariants Property of the FuzzyString Object

This property returns a reference to the string which contains the recognition variants of a character in the specified position of a word.

Visual Basic Syntax

```
Property CharacterVariants(Position As Long) As String
  read-only
```

C++ Syntax

```
HRESULT get_CharacterVariants(
   long  Position,
   BSTR* Result
);
```

Parameters

Position

[in] This variable contains the position of the character in the word.

Result

[out, retval] A pointer to the string variable which contains a string with the recognition variants.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

If the returned string contains the U+FFFD symbol, any symbol may be used in the specified position in the word.

See also

FuzzyString
Working with Properties

Text-Related Objects

Text object represents the recognized text. The **Text** object exposes a collection of paragraphs represented by the **Paragraph** objects. The **ParagraphParams** and **CharParams** objects comprise paragraph and single character properties respectively. Geometrical information on the recognized text lines is stored in the **ParagraphLine** object.

This section contains descriptions of the following text-related objects:

- **Text**
- **Paragraphs**
- **Paragraph**
- **ParagraphLines**
- **ParagraphLine**
- **ParagraphParams**
- **CharParams**
- **WordRecognitionVariants**
- **WordRecognitionVariant**
- **CharacterRecognitionVariants**
- **CharacterRecognitionVariant**
- **Words**
- **Word**
- **Hyperlink**
- **TabPositions**
- **TabPosition**
- **TextOrientation**
- **PlainText**

You can find additional information in the Working with Text section.

The text-related objects hierarchy

For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram.**

Text Object (IText Interface)

This object represents recognized text. The recognized text is a collection of paragraphs. Access to this collection is provided through the **Paragraphs** property. Besides, this object exposes properties for accessing different text attributes and methods allowing operations upon it, such as vertical and horizontal splitting, range removal etc. The **Text** object may exist either independently or be a subobject of some other object representing a unit of layout (text block, table cell etc.). A position in text is defined by the "coordinate pair" *(paragraph;symbol)*. There exists the so called "special position" or **theSpecialPos**, for which *paragraph=<the number of paragraphs>*, *symbol=0*.

Properties

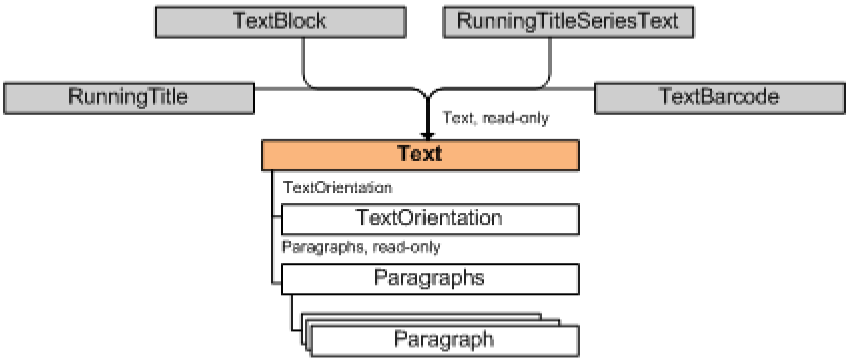
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
IsInverted	**Boolean**	Specifies if the colors of the whole text are inverted. This attribute is set to TRUE if the recognized text belongs to a block or table cell that also has the inverted attribute. It may

		be used in user interface to display the text with white font against the black background. **Note:** The property is available for writing only if, the **Text** object is received via the **ITextBlock::Text** property. Otherwise the property is read-only.
Paragraphs	**Paragraphs**, read-only	Provides access to the collection of paragraphs of the **Text** object. Every **Text** object, even an empty one, contains a valid subobject of the **Paragraphs** type. This object is a collection of the **Paragraph** objects and may not contain any elements. The **Paragraph** object represents a paragraph in the recognized text. It is through this object that the content of the recognized text may be obtained.
TextOrientation	**TextOrientation**	Stores the orientation of the text. It is used internally by the ABBYY FineReader Engine when exporting the recognized text. This property only matters after the recognition. If you want to specify the text orientation before the recognition, you must use the **RotationType** property of the **ImageProcessingParams** object. **Note:** The property returns a constant object. To change the text orientation, you must first receive an intermediate **TextOrientation** object with the help of the **IEngine::CreateTextOrientation** method, change the necessary parameters, and then assign this object to the property. The property is available for writing only if, the **Text** object is received via the **ITextBlock::Text** property. Otherwise the property is read-only.
TextRole	**TextRoleEnum**	Stores the role of the text.

Methods

Name	Description
AppendEmptyParagraph	Appends empty paragraph to the end of the current text.
GetRange	Returns a copy of the range of text.
Remove	Removes a range from the current text.
RemoveAll	Removes all paragraphs from the current text.

Related objects



Output parameter

This object is the output parameter of the **GetAsText** method of the **PageElement** object.

Input parameter

This object is the input parameter of the **InsertText** method of the **Paragraph** object.

See also

Working with Text
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

AppendEmptyParagraph Method of the Text Object

This method appends an empty paragraph at the end of the current text. Parameters of the new paragraph are initialized with default values.

Visual Basic Syntax

```
Method AppendEmptyParagraph()
```

C++ Syntax

```
HRESULT AppendEmptyParagraph();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Text

GetRange Method of the Text Object

This method allows you to get a copy of the range of text. You can insert the resulting text into the text of a paragraph (the **IParagraph::InsertText** method).

Visual Basic Syntax

```
Method GetRange(
  fromParagraph As Long,
  fromPos       As Long,
  toParagraph   As Long,
  toPos         As Long
) As Text
```

C++ Syntax

```
HRESULT GetRange(
  long    fromParagraph,
  long    fromPos,
  long    toParagraph,
  long    toPos,
  IText** text
);
```

Parameters

fromParagraph

[in] Variable specifying the index of the paragraph for the starting point of the range to be copied.

fromPos

[in] Variable defining the index of character in the starting paragraph, for the starting point of the range to be copied.

toParagraph

[in] Variable defining the index of the paragraph for the ending point of the range to be copied.

toPos

[in] Variable defining the index of character in the ending paragraph, for the ending point of the range to be copied. This character itself is not included in the copied text.

text

[out, retval] A pointer to the **IText*** pointer variable that receives the interface pointer of the **Text** object representing the range. *text* should not be NULL. **text* must be NULL, otherwise an error code is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The ending position of the range should be farther in the text than the starting one, otherwise an error code is returned. The symbol in *(fromParagraph;fromSymbol)* position is included in the range, while the character in *(toParagraph;toSymbol)* position is not included. To get a copy of the whole text, pass the *(0;0)* coordinates for the beginning of the range, and the **theSpecialPos** coordinates for the end of the range.

See also

Text

Remove Method of the Text Object

This method removes a range of text specified by the positions of paragraphs and symbols.

Visual Basic Syntax

```
Method Remove(
  fromParagraph As Long,
  fromSymbol    As Long,
  toParagraph   As Long,
  toSymbol      As Long
)
```

C++ Syntax

```
HRESULT Remove(
  long fromParagraph,
  long fromSymbol,
  long toParagraph,
  long toSymbol
);
```

Parameters

fromParagraph

[in] Variable specifying the index of the paragraph for the starting point of the range to be removed.

fromSymbol

[in] Variable specifying the index of character in the starting paragraph, for the starting point of the range to be removed.

toParagraph

[in] Variable specifying the index of the paragraph for the ending point of the range to be removed.

toSymbol

[in] Variable specifying the index of character in the ending paragraph, for the ending point of the range to be removed. This character itself is not removed from the text.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The ending position for the text to be removed should be farther in the text than the starting one, otherwise an error code is returned. The symbol in *(fromParagraph;fromSymbol)* position is removed from the text, while the character in *(toParagraph;toSymbol)* position is not removed.

See also

Text
IText::RemoveAll

RemoveAll Method of the Text Object

This method empties the collection of paragraphs of the **Text** object.

Visual Basic Syntax

```
Method RemoveAll()
```

C++ Syntax

```
HRESULT RemoveAll();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Text
IText::Remove

Paragraphs Object (IParagraphs Interface)

This object provides access to the collection of recognized text paragraphs. Besides standard collection methods and properties, it contains the **GetIndex** method that allows you to find paragraph index in the collection given a pointer to **Paragraph** object. The collection is accessible via the **Text** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

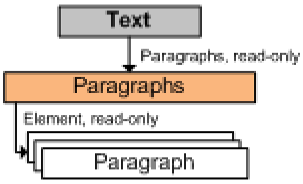
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**Paragraph**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
GetIndex	Finds the index of a paragraph in the collection of paragraphs.
Item	Provides access to a single element of the collection.

Related objects



See also

Text
Paragraph
Working with Text
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

GetIndex Method of the Paragraphs Object

This method finds the index of a paragraph in the collection of paragraphs. If there is no such paragraph in the collection, -1 is returned.

Visual Basic Syntax

```
Method Find(
  paragraph As Paragraph
) As Long
```

C++ Syntax

```
HRESULT Find(
  IParagraph* paragraph,
  long*       index
);
```

Parameters

paragraph

[in] This parameter refers to the interface of the **Paragraph** object.

index

[out] A pointer to **long** variable that receives the return value of this method. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraphs

Paragraph Object (IParagraph Interface)

This object exposes methods and properties for working with a single paragraph of the recognized text.

A paragraph in the ABBYY FineReader Engine object model is an elementary text unit. It is through this object that a user can get:

- the recognized text (use **Text** property for this purpose)
- different paragraph parameters (**ExtendedParams**, **ListParams**, **ParagraphStyle** properties)
- collections of paragraph lines and words (**Lines** and **Words** properties)
- a single character parameters (**GetCharParams**, **SetCharParams** and **GetDropCapCharParams** methods)

Note: The coordinates of the paragraph borders (**Left**, **Top**, **Right**, **Bottom** properties) are not available for the paragraphs of barcodes.

Properties

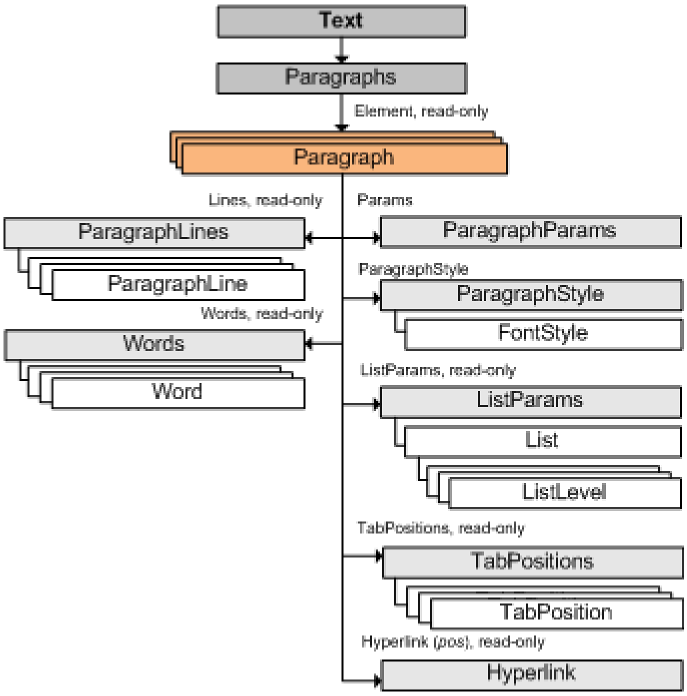
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Bookmark	**String**, read-only	Provides access to the bookmark by its index in the internal collection of the paragraph's bookmarks.
BookmarkCount	**Long**, read-only	Specifies the number of bookmarks in the paragraph.
Bottom	**Long**, read-only	Stores the coordinate of the bottom border of the paragraph as it is positioned on the image.
ColumnNumber	**Long**, read-only	Stores the number of the column to which the character in the position belongs.
DropCapCharsCount	**Long**	Provides access to the number of characters in the dropped capital of a paragraph. The first **DropCapCharsCount** symbols of the paragraph are assumed to be dropped capital. This property is not changed when paragraph is edited, so it may be greater than the length of the paragraph.
ExtendedParams	**ParagraphParams**	Provides access to the parameters of the **Paragraph** object exposed by the **ParagraphParams** object.
HasOverflowedHead	**Boolean**, read-only	Specifies if the paragraph is a part of another paragraph located on several pages, and has the beginning on another page. This property makes sense only after document synthesis with the **IDocumentStructureDetectionParams::DetectOverflowingParagraphs** property set to TRUE.
HasOverflowedTail	**Boolean**, read-only	Specifies if the paragraph is a part of another paragraph located on several pages, and has the end on another page. This property makes sense only after document synthesis with the **IDocumentStructureDetectionParams::DetectOverflowingParagraphs** property set to TRUE.
Hyperlink	**Hyperlink**, read-only	Returns a reference to the **Hyperlink** object which describes the hyperlink in the position. If there is no hyperlink, this property is set to 0.
InlinePictureID	**String**, read-only	Returns the ID of the **PageElement** object which describes the embedded picture in the position.
Left	**Long**, read-only	Stores the coordinate of the left border of the paragraph as it is positioned on the image.
Length	**Long**, read-only	This property contains the number of characters in paragraph. This value is the same as the number of characters in the string received through the **Text** property.

Lines	**ParagraphLines**, read-only	Provides access to the collection of the paragraph lines. The property returns a constant object.
ListParams	**ListParams**, read-only	Provides access to the parameters of the list to which the paragraph belongs. If the paragraph is not in the list, the **IListParams::List** property returns NULL.
ParagraphStyle	**ParagraphStyle**	Provides access to the parameters of the paragraph style. These parameters become accessible only after document synthesis. **Note:** The property returns a constant object. To change the paragraph style, you must first receive an intermediate **ParagraphStyle** object with the help of the **IGlobalStyleStorage::CreateParagraphStyle** method, change the necessary parameters, and then assign this object to the property.
Right	**Long**, read-only	Stores the coordinate of the right border of the paragraph as it is positioned on the image.
TabPositions	**TabPositions**, read-only	Returns a collection of tab leaders of the paragraph. If there is no tab leader, this property is set to 0.
Text	**String**, read-only	Provides access to the recognized text of the paragraph. It is through this property that you get the recognized text.
Top	**Long**, read-only	Stores the coordinate of the top border of the paragraph as it is positioned on the image.
Words	**Words**, read-only	Provides access to the collection of the paragraph words.

Methods

Name	Description
DeleteBookmark	Deletes the specified bookmark from the paragraph.
GetBookmarkRange	Detects the index of the initial character and the length of the string that forms the bookmark by its name.
GetCharParams	Provides access to parameters of a single character.
GetDropCapCharParams	Provides access to the parameters of a paragraph's dropped capital.
GetHyperlinkRange	Analyzes a single hyperlink character and detects the index of the initial character and the length of the string that forms the hyperlink.
GetWordRecognitionVariants	Returns a collection of variants of a word's recognition in the current position inside the text of a paragraph.
Insert	Inserts a string into the text of the paragraph.
InsertParagraphBreak	Divides the paragraph into two parts.
InsertText	Inserts the specified text into the text of the paragraph.
NextGroup	Finds the next character in the paragraph, which has the selected parameters different from such parameters of the character specified. This method can be used to find all bold or italic words in the paragraph, all uncertainly recognized characters, etc. **Note:** This method is obsolete and is intended to be removed in the next version of ABBYY FineReader Engine.
Range	Returns a substring from the text of the paragraph.
Remove	Deletes a range from the text of the paragraph.
SetBookmark	Sets a bookmark to a string within a paragraph.
SetCharParams	Sets parameters for a group of characters.
SetHyperlink	Sets a hyperlink to a string within a paragraph.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **Item** method of the **Paragraphs** object
- **GetAsParagraph** method of the **DocumentElement** object

Input parameter

This object is the input parameter of the **GetIndex** method of the **Paragraphs** object.

See also

Paragraphs
Working with Text
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

Bookmark Property of the Paragraph Object

This property provides access to the bookmark by its index in the internal collection of the paragraph's bookmarks.

Visual Basic Syntax

```
Property Bookmark(pos As Long) As String
  read-only
```

C++ Syntax

```
HRESULT get_Bookmark(
   long  pos,
   BSTR* result
);
```

Parameters

pos

[in] This variable contains the index of the bookmark in the internal collection of the paragraph's bookmarks. The value of this property must be in range from 0 to **IParagraph::BookmarkCount** -1.

result

[out, retval] A pointer to **BSTR** variable that receives the bookmark.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
Working with Properties

ColumnNumber Property of the Paragraph Object

This property provides access to the number of the column to which the character in the position belongs.

Visual Basic Syntax

```
Property ColumnNumber(pos As Long) As Long
  read-only
```

C++ Syntax

```
HRESULT get_ColumnNumber(
   long  pos,
   long* result
);
```

Parameters

pos

[in] This variable contains the index of a character inside the paragraph.

result

[out, retval] A pointer to **long** variable that receives the number of the column.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

The property returns -1, if the text, to which the paragraph belongs, has the TR_AbstractText role (**IText::TextRole** property).

See also

Paragraph
Working with Properties

Hyperlink Property of the Paragraph Object

This property returns a reference to the **Hyperlink** object which describes the hyperlink in the position. If there are no hyperlinks, this property is set to 0.

Visual Basic Syntax

```
Property Hyperlink(pos As Long) As Hyperlink
  read-only
```

C++ Syntax

```
HRESULT get_Hyperlink(
   long         pos,
   IHyperlink** result
);
```

Parameters

pos

[in] This variable contains the index of a character inside the paragraph.

result

[out, retval] A pointer to **IHyperlink*** pointer variable that receives the interface pointer of the **Hyperlink** output object. This object exposes properties of the hyperlink. If there is no hyperlink, this property is set to 0.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
Hyperlink
Working with Properties

InlinePictureID Property of the Paragraph Object

This property returns the ID of the **PageElement** object which describes the embedded picture in the position. If there is no embedded image in the specified position, an empty string is returned.

Note: You can receive positions of embedded pictures in the paragraph using the **IParagraph::NextGroup** method with the CFL_Picture constant as a mask.

You can then access the properties of an inline picture:

1. Use the **IParagraph::InlinePictureID** property to receive the ID of the **PageElement** object which describes the embedded image.
2. Find the corresponding **PageElement** object by its ID.
3. Receive its **TextPicture** object using the **GetAsPicture** method and work with its properties.

Visual Basic Syntax

```
Property InlineElementId(position As Long) As String
  read-only
```

C++ Syntax

```
HRESULT get_InlineElementId(
   long  position,
   BSTR* result
);
```

Parameters

position

[in] This variable contains the index of a character inside the paragraph.

result

[out, retval] A pointer to **BSTR** variable that receives the ID of the **PageElement** object which describes the embedded image in the position. If there is no embedded image in the specified position, an empty string is returned.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
Working with Properties

DeleteBookmark Method of the Paragraph Object

This method deletes the specified bookmark from the paragraph.

Visual Basic Syntax

```
Method DeleteBookmark(
  bookmark As String
)
```

C++ Syntax

```
HRESULT DeleteBookmark(
  BSTR bookmark
);
```

Parameters

bookmark

[in] This variable specifies the bookmark to be deleted.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph

GetBookmarkRange Method of the Paragraph Object

This method detects the index of the initial character and the length of the string that forms the bookmark by its name.

Visual Basic Syntax

```
Method GetBookmarkRange(
  bookmark As String,
  startPos As Long,
  count    As Long
)
```

C++ Syntax

```
HRESULT GetBookmarkRange(
  BSTR  bookmark,
  long* startPos,
  long* count
);
```

Parameters

bookmark

[in] The name of the bookmark.

startPos

[in, out] The index of the initial character of the bookmark.

count

[in, out] The length of the string that forms the bookmark.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph

GetCharParams Method of the Paragraph Object

This method provides access to parameters of a single character. A character is indexed with its position inside the text of paragraph. This index must be in the range from zero to the length of paragraph. The length of paragraph may be obtained from the **IParagraph::Length** property. When the length of paragraph is passed into this method, this property refers to the parameters that would have received a character if it was inserted at the end of paragraph.

Visual Basic Syntax

```
Method GetCharParams(
  pos    As Long,
  params As CharParams
)
```

C++ Syntax

```
HRESULT GetCharParams(
   long        pos,
   ICharParams* params
);
```

Parameters

pos

[in] This variable contains the index of the character inside the paragraph.

params

[in] This variable refers to a **CharParams** object. This object properties are initialized with values corresponding to parameters of the character. A valid object should be passed as this parameter.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
CharParams

See sample: RecognizedTextProcessing

GetDropCapCharParams Method of the Paragraph Object

This method provides access to parameters of a character with the specified position in a paragraph's dropped capital.

Visual Basic Syntax

```
Method GetDropCapCharParams(
   Pos    As Long
) As CharParams
```

C++ Syntax

```
HRESULT GetDropCapCharParams(
   long          Pos,
   ICharParams** params
);
```

Parameters

Pos

[in] This variable contains the index of the character inside the paragraph dropped capital.

params

[out] A pointer to the **ICharParams*** pointer variable that receives the interface pointer of the **CharParams** object representing the parameters of the dropped capital. *param* must not be NULL, otherwise an error code is returned.

Return Values

This function has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

Paragraph
CharParams

GetHyperlinkRange Method of the Paragraph Object

This method analyzes a single hyperlink character and detects the index of the initial character and the length of the string that forms the hyperlink. The analyzed character must have a non-null value of the **IParagraph::Hyperlink** property.

Visual Basic Syntax

```
Method GetHyperlinkRange(
  pos      As Long,
  startPos As Long,
  count    As Long
)
```

C++ Syntax

```
HRESULT GetHyperlinkRange(
  long  pos,
  long* startPos,
  long* count
);
```

Parameters

pos

[in] The index of the analyzed character. The analyzed character in this position must have a non-null value of the **IParagraph::Hyperlink** property.

startPos

[in, out] The index of the initial character of the hyperlink.

count

[in, out] The length of the string that forms the hyperlink.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph

GetWordRecognitionVariants Method of the Paragraph Object

This method returns a collection of variants of a word's recognition in the current position inside the text of a paragraph. This index must be in the range from zero to the length of the paragraph. The length of the paragraph may be obtained from the **IParagraph::Length** property. When the length of the paragraph is passed into this method, this property refers to the parameters that would have received the character if it were inserted at the end of the paragraph. The method returns zero for non-printable characters (spaces, carriage returns, etc.) and characters that were not recognized but added to the text during explicit editing. Zero is also returned if the text was recognized by one of the previous ABBYY FineReader Engine versions. If the **IRecognizerParams::SaveWordRecognitionVariants** property is set to FALSE the return collection contains one element, otherwise the collection contains no less than one element and the variants are ordered from the best to the worst.

Visual Basic Syntax

```
Method GetWordRecognitionVariants(
  pos As Long
) As WordRecognitionVariants
```

C++ Syntax

```
HRESULT GetWordRecognitionVariants(
  long                        pos,
  IWordRecognitionVariants** result
);
```

Parameters

pos

[in] This variable contains the index of the character inside the paragraph.

result

[out] A pointer to **IWordRecognitionVariants*** pointer variable that receives the interface pointer to the **WordRecognitionVariants** output object.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
ICharParams::WordRecognitionVariants
Voting API

Insert Method of the Paragraph Object

This method inserts a string into the text of the paragraph.

Visual Basic Syntax

```
Method Insert(
  pos    As Long,
  st     As String,
  params As CharParams
)
```

C++ Syntax

```
HRESULT Insert(
  long         pos,
  BSTR         st,
  ICharParams* params
);
```

Parameters

pos

[in] Position where the string is inserted. Must be not less than 0 and not greater than the length of paragraph.

st

[in] This string may contain the object replacement characters (Unicode 0xFFFC). The object replacement character denotes an embedded picture.

params

[in] This variable refers to the **CharParams** object that contains attributes for all characters of the newly inserted string. This parameter may be 0, in which case the default character parameters are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
CharParams

InsertParagraphBreak Method of the Paragraph Object

This method divides the paragraph into two parts.

Visual Basic Syntax

```
Method InsertParagraphBreak(
  position   As Long,
  charParams As CharParams
)
```

C++ Syntax

```
HRESULT InsertParagraphBreak(
  long         position,
  ICharParams* charParams
);
```

Parameters

position

[in] Position where the paragraph is to be divided. Must be not less than 0 and not greater than the length of the paragraph.

charParams

[in] This variable refers to the **CharParams** object that contains attributes for all characters of the newly created paragraph. This parameter may be 0, in which case the default character parameters are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
CharParams

InsertText Method of the Paragraph Object

This method inserts the specified text into the text of the paragraph.

Visual Basic Syntax

```
Method InsertText(
  pos  As Long,
  text As Text
)
```

C++ Syntax

```
HRESULT InsertText(
  long   pos,
  IText* text
);
```

Parameters

pos

[in] Position where the text is inserted. Must be not less than 0 and not greater than the length of paragraph.

text

[in] This variable must refer to a valid **Text** object that contains the newly inserted text.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph

NextGroup Method of the Paragraph Object

This method returns the index of the next character in the paragraph, which has the selected parameters different from such parameters of the character specified.

For example, if you set the value of the *styleFlagMask* parameter to SF_Bold (which means that the **ICharParams::IsBold** property should be taken into account) and the character with the *position* index is not bold, the method will return the index of the next bold character, and vice versa if the character with the *position* index is bold, the method will return the index of the next character, which is not bold.

Visual Basic Syntax

```
Method NextGroup(
  position      As Long,
  charFlagMask  As Long,
  styleFlagMask As Long
) As Long
```

C++ Syntax

```
HRESULT NextGroup(
  long  position,
  long  charFlagMask,
  long  styleFlagMask
  long* result
);
```

Parameters

position

[in] This variable contains the index of the character, which defines parameters of the search for the next character.

charFlagMask

[in] This variable contains any OR combination of the **CFL_** prefixed flags. It defines what character parameters are taken into account when searching for the next character.

styleFlagMask

[in] This variable contains any OR combination of the **StyleParamsEnum** constants. It defines what style parameters are taken into account when searching for the next character.

result

[out, retval] A pointer to a **long** variable that receives the position of the next character, which has the selected parameters different from such parameters of the character with the *position* index. If the next character is not found, the length of the paragraph is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- This method can be used to find all bold or italic words in the paragraph, all uncertainly recognized characters, etc.
- This method is intended to be removed in the next version of ABBYY FineReader Engine.

See also

Paragraph
CFL_flags

CFL_ prefixed flags

The **CFL_** prefixed flags are used as a mask in some methods of the **Paragraph** object. The mask is an OR combination of these flags' values and define what properties of the **CharParams** object should be taken into account in these methods. The CFL_Picture is a special constant for inline pictures in a paragraph.

For the **IParagraph::SetCharParams** method the constants define what character properties should be set, and for the **IParagraph::NextGroup** method they define parameters separating a group of symbols.

```
module CharacterFlags
{
  const long CFL_Subscript    = 0x00000001;
  const long CFL_Superscript  = 0x00000002;
  const long CFL_Suspicious   = 0x00000100;
  const long CFL_Proofed      = 0x00000800;
  const long CFL_LanguageID   = 0x00010000;
  const long CFL_LanguageName = 0x00020000;
  const long CFL_Picture      = 0x00040000;
};
```

Elements

Flag name	Description
CFL_Subscript	The **ICharParams::IsSubscript** property should be taken into account.
CFL_Superscript	The **ICharParams::IsSuperscript** property should be taken into account.
CFL_Suspicious	The **ICharParams::IsSuspicious** property should be taken into account.
CFL_Proofed	The **ICharParams::IsProofed** property should be taken into account.

CFL_LanguageID	The **ICharParams::LanguageId** property should be taken into account.
CFL_LanguageName	The **ICharParams::LanguageName** property should be taken into account.
CFL_Picture	The **IParagraph::InlinePictureID** property should be taken into account.

See also

IParagraph::SetCharParams
IParagraph::NextGroup
CharParams

Range Method of the Paragraph Object

This method returns a substring from the text of the paragraph.

Visual Basic Syntax

```
Method Range(
  fromPos As Long,
  toPos   As Long
) As String
```

C++ Syntax

```
HRESULT Range(
  long fromPos,
  long toPos,
  BSTR*      st
);
```

Parameters

fromPos

[in] Position where the substring is started. Must be not less than 0 and not greater than the length of paragraph.

toPos

[in] Position where the substring is ended. Must be not less than variable *fromPos* and 0 and not greater than the length of paragraph.

st

[out] A pointer to **BSTR** variable that receives the substring that is started at *fromPos* position and is ended to *toPos* position. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
CharParams

Remove Method of the Paragraph Object

This method deletes a range from the text of the paragraph.

Visual Basic Syntax

```
Method Remove(
  fromPos As Long,
  toPos   As Long
)
```

C++ Syntax

```
HRESULT Remove(
  long fromPos,
  long toPos
);
```

Parameters

fromPos

[in] Position where the range is started. Must be not less than 0 and not greater than the length of paragraph.

toPos

[in] Position where the range is ended. Must be not less than variable *fromPos* and 0 and not greater than the length of paragraph. By default this variable is set to -1. If this variable is not set or is set to -1 one character will be removed only.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
CharParams

SetBookmark Method of the Paragraph Object

This method sets a bookmark to a string within a paragraph.

Visual Basic Syntax

```
Method SetBookmark(
  pos      As Long,
  count    As Long,
  bookmark As String
)
```

C++ Syntax

```
HRESULT SetBookmark(
  long pos,
  long count,
  BSTR bookmark
);
```

Parameters

pos

[in] The index of the initial character of the bookmark.

count

[in] The length of the string that forms the bookmark.

bookmark

[in] This variable specifies the bookmark to be set.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph

SetCharParams Method of the Paragraph Object

This method sets parameters for a group of characters.

Visual Basic Syntax

```
Method SetCharParams(
  position      As Long,
  count         As Long,
  params        As CharParams,
  charFlagMask  As Long,
  styleFlagMask As Long
)
```

C++ Syntax

```
HRESULT SetCharParams(
  long         position,
  long         count,
  ICharParams* params,
  long         charFlagMask,
  long         styleFlagMask
);
```

Parameters

position

[in] Position of character in paragraph that starts the group of characters for which the parameters are set. It should be in the range from 0 to the length of paragraph.

count

[in] A number of characters for which the parameters are set. It should be no less than 0 and meet the following requirement: *position + count <= paragraph length + 1*

params

[in] This variable refers to the **CharParams** object that contains the new parameters for the group of characters. It must refer to a valid object.

charFlagMask

[in] This variable may contain any OR combination of the CFL_prefixed flags. It specifies what character parameters are to be copied from the *params* object.

styleFlagMask

[in] This variable may contain any OR combination of the **StyleParamsEnum** constants. It specifies what style parameters are to be copied from the *params* object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph
CFL_prefixed flags
CharParams

SetHyperlink Method of the Paragraph Object

This method sets a hyperlink to a string within a paragraph.

Visual Basic Syntax

```
Method SetHyperlink(
  pos       As Long,
  count     As Long,
  hyperlink As Hyperlink
)
```

C++ Syntax

```
HRESULT SetHyperlink(
  long        pos,
  long        count,
  IHyperlink* hyperlink
);
```

Parameters

pos

[in] The index of the initial character of the hyperlink.

count

[in] The length of the string that forms the hyperlink.

hyperlink

[in] This variable specifies the **Hyperlink** to be set.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Paragraph

ParagraphLines Object (IParagraphLines Interface)

This object represents a collection of paragraph lines. It contains methods for getting the number of paragraph lines in collection, accessing a single element of this collection and iterating through the elements of the collection. The collection is accessible via the **Paragraph** object.

Important! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	ParagraphLine, read-only	Provides access to a single element of the collection.

Methods

Name	**Description**
Item	Provides access to a single element of the collection.

Output parameter

This object is the output parameter of the **Lines** property of the **Paragraph** object.

See also

Paragraph
ParagraphLine
Working with Text
Working with Properties

ParagraphLine Object (IParagraphLine Interface)

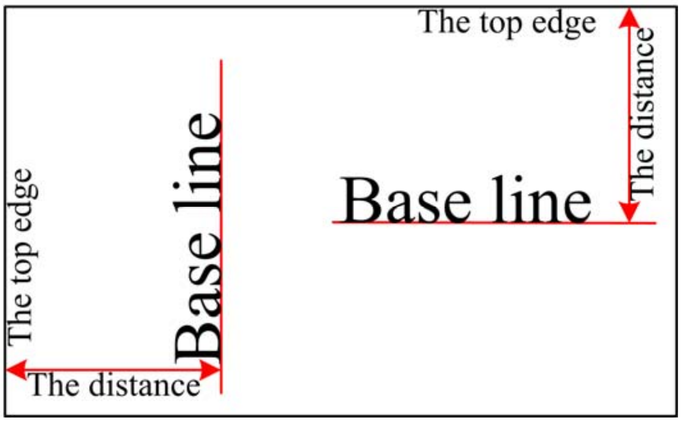
This object represents a single line in the paragraph of a recognized text. Its properties provide access to the line's geometrical attributes and allow you to find out where the line starts and where it ends in terms of characters.

The rectangle occupied by the line is defined by the **Left, Top, Right, Bottom** properties. In case it is undefined, all the four properties contain 0.
Note: The coordinates of the rectangle are not available for the paragraphs of barcodes.

This object is an element of a collection of paragraph lines (**ParagraphLines** object).

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
BaseLine	**Long**, read-only	Contains the distance from the base line to the top edge of the page. The base line is the line on which the characters are located. The top edge of the page is determined by the characters orientation (as shown in the figure below).

		 The top edge The distance Base line The top edge Base line The distance It may have undefined value that corresponds to **INT_MAX** or 0x7FFFFFFF.
Bottom	**Long**, read-only	Contains the coordinate of the bottom border of a surrounding rectangle. The surrounding rectangle is a minimal rectangle containing all the characters of the line. If this rectangle is undefined, this property contains 0. It is given "as is" regardless of the text orientation.
CharactersCount	**Long**, read-only	Contains the number of characters in the current paragraph line.
FirstCharIndex	**Long**, read-only	Contains the index of the first character of the current line. The index of the character is a sequence number of the character within the recognized text of the paragraph. The recognized text of the line is stored in the **Text** property of the **Paragraph** object.
Left	**Long**, read-only	Contains the coordinate of the left border of a surrounding rectangle. The surrounding rectangle is a minimal rectangle containing all the characters of the line. If this rectangle is undefined, this property contains 0. It is given "as is" regardless of the text orientation.
Right	**Long**, read-only	Contains the coordinate of the right border of a surrounding rectangle. The surrounding rectangle is a minimal rectangle containing all the characters of the line. If this rectangle is undefined, this property contains 0. It is given "as is" regardless of the text orientation.
Top	**Long**, read-only	Contains the coordinate of the top border of a surrounding rectangle. The surrounding rectangle is a minimal rectangle containing all the characters of the line. If this rectangle is undefined, this property contains 0. It is given "as is" regardless of the text orientation.

Output parameter

This object is the output parameter of the **Item** method and **Element** property of the **ParagraphLines** object.

See also

ParagraphLines
Working with Text
Working with Properties

ParagraphParams Object (IParagraphParams Interface)

This object exposes extended properties of a single paragraph.

Important! If you wish to work with the properties of a single paragraph, you must first call any of the functions that perform synthesis (e.g. the **Process** or **Synthesize** method of the **FRDocument** object), as these properties become meaningful only after synthesis.

The **ParagraphParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColor	**Long**	Stores the background color of the text. By default the background color

		is white or RGB(255,255,255). 📝**Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color white equals 16777215.
FirstLineIndent	**Long**	Contains and allows you to set the indent of the first line of the paragraph from the left border of the paragraph. By default this value is 0. The value of this property affects the results of export.
IsRightToLeft	**Boolean**	Indicates if the paragraph has right-to-left writing direction (like for Hebrew). By default the value of this property is FALSE.
LeftIndent	**Long**	Contains and allows you to set the left indent for the paragraph. The value of indent is the distance in pixels from the left border of the block to the left border of the paragraph. By default this value is 0. The value of this property affects the results of export.
LineSpacing	**Long**	Contains and allows you to set the line spacing for the paragraph. The value of line spacing is the average distance in pixels between base lines of paragraph strings. By default this value is 0. The zero value of this property means that the line spacing is undefined or does not have a sense (for example for a text in barcode block). If the value of this property is zero, it is ignored during recognized text export.
ParagraphAlignment	**ParagraphAlignmentEnum**	Stores and allows you to change the horizontal paragraph alignment. By default this value is PA_Left.
RightIndent	**Long**	Contains and allows you to set the right indent for the paragraph. The value of indent is the distance in pixels from the right border of the block to the right border of the paragraph. By default this value is 0. The value of this property affects the results of export.
SpaceAfter	**Long**	Contains and allows you to set the value of space after the paragraph. The space after the paragraph is the distance in pixels from the bottom border of block or top border of the next paragraph to the bottom border of the paragraph itself. By default this value is 0. The value of this property affects the results of export.
SpaceBefore	**Long**	Contains and allows you to set the value of space before the paragraph. The space before the paragraph is the distance in pixels from the top border of block or bottom border of previous paragraph to the top border of the paragraph itself. By default this value is 0. The value of this property affects the results of export.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the following methods and properties:

- **CreateParagraphParams** method of the **Engine** object
- **Params** property of the **ParagraphStyle** object
- **Params** property of the **Paragraph** object

Input parameter

This object is the output parameter of the **Params** property of the **Paragraph** object

See also

Paragraph
ParagraphStyle
Working with Properties

CharParams Object (ICharParams Interface)

This object allows you to access different parameters of a single character in recognized text: its formatting, rectangle on the image, recognition language, and hypotheses of recognition. All the Boolean properties of a newly created object of this type are set to FALSE.

Important! If you wish to work with the parameters of a certain character in the recognized text, you must first call any of the functions that perform synthesis (e.g. the **Process** or **Synthesize** method of the **FRDocument** object), as these parameters become meaningful only after synthesis.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
BaseLine	**Long**	Sets the shift of a character from the base line of the string in pixels. The base line of the string is defined by the **IParagraphLine::BaseLine** property. This property is mainly used for the pictures embedded in text.
Bottom	**Long**, read-only	Stores the coordinate of the bottom border of the character rectangle. This rectangle is defined on image, not accounting for the text orientation. It may be undefined in which case all four of its coordinate are zeros. This property cannot be changed directly but through the **SetRect** method.
CharacterRecognitionVariantIndex	**Long**, read-only	Stores the index of the selected variant of character recognition in the **CharacterRecognitionVariants** collection.
CharacterRecognitionVariants	CharacterRecognitionVariants, read-only	Returns a collection of variants of character recognition. The property contains zero for non-printable characters (spaces, carriage returns, etc.) and characters that were not recognized but added to the text during explicit editing. Zero is also returned if the text was recognized by one of the previous ABBYY FineReader Engine versions. If the **IRecognizerParams::SaveCharacterRecognitionVariants** property is set to FALSE the return collection contains one element, otherwise the collection contains no less than one element and the variants are ordered from the best to the worst.
CharacterRegion	**Region**	Specifies the character exact region. The region may not be rectangular and initially is contained in the rectangle defined by the **Left**, **Top**, **Right**, and **Bottom** properties. If you change the character exact region, the **Left**, **Top**, **Right**, and **Bottom** properties are not changed. The property is only available after recognition with the **IRecognizerParams::SaveCharacterRegions** property set to TRUE. **Note:** The property returns a constant object. To change the character exact region, you must first receive an intermediate **Region** object with the help of the **IEngine::CreateRegion** method, change the necessary parameters, and then assign this object to the property.
Color	**Long**	Sets the RGB value of the color for the symbol. Its background color is defined for the whole paragraph by the **IParagraphParams::BackgroundColor** property. By default

		the text color is black or RGB(0,0,0). **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color black equals 0.
FontName	**String**, read-only	Stores the name of the font for a character. By default this value is "Times New Roman". This property cannot be changed directly but via the **SetFont** method.
FontSize	**Long**	Specifies the height of the font of the character in twips. Twip is 1/20 of point, and point is 1/72". Default value of this property corresponds to 10 points or 200 twips.
FontStyle	**FontStyle**	Provides access to the font style of the character.
FontType	FontTypeEnum, read-only	Stores the type of the font for a character. By default this value is FT_Serif. This property cannot be changed directly but via the **SetFont** method.
HorizontalScale	**Long**	Stores horizontal scaling for a character in 1/1000. Default for this property is 1000, which corresponds to no scaling.
IsBold	**Boolean**	Specifies whether the character is bold.
IsItalic	**Boolean**	Specifies whether the character is italic.
IsProofed	**Boolean**	Specifies whether a spell-checking was performed upon this character. It is not used or set internally by ABBYY FineReader Engine and just provides you a framework for spelling.
IsSmallCaps	**Boolean**	Specifies whether the character has "small caps" style. This means that the small characters are displayed as small capitals.
IsStrikeout	**Boolean**	Specifies whether the character is strikeout.
IsSubscript	**Boolean**	Specifies whether the character is subscript. It cannot be set to TRUE simultaneously with the **IsSuperscript** property, as this will lead to errors during recognized text export.
IsSuperscript	**Boolean**	Specifies whether the character is superscript. It cannot be set to TRUE simultaneously with the **IsSubscript** property, as this will lead to errors during recognized text export.
IsSuspicious	**Boolean**	This property set to TRUE means that the character was recognized uncertainly. More detailed information about recognition confidence may be obtained for the certain recognition variant from the **CharacterRecognitionVariant** object. In ABBYY FineReader uncertainly recognized characters are highlighted with background color in the recognized text. See also What is the difference between the CharConfidence and the IsSuspicious properties?
IsUnderlined	**Boolean**	Specifies whether the character is underlined.
IsWordStart	**Boolean**	Specifies whether the character is the first character in a word.
LanguageId	**LanguageIdEnum**	Specifies the ID of the language of the character. To convert it to Win32 LCID use the **IEngine::ConvertLanguageIdToLCID** method. By default this property is initialized with system default language ID.
LanguageName	**String**	Stores and allows you to set internal name of the language for a character. **Note:** If one base recognition language corresponds to one recognized word, the **LanguageName** property for each character in this word is set to the internal name of the base language after recognition. If several base recognition languages correspond to one word (e.g. for bilingual

		compound words), the **LanguageName** property for the characters in this word is empty. While the **LanguageId** property contains the identifier of the base language no matter what the recognized word is.
Left	**Long**, read-only	Stores the coordinate of the left border of the character rectangle. This rectangle is defined on image, not accounting for the text orientation. It may be undefined in which case all four of its coordinate are zeros. This property cannot be changed directly but through the **SetRect** method.
Right	**Long**, read-only	Stores the coordinate of the right border of the character rectangle. This rectangle is defined on image, not accounting for the text orientation. It may be undefined in which case all four of its coordinate are zeros. This property cannot be changed directly but through the **SetRect** method.
SelectedCharacterRecognitionVariant	CharacterRecognitionVariant, read-only	Stores the selected variant of character recognition. It is the element with the index **CharacterRecognitionVariantIndex** in the collection of character recognition variants (the **CharacterRecognitionVariants** property).
Spacing	**Long**	Specifies additional spacing between characters in twips. Twip is 1/20 of point, and point is 1/72". Default value of this property is 0.
Top	**Long**, read-only	Stores the coordinate of the top border of the character rectangle. This rectangle is defined on image, not accounting for the text orientation. It may be undefined in which case all four of its coordinate are zeros. This property cannot be changed directly but through the **SetRect** method.
WordRecognitionVariants	WordRecognitionVariants, read-only	Returns a collection of recognition variants for the word to which the character belongs. The property contains zero for non-printable characters (spaces, carriage returns, etc.) and characters that were not recognized but added to the text during explicit editing. Zero is also returned if the text was recognized by one of the previous ABBYY FineReader Engine versions. If the **IRecognizerParams::SaveWordRecognitionVariants** property is set to FALSE the return collection contains one element, otherwise the collection contains no less than one element and the variants are ordered from the best to the worst.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with the values of similar properties of another object.
SetFont	Sets the new font for the symbol.
SetRect	Sets the new rectangle for the symbol.

Output parameter

This object is the output parameter of the following methods:

- **CreateCharParams** of the **Engine** object,
- **GetDropCapCharParams** of the **Paragraph** object.

Input parameter

This object is the input parameter of the following methods:

- **GetCharParams**, **SetCharParams**, **Insert**, **InsertParagraphBreak** of the **Paragraph** object,

- **GetCharParams** of the **WordRecognitionVariant** object.

See also

Paragraph
Working with Text
Working with Properties

See sample: RecognizedTextProcessing

SetFont Method of the CharParams Object

This method allows you to set a new font for the symbol. It simultaneously specifies the name of the font and its type, as these are interdependent parameters. This method affects the **ICharParams::FontName** and **ICharParams::FontType** properties.

Visual Basic Syntax

```
Method SetFont(
  fontName As String,
  fontType As FontTypeEnum
)
```

C++ Syntax

```
HRESULT SetFont(
  BSTR         fontName,
  FontTypeEnum fontType
);
```

Parameters

fontName

[in] This variable specifies the name of the new font.

fontType

[in] This variable specifies the type of the new font. It may be set to one of the constants from the **FontTypeEnum** enumeration.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

CharParams
FontTypeEnum

SetRect Method of the CharParams Object

This method allows you to set the rectangle for the symbol. It affects its **Left**, **Top**, **Right**, **Bottom** properties, and does not affect the **CharacterRegion** property. The rectangle is defined in pixel coordinates on image.

Visual Basic Syntax

```
Method SetRect(
  left   As Long,
  top    As Long,
  right  As Long,
  bottom As Long
)
```

C++ Syntax

```
HRESULT SetRect(
  long left,
  long top,
  long right,
  long bottom
);
```

Parameters

left

[in] Coordinate for the left border of the rectangle.

top

[in] Coordinate for the top border of the rectangle.

right

[in] Coordinate for the right border of the rectangle.

bottom

[in] Coordinate for the bottom border of the rectangle.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

CharParams

WordRecognitionVariants Object (IWordRecognitionVariants Interface)

This object is a collection of variants of a word's recognition. The collection contains recognition variants ranked from best to worst.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	WordRecognitionVariant, read-only	Provides access to a single element of the collection.

Methods

Name	**Description**
Item	Provides access to a single element of the collection.

Output parameter

This object is the output parameter of the following methods and properties:

- **GetWordRecognitionVariants** method of the **Paragraph** object
- **GetRecognitionVariants** method of the **Word** object
- **WordRecognitionVariants** property of the **CharParams** object

See also

WordRecognitionVariant
CharacterRecognitionVariants
Voting API
Working with Properties

WordRecognitionVariant Object (IWordRecognitionVariant Interface)

This object represents a variant of a word recognition. It is an element of **WordRecognitionVariants** collection.

Properties

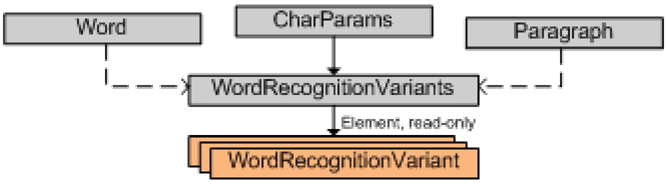
Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.

IsWordFromDictionary	**Boolean**, read-only	Specifies whether the word was found in the dictionary.
MeanStrokeWidth	**Double**, read-only	Returns the mean width of stroke in the RLE representation of a word image, expressed in pixels.
ModelType	**WordModelTypeEnum**, read-only	Returns type of model used while composing the word.
Text	**String**, read-only	Returns the word.
WordConfidence	**Long**, read-only	Stores the value of word confidence. It is in the range from 0 to 100. It represents an estimate of recognition confidence of the word in percentage points. The greater its value, the greater the confidence. To calculate confidence more accurately, set the **IRecognizerParams::ExactConfidenceCalculation** property to TRUE.

Methods

Name	Description
GetCharParams	Provides access to parameters of a single character.

Related objects



Output parameter

This object is the output parameter of the **Item** method of the **WordRecognitionVariants** object.

See also

WordRecognitionVariants
CharacterRecognitionVariant
Voting API
Working with Properties

GetCharParams Method of the WordRecognitionVariant Object

This method provides access to parameters of a single character. A character is indexed with its position inside the word. This index must be in the range from zero to the length of word.

Visual Basic Syntax

```
Method GetCharParams(
   pos As Long,
   params As CharParams
)
```

C++ Syntax

```
HRESULT GetCharParams(
   long         pos,
   ICharParams* params
);
```

Parameters

pos

[in] This variable contains the index of the character inside the word.

params

[in] This variable refers to a **CharParams** object. This object's properties are initialized with values corresponding to parameters of the character. A valid object should be passed as this parameter.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

WordRecognitionVariant
CharParams
Voting API

CharacterRecognitionVariants Object (ICharacterRecognitionVariants Interface)

This object is a collection of recognition variants for a single character. The collection contains recognition variants ranked from the best to the worst. You can select the best recognition variant for a character by voting between the variants. See for details Using Voting API. The collection is accessible via the **CharParams** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

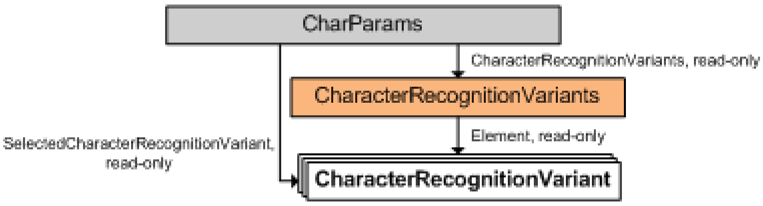
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	CharacterRecognitionVariant, read-only	Provides access to a single element of the collection.

Methods

Name	**Description**
Item	Provides access to a single element of the collection.

Related objects


CharParams
CharacterRecognitionVariants, read-only
CharacterRecognitionVariants
Element, read-only
SelectedCharacterRecognitionVariant, read-only
CharacterRecognitionVariant

See also

CharParams
CharacterRecognitionVariant
WordRecognitionVariants
Using Voting API
Working with Properties

CharacterRecognitionVariant Object (ICharacterRecognitionVariant Interface)

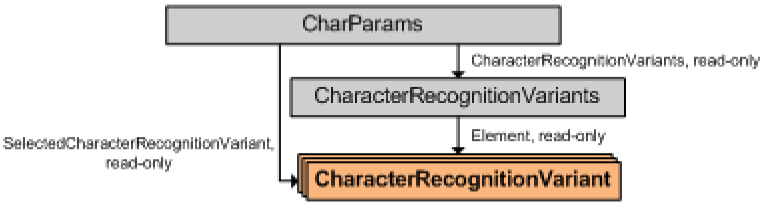
This object represents the variant of a character recognition. The object provides access to the variant itself and its confidence, probability that a character is written with a Serif font, and the information whether the character is superscript or subscript. It is an element of the **CharacterRecognitionVariants** collection. You can select the best recognition variant for a character by voting between the variants. See for details Using Voting API.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
Character	**String**, read-only	Returns the variant of a character recognition.
CharConfidence	**Long**, read-only	Stores the value of character confidence. It is in the range from 0 to 100, and 255 corresponds to the fact that confidence is undefined. It represents an estimate of recognition confidence of a character in percentage points. The greater its value, the greater the confidence. Character

		confidence can be undefined, for example, for characters which were added during text editing. In this case, the value of this property is -1. To calculate character confidence more accurately, set the **IRecognizerParams::ExactConfidenceCalculation** property to TRUE. See also **What is the difference between the CharConfidence and the IsSuspicious properties?**
IsSubscript	**Boolean**, read-only	Specifies whether the character is subscript.
IsSuperscript	**Boolean**, read-only	Specifies whether the character is superscript.
SerifProbability	**Long**, read-only	The value of this property specifies probability that a character is written with a Serif font. It is in the range from 0 to 100, and 255 corresponds to the fact that this probability is undefined.

Related objects


CharParams
CharacterRecognitionVariants, read-only
CharacterRecognitionVariants
Element, read-only
SelectedCharacterRecognitionVariant, read-only
CharacterRecognitionVariant

Output parameter

This object is the output parameter of the **Item** method of the **CharacterRecognitionVariants** object.

See also

CharacterRecognitionVariants
WordRecognitionVariant
Using Voting API
Working with Properties

Words Object (IWords Interface)

This object represents a collection of words. The collection is accessible via the **Paragraph** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**Word**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Item	Provides access to a single element of the collection.

Output parameter

This object is the output parameter of the **Words** property of the **Paragraph** object.

See also

Word
Paragraph
Working with Properties

See samples: RecognizedTextProcessing, CustomLanguage

Word Object (IWord Interface)

This object represents a word. It is an element of the collection of words (**Words** object).

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
FirstSymbolPosition	**Long**, read-only	Returns the index of the first character in the word. The index is the character position in the paragraph, it may be in the range of 0 to the value of the IParagraph::Length property minus 1.
IsWordFromDictionary	**Boolean**, read-only	Specifies whether the word was found in the dictionary.
ModelType	WordModelTypeEnum, read-only	Returns type of model used while composing the word.
Text	**String**, read-only	Returns the word.

Methods

Name	Description
GetRecognitionVariants	Returns a collection of variants of a word's recognition.

Output parameter

This object is the output parameter of the **Item** method and **Element** property of the **Words** object.

See also

Words
Working with Properties

See sample: RecognizedTextProcessing

GetRecognitionVariants Method of the Word Object

This method returns a collection of variants of a word recognition. The method returns zero for non-printable characters (spaces, carriage returns, etc.) and characters that were not recognized but added to the text during explicit editing. Zero is also returned if the text was recognized by one of the previous ABBYY FineReader Engine versions. If the **IRecognizerParams::SaveWordRecognitionVariants** property is set to FALSE the return collection contains one element, otherwise the collection contains no less than one element and the variants are ordered from the best to the worst.

Visual Basic Syntax

```
Method GetRecognitionVariants(
) As WordRecognitionVariants
```

C++ Syntax

```
HRESULT GetRecognitionVariants(
  IWordRecognitionVariants** result
);
```

Parameters

result

[out] A pointer to **IWordRecognitionVariants*** pointer variable that receives the interface pointer to the **WordRecognitionVariants** output object.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Word

Hyperlink Object (IHyperlink Interface)

This object exposes method and properties of a hyperlink.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Scheme	**HyperlinkSchemeEnum**	Stores the hyperlink type which is detected automatically when the **Target** property is assigned a hyperlink address. If the value of the **IHyperlink::Scheme** property is HS_Unknown, the type of the hyperlink assigned to the **Target** property will be detected automatically.
Target	**String**	Stores the hyperlink address. If the link is local (i.e. to a text fragment in the same **Text** object), this property must be assigned the same text as the **IParagraph::SetBookmark** method, and the **Scheme** property must be specified for the HS_Local value.

Methods

Name	Description
ParseTarget	Brings the **Target** property to the canonical form, according to the types described in **HyperlinkSchemeEnum**.

Input parameter

This object is the input parameter of the **IParagraph::SetHyperlink** method.

Output parameter

This object is the output parameter of the following methods and properties:

- **CreateHyperlink** method of the **Engine** object
- **Hyperlink** property of the **Paragraph** object

See also

Working with Text
Working with Properties

ParseTarget Method of the Hyperlink Object

This method brings the **IHyperlink::Target** property to the canonical form, according to the types described in **HyperlinkSchemeEnum**. In particular, for all non-local hyperlinks, this method substitutes escape characters (white space, backslash, etc.) with corresponding escape sequences. Besides, for example, the e-mail address "engine_support@abbyy.com" is changed to "mailto:engine_support@abbyy.com", or the web-site address "www.abbyy.com" is changed to "http://www.abbyy.com".

Visual Basic Syntax

```
Method ParseTarget() As HyperlinkSchemeEnum
```

C++ Syntax

```
HRESULT ParseTarget(
  HyperlinkSchemeEnum* result
);
```

Parameters

result

[out] Refer to the **HyperlinkSchemeEnum** denoting the type of hyperlink.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Hyperlink

TabPositions Object (ITabPositions Interface)

This object provides access to all the tab stops in a single paragraph. It allows you to access parameters of a single tab stop, add a new tab stop and remove tab stops. The **TabPositions** object is accessed via the **IParagraph::TabPositions** property.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

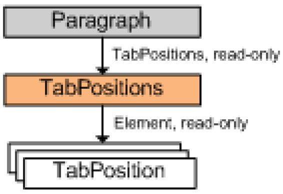
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**TabPosition**, read-only	Provides access to a single element of the collection. The property returns a constant object.

Methods

Name	Description
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
CreateTabPosition	Creates the **TabPosition** object.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



See also

TabPosition
Working with Properties

CreateTabPosition Method of the TabPositions Object

This method creates the **TabPosition** object. The newly created object has default values.

Visual Basic Syntax

```
Method CreateTabPosition(
) As TabPosition
```

C++ Syntax

```
HRESULT CreateTabPosition(
  ITabPosition** result
);
```

Parameters

result

[out] A pointer to **ITabPosition*** pointer variable that receives the interface pointer of the created object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TabPositions

TabPosition Object (ITabPosition Interface)

This object provides access to a single tab stop: the tab symbol, its alignment, and position in the paragraph.

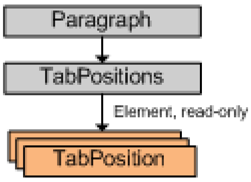
Properties

Name	Type	Description
Alignment	**ParagraphTabAlignmentEnum**	Specifies the alignment of the tab stop. By default the value of this property is PTA_Left.
Application	**Engine**, read-only	Returns the **Engine** object.
Position	**Long**	Specifies the position of the tab stop, counted from the left border of the paragraph in hundredth parts of point. By default the value of this property is -1, which means that the position is undefined.
TabLeaderType	**TabLeaderTypeEnum**	Specifies the type of tab symbol. The value of this property is TLT_None by default.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **CreateTabPosition** method of the **Engine** object
- **Item** method of the **TabPositions** object

Input parameter

This object is the input parameter of the **Add** method of the **TabPositions** object.

See also

TabPositions
Working with Properties

TextOrientation Object (ITextOrientation Interface)

This object represents a text orientation.

Properties

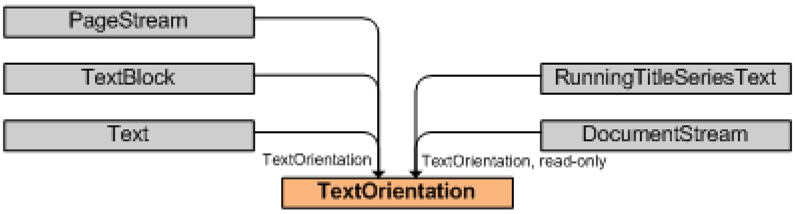
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
IsVerticalMirrored	**Boolean**	Specifies if the image is mirrored around the vertical axis during preprocessing. The recognized text receives this attribute if it belongs to a block or table cell that is also mirrored. This property is FALSE by default.
ReadingType	**ReadingTypeEnum**	Specifies if the text on the page is divided into several columns or it is written in one column. This property is TRT_Unknown by default.

RotationType	**RotationTypeEnum**	Specifies the orientation of a text. This property is RT_NoRotation by default, which means that the orientation is normal.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
IsEqualTo	Checks if the text orientation is equal to the specified orientation.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **CreateTextOrientation** of the **Engine** object
- **DetectOrientation** of the **FRPage** object

See also

DocumentStream,
TextBlock
Working with Properties

IsEqualTo Method of the TextOrientation Object

This method allows you to check if the orientation is equal to the specified orientation.

Visual Basic Syntax

```
Method IsEqualTo(
  orientation As TextOrientation
) As Boolean
```

C++ Syntax

```
HRESULT IsEqualTo(
  ITextOrientation* orientation,
  VARIANT_BOOL*     result
);
```

Parameters

orientation

[in] This variable refers to the **TextOrientation** object that is to be compared with the current object.

result

[out, retval] This variable contains the result of comparison. It returns TRUE, if the orientations are equal.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextOrientation

PlainText Object (IPlainText Interface)

This object represents recognized text in a special "plain text" format. It provides information only about the recognized text symbols, their recognition confidence and positions as relative to the source image. You can receive this information either for certain character or for all the characters in the text.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Bottom	**Long**, read-only	This property is indexed by the index of a symbol in the recognized text. It returns the coordinate of the bottom border of the symbol's rectangle as relative to the deskewed black-and-white plane of the source image.
CharConfidence	**Long**, read-only	This property is indexed by the index of a symbol in the recognized text. It returns the character confidence. It is in the range from 0 to 100, and 255 corresponds to the fact that confidence is undefined. It represents an estimate of recognition confidence of a character in percentage points. The greater its value, the greater the confidence. Character confidence can be undefined, for example, for characters which were added during text editing. In this case, the value of this property is -1. To calculate character confidence more accurately, set the **IRecognizerParams::ExactConfidenceCalculation** property to TRUE. See also **What is the difference between the CharConfidence and the IsSuspicious properties?**
Left	**Long**, read-only	This property is indexed by the index of a symbol in the recognized text. It returns the coordinate of the left border of the character's rectangle as relative to the deskewed black-and-white plane of the source image.
PageNumber	**Long**, read-only	This property is indexed by the index of a symbol in the recognized text. It returns the number of the page on which the specified symbol is located.
Right	**Long**, read-only	This property is indexed by the index of a symbol in the recognized text. It returns the coordinate of the right border of the symbol's rectangle as relative to the deskewed black-and-white plane of the source image.
SymbolsCount	**Long**, read-only	Returns the number of symbols in the text, including the special characters.
Text	**String**, read-only	Provides access to the whole recognized text in a form of Unicode string. This string may contain the following special characters: • 0x2028 — Line break symbol • 0x2029 — Paragraph break symbol • 0xFFFC — Object replacement character • 0x0009 — Tabulation • 0x005E — Circumflex accent **Note:** If the image has tables, text from the table cells will be stored in the logical reading order (left-to-right and top-down).
Top	**Long**, read-only	This property is indexed by the index of a symbol in the recognized text. It returns the coordinate of the top border of the symbol's rectangle as relative to the deskewed black-and-white plane of the source image.

Methods

Name	Description
GetCharacterData	Returns the information about all characters in the text as a set of arrays: the page numbers on which the characters are located, the coordinates of characters' rectangles, and characters' confidences.

SaveToAsciiXMLFile	Saves the recognized text into an XML file.
SaveToTextFile	Saves the recognized text into a text file with the specified encoding.

Output parameter

This object is the output parameter of the following methods:

- **RecognizeImageDocumentAsPlainText**, **RecognizeImageAsPlainText** method of the **Engine** object.
- **RecognizeImageDocumentAsPlainText** method of the **DocumentAnalyzer** object.
- **PlainText** property of the **FRDocument** object.
- **PlainText** property of the **FRPage** object.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Analyze and recognize the image
 FREngine::IPlainTextPtr text = Engine->RecognizeImageAsPlainText( L"D:\\Demo.tif", 0,
0, 0 );

 // Save results
 text->SaveToTextFile( L"D:\\sample.txt", FREngine::TET_Simple, FREngine::CP_Latin );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Analyze and recognize the image
 Dim Text As FREngine.PlainText
 Set Text = Engine.RecognizeImageAsPlainText("D:\Demo.tif")

 ' Save results
 Text.SaveToTextFile "D:\sample.txt", TET_Simple, CP_Latin
```

See also

Working with Text
Working with Properties

GetCharacterData Method of the PlainText Object

This method returns the information about all characters in the text as a set of arrays: the page numbers on which the characters are located, the coordinates of characters' rectangles, and characters' confidences.

Visual Basic Syntax

```
Method GetCharacterData(
  pageNumbers   As SAFEARRAY,
  leftBorders   As SAFEARRAY,
  topBorders    As SAFEARRAY,
  rightBorders  As SAFEARRAY,
  bottomBorders As SAFEARRAY,
  confidences   As SAFEARRAY
)
```

C++ Syntax

```
HRESULT GetCharacterData(
  SAFEARRAY* pageNumbers,
  SAFEARRAY* leftBorders,
  SAFEARRAY* topBorders,
  SAFEARRAY* rightBorders,
  SAFEARRAY* bottomBorders,
  SAFEARRAY* confidences
);
```

Parameters

pageNumbers

[out] An array of page numbers on which the characters are located.

leftBorders

[out] An array of coordinates of left borders of characters' rectangles as relative to the deskewed black-and-white plane of the source image.

topBorders

[out] An array of coordinates of top borders of characters' rectangles as relative to the deskewed black-and-white plane of the source image.

rightBorders

[out] An array of coordinates of right borders of characters' rectangles as relative to the deskewed black-and-white plane of the source image.

bottomBorders

[out] An array of coordinates of bottom borders of characters' rectangles as relative to the deskewed black-and-white plane of the source image.

confidences

[out] An array of characters' confidences.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

PlainText

SaveToAsciiXMLFile Method of the PlainText Object

This method saves the recognized text from the **PlainText** object into a XML file, including characters positions and recognition confidence information. The format of this XML file is the same as when exporting to XML format with **IXMLExportParams::WriteCharAttributes** property set to XCA_Ascii.

Visual Basic Syntax

```
Method SaveToAsciiXMLFile(
  path As String
)
```

C++ Syntax

```
HRESULT SaveToAsciiXMLFile(
  BSTR path
);
```

Parameters

path

[in] A string containing the full path to the file where the text should be saved. If this file does not exist, it will be created. If it does exist, it will be overwritten without prompt.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

PlainText

SaveToTextFile Method of the PlainText Object

This method saves the recognized text from the **PlainText** object into a text file with the specified encoding.

Visual Basic Syntax

```
Method SaveToTextFile(
  path         As String,
  encodingType As TextEncodingTypeEnum,
  codePageEnum As CodePageEnum
)
```

C++ Syntax

```
HRESULT SaveToTextFile(
  BSTR                 path,
  TextEncodingTypeEnum encodingType,
  CodePageEnum         codePageEnum
);
```

Parameters

path

[in] A string containing the full path to the file where the text should be saved. If this file does not exist, it will be created. If it exists, it will be overwritten without prompt.

encodingType

[in] Specifies the text encoding type. It may be set to one of the constants from the **TextEncodingTypeEnum** enumeration.

codePageEnum

[in] Specifies the code page. It may be set to one of the constants from the **CodePageEnum** enumeration. The value of this parameter is taken into account only when the *encodingType* parameter has value TET_Simple (exported text is not Unicode). If this property does not specify any code page (CP_Null), the code page is selected automatically.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Analyze and recognize the image
 FREngine::IPlainTextPtr text = Engine->RecognizeImageAsPlainText( L"D:\\Demo.tif", 0,
0, 0 );

 // Save results
 text->SaveToTextFile( L"D:\\sample.txt", FREngine::TET_Simple, FREngine::CP_Latin );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Analyze and recognize the image
 Dim Text As FREngine.PlainText
 Set Text = Engine.RecognizeImageAsPlainText("D:\Demo.tif")

 ' Save results
 Text.SaveToTextFile "D:\sample.txt", TET_Simple, CP_Latin
```

See also

PlainText

Document–Related Objects

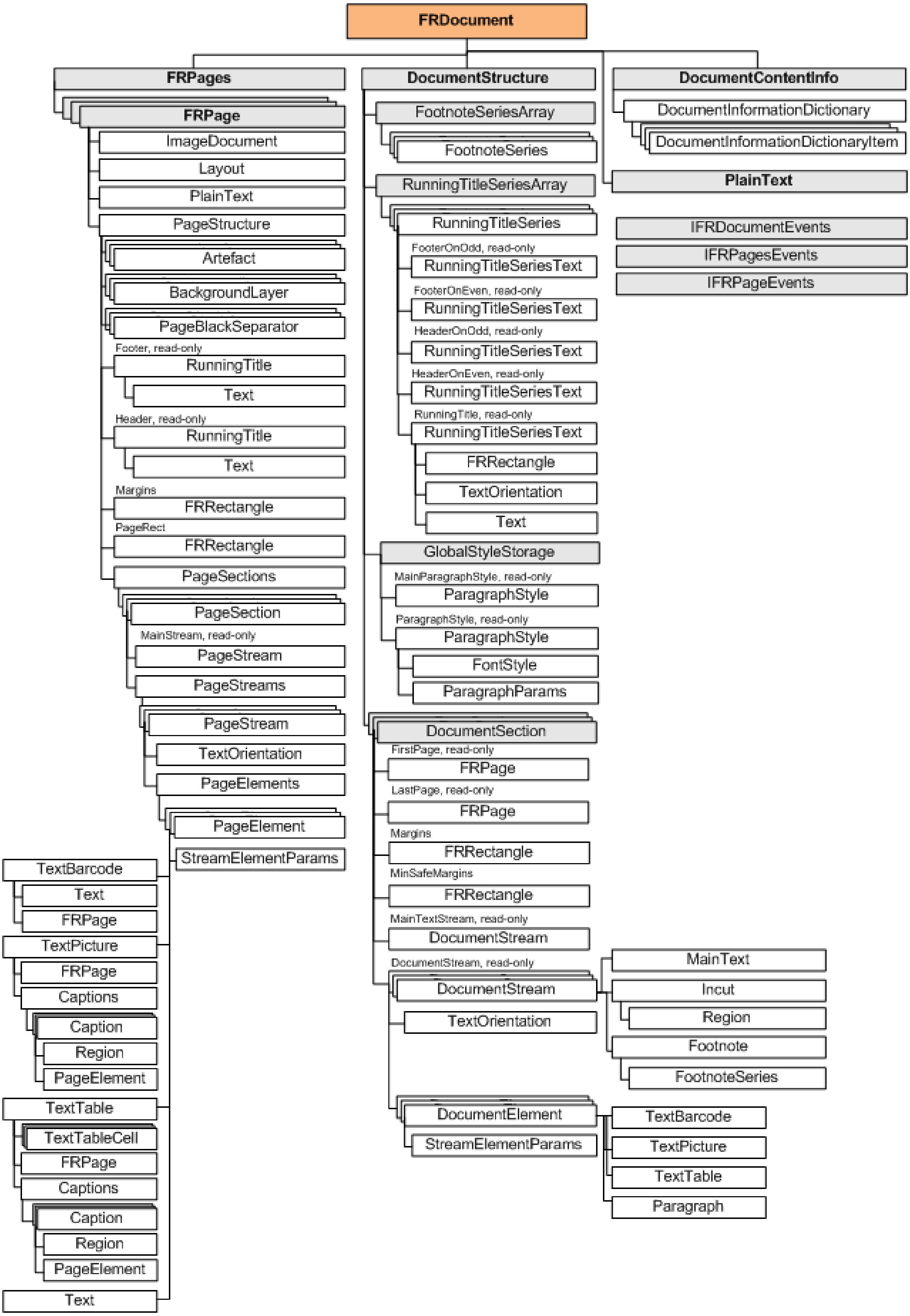
These objects can be divided into the following three groups:

- Document Organization Objects
  Document organization is represented by the document itself and its pages. ABBYY FineReader Engine provides the **FRDocument**, **FRPages** and **FRPage** objects for working with the document and its pages.

- Document Synthesis Objects
  Document synthesis is performed after recognition and allows the program to recreate the logical structure of a document and formatting attributes including headers, footers, page numbers, fonts and styles and more. ABBYY FineReader Engine provides the **DocumentStructure** and **PageStructure** objects and a set of their subobjects to access the results of document and page synthesis.

- Supplementary Objects
  Some additional information which is stored in the document, such as information about the author, keywords, subject, title of the document, can be accessed using these objects.

The document-related objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

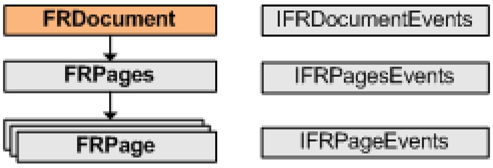
Document Organization Objects

Document organization is represented by the document itself and its pages. ABBYY FineReader Engine provides the **FRDocument**, **FRPages** and **FRPage** objects for working with the document and its pages. The **FRDocument** object is at the top of the document organization object's hierarchy. It exposes a set of analysis, recognition, synthesis and export methods. The **FRPage** object provides a set of methods for working with a certain page. The **FRPages** object contains the collection of document pages.

This section contains the descriptions of the following document organization objects and callback interfaces:

- **FRDocument**
- **FRPages**
- **FRPage**
- **IFRDocumentEvents**
- **IFRPagesEvents**
- **IFRPageEvents**

The document organization objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

FRDocument Object (IFRDocument Interface)

This object corresponds to a processing document which may contain several pages. The **FRDocument** object is a root for a collection of document pages. Each page represents an open image file and image layout. The object contains properties for accessing different document attributes such as its author, keywords, subject, and title, which are obtainable via the **DocumentContentInfo** property, and provides a set of properties and methods for document processing.

It is not recommended to recognize more than one document with the use of a single instance of the **FRDocument** object, as it may lead to unpredictable effects. Create a new instance of the **FRDocument** object for each new document.

After you have finished your work with the **FRDocument** object, release all the resources that were used by this object (use the **Close** method).

⚠**Important!** Pointers to child object's interfaces are valid until the **FRDocument** object exists. An attempt to access a child object after its parent object has been destroyed may result in error. Please, see for details Working with Properties.

The **FRDocument** object is a so-called "connectable object." It may be declared *WithEvents* in Visual Basic. For a C++ user, this means that it supports the **IConnectionPointContainer** interface. To receive notification events during recognition, a C++ user should create an object derived from the **IFRDocumentEvents** interface, then set up the connection between it and the events source implemented in the **FRDocument** object by standard COM means.

The methods of the **FRDocument** object report information about document processing progress through special outgoing interfaces. These interfaces are **IFRDocumentEvents** (for C++) and dispinterface **DIFRDocumentEvents** (for Visual Basic). It should be noted that Visual Basic users should not care for details of implementing event interfaces, as this development platform provides easy means for handling them.

Properties

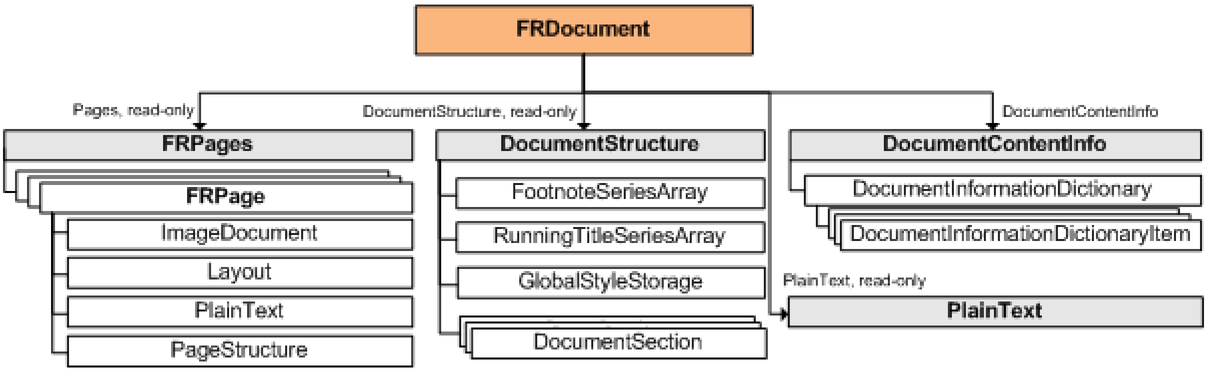
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DocumentContentInfo	**DocumentContentInfo**	Returns a reference to the **DocumentContentInfo** object, which contains information about the author, keywords, subject, and title of the document and stores the document information dictionary.
DocumentStructure	**DocumentStructure**, read-only	Provides access to the logical structure and styles of the document. This property becomes meaningful only after document synthesis.

PageFlushingPolicy	**PageFlushingPolicyEnum**	Specifies if the **ImageDocument** and the **Layout** objects for corresponding pages should be unloaded and saved to disk if there are no references to these objects. This property is PFP_Auto by default. **Note:** To unload and save to disk the **ImageDocument** and the **Layout** objects for separate pages of the document, use the **IFRPage::Flush** method for the corresponding pages.
Pages	**FRPages**, read-only	Returns a collection of pages of a document.
PlainText	**PlainText**, read-only	Returns the text of the document in a special "plain text" format.
TempDir	**String**	Specifies the path to the folder where the temporary image files in the ABBYY FineReader Engine internal format are stored.

Methods

Name	**Description**
AddImage	Adds one open image, represented by the **ImageDocument** object, to the document.
AddImageFile	Opens an image file and adds the pages corresponding to the opened file to the document.
AddImageFileWithPassword	Opens a password-protected image file and adds the pages corresponding to the opened file to the document.
AddImageFileWithPasswordCallback	Opens an image file using the **IImagePasswordCallback** interface and adds the pages corresponding to the opened file to the document.
Analyze	Performs layout analysis of all pages in the document.
AnalyzeAndRecognize	Performs layout analysis, recognition, and page synthesis of all pages in the document.
AnalyzeAndRecognizePages	Performs layout analysis, recognition, and page synthesis of the specified pages in the document.
AnalyzePages	Performs layout analysis of specified pages in a document.
Close	Releases all the resources that were used by the **FRDocument** object and returns the object into the initial state (as after its creation with the **IEngine::CreateFRDocument** method).
Export	Saves the document into a file in an external format.
ExportPages	Saves the specified pages into a file in an external format.
Process	Performs layout analysis, recognition, and synthesis of all pages in the document.
Recognize	Performs recognition and page synthesis of all pages in the document.
RecognizePages	Performs recognition and page synthesis of the specified pages in the document.
Synthesize	Performs document synthesis of all pages in the document.
SynthesizePages	Performs document synthesis of the specified pages in the document.

Related objects



Output parameter

The **FRDocument** object is the output parameter of the **CreateFRDocument** and **CreateFRDocumentFromImage** methods of the **Engine** object.

See also

FRPage
IFRDocumentEvents
Working with Connectable Objects
Working with Properties

See samples: Hello, RecognizedTextProcessing, CustomLanguage, EventsHandling

AddImage Method of the FRDocument Object

This method adds one open image, represented by the **ImageDocument** object, to a document.

This method does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method AddImage(
  image As ImageDocument
)
```

C++ Syntax

```
HRESULT AddImage(
  IImageDocument* image
);
```

Parameters

image

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be added.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRDocument
Working with Images

AddImageFile Method of the FRDocument Object

This method opens a specified image file and adds the pages corresponding to the opened file to a document. This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method AddImageFile(
  imageFileName As String,
  prepareMode   As PrepareImageMode,
  pageIndices   As LongsCollection
)
```

C++ Syntax

```
HRESULT AddImageFile(
  BSTR               imageFileName,
  IPrepareImageMode* prepareMode,
  ILongsCollection*  pageIndices
);
```

Parameters

imageFileName

[in] This variable contains a full path to the image file to open. For example, "C:\MyPictures\MyPic.bmp".

prepareMode

[in] This parameter refers to the **PrepareImageMode** object which specifies how an image will be preprocessed during opening.

pageIndices

[in] This parameter refers to the **LongsCollection** object which specifies the indices of the pages which have to be added to a document. This parameter is optional and may be 0, in which case all the pages corresponding to the opened file will be added to the document.

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
// Create FRDocument object
FREngine::IFRDocumentPtr frDocument = Engine->CreateFRDocument();
// Add image file
frDocument->AddImageFile( L"D:\\Demo.tif", 0, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
Public Engine As FREngine.Engine
' Create FRDocument object
Dim frDocument As FREngine.FRDocument
Set frDocument = Engine.CreateFRDocument
' Add image file
frDocument.AddImageFile "D:\Demo.tif"
```

See also

FRDocument
IFRDocument::AddImageFileWithPassword
IFRDocument::AddImageFileWithPasswordCallback
IFRDocument::AddImage
Working with Images

See sample: EventsHandling

AddImageFileWithPassword Method of the FRDocument Object

This method opens a password-protected image file and adds the pages corresponding to the opened file to a document.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method AddImageFileWithPassword(
  imageFileName As String,
  password      As String,
  prepareMode   As PrepareImageMode,
  pageIndices   As LongsCollection
)
```

C++ Syntax

```
HRESULT AddImageFileWithPassword(
  BSTR               imageFileName,
  BSTR               password,
  IPrepareImageMode* prepareMode,
  ILongsCollection*  pageIndices
);
```

Parameters

imageFileName

[in] This variable contains the full path to the image file to be opened. For example, "C:\MyPictures\MyPic.bmp".

password

[in] This variable contains a password for accessing images in PDF format.

prepareMode

[in] This parameter refers to the **PrepareImageMode** object which specifies how an image will be preprocessed during opening.

pageIndices

[in] This parameter refers to the **LongsCollection** object which specifies the indices of the pages which have to be added to a document. This parameter is optional and may be 0, in which case all the pages corresponding to the opened file will be added to the document.

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

FRDocument
IFRDocument::AddImageFileWithPasswordCallback
IFRDocument::AddImageFile
Working with Images

AddImageFileWithPasswordCallback Method of the FRDocument Object

This method opens an image file using the **IImagePasswordCallback** interface and adds the pages corresponding to the opened file to a document.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method AddImageFileWithPasswordCallback(
  imageFileName As String,
  callback      As ImagePasswordCallback,
  prepareMode   As PrepareImageMode,
  pageIndices   As LongsCollection
)
```

C++ Syntax

```
HRESULT AddImageFileWithPasswordCallback(
  BSTR                    imageFileName,
  IImagePasswordCallback* callback,
  IPrepareImageMode*      prepareMode,
  ILongsCollection*       pageIndices
);
```

Parameters

imageFileName

[in] This variable contains the full path to the image file to be opened. For example, "C:\MyPictures\MyPic.pdf".

callback

[in] This variable refers to the interface of the user-implemented object of the type **ImagePasswordCallback** which is used to handle possible password requests for accessing images in PDF format. This parameter is optional and may be 0, in which case password-protected files cannot be processed.

prepareMode

[in] This parameter refers to the **PrepareImageMode** object which specifies how an image will be preprocessed during opening.

pageIndices

[in] This parameter refers to the **LongsCollection** object which specifies the indices of the pages which have to be added to a document. This parameter is optional and may be 0, in which case all the pages corresponding to the opened file will be added to the document.

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

FRDocument
IFRDocument::AddImageFileWithPassword
IFRDocument::AddImageFile
Working with Images

Analyze Method of the FRDocument Object

This method performs the layout analysis of all pages in a document.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method Analyze(
  pageProcessingParams As PageProcessingParams
)
```

C++ Syntax

```
HRESULT Analyze(
  IPageProcessingParams* pageProcessingParams
);
```

Parameters

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument
Working with Profiles

AnalyzeAndRecognize Method of the FRDocument Object

This method performs the layout analysis, recognition, and page synthesis of all pages in the document.

Visual Basic Syntax

```
Method AnalyzeAndRecognize(
  pageProcessingParams   As PageProcessingParams,
  synthesisParamsForPage As SynthesisParamsForPage
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognize(
  IPageProcessingParams*   pageProcessingParams,
  ISynthesisParamsForPage* synthesisParamsForPage
);
```

Parameters

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case each page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis or recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.
- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument

AnalyzeAndRecognizePages Method of the FRDocument Object

This method performs layout analysis, recognition, and page synthesis of the specified pages in the document.

Visual Basic Syntax

```
Method AnalyzeAndRecognizePages(
  pageIndices            As LongsCollection,
  pageProcessingParams   As PageProcessingParams,
  synthesisParamsForPage As SynthesisParamsForPage
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognizePages(
  ILongsCollection*        pageIndices,
  IPageProcessingParams*   pageProcessingParams,
  ISynthesisParamsForPage* synthesisParamsForPage
);
```

Parameters

pageIndices

[in] This parameter refers to the **LongsCollection** object that contains the numbers of pages to be processed.

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case each page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis or recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.
- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument
Working with Profiles

AnalyzePages Method of the FRDocument Object

This method performs the layout analysis of specified pages in a document.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method AnalyzePages(
  pageIndices          As LongsCollection,
  pageProcessingParams As PageProcessingParams
)
```

C++ Syntax

```
HRESULT AnalyzePages(
  ILongsCollection*      pageIndices,
  IPageProcessingParams* pageProcessingParams
);
```

Parameters

pageIndices

[in] This parameter refers to the **LongsCollection** object that contains the numbers of pages to be analyzed.

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters — all page processing parameters are set to default values, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument
Working with Profiles

Close Method of the FRDocument Object

This method releases all the resources that were used by the **FRDocument** object (frees the memory, removes temporary files). The **FRDocument** object is returned to the initial state — the state of the object after its creation with the **IEngine::CreateFRDocument** method.

Visual Basic Syntax

```
Method Close()
```

C++ Syntax

```
HRESULT Close();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

- We recommend that you use this method each time you have finished to work with the current **FRDocument** object. After the method is called, the object can be reused.

- For .NET calling of this method is required. We recommend using it in **finally** blocks.

See also

FRDocument
Engine

Export Method of the FRDocument Object

This method saves document into a file in an external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants. This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method Export(
  exportFileName As String,
  format         As FileExportFormatEnum,
  exportParams   As Unknown
)
```

C++ Syntax

```
HRESULT Export(
  BSTR                 exportFileName,
  FileExportFormatEnum format,
  IUnknown*            exportParams
);
```

Parameters

exportFileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description for the supported file formats.

exportParams

[in] Pass the export parameters object of the type corresponding to your file format through this input parameter. For example, if you are saving the text into an RTF file, create an **RTFExportParams** object, set the necessary parameters in it, and pass it to this method as the *exportParams* input parameter. This parameter may be 0, in which case the default values for the export parameters are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRDocument
IEngine::ExportPages
IExporter::ExportPages

See samples: Hello, EventsHandling

ExportPages Method of the FRDocument Object

This method saves specified pages into a file in an external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants. This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method ExportPages(
  exportFileName As String,
  format         As FileExportFormatEnum,
  exportParams   As Unknown,
  pageIndices    As LongsCollection
)
```

C++ Syntax

```
HRESULT ExportPages(
  BSTR                 exportFileName,
  FileExportFormatEnum format,
  IUnknown*            exportParams,
  ILongsCollection*    pageIndices
);
```

Parameters

exportFileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description for the supported file formats.

exportParams

[in] Pass the export parameters object of the type corresponding to your file format through this input parameter. For example, if you are saving the text into an RTF file, create an **RTFExportParams** object, set the necessary parameters in it, and pass it to this method as the *exportParams* input parameter. This parameter may be 0, in which case the default values for the export parameters are used.

pageIndices

[in] This parameter refers to the **LongsCollection** object that contains the numbers of pages to be exported. Must not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRDocument
IEngine::ExportPages
IExporter::ExportPages

Process Method of the FRDocument Object

This method performs the layout analysis, recognition, and synthesis of all pages in a document. This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

Visual Basic Syntax

```
Method Process(
  pageProcessingParams        As PageProcessingParams,
  synthesisParamsForPage      As SynthesisParamsForPage,
  synthesisParamsForDocument  As SynthesisParamsForDocument
)
```

C++ Syntax

```
HRESULT Process(
  IPageProcessingParams*       pageProcessingParams,
  ISynthesisParamsForPage*     synthesisParamsForPage,
  ISynthesisParamsForDocument* synthesisParamsForDocument
);
```

Parameters

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case each page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParamsForDocument

[in] The **SynthesisParamsForDocument** object that stores parameters of document synthesis. This parameter may be 0. In this case the document is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If the layout analysis, recognition, or synthesis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument
Working with Profiles

See samples: Hello, RecognizedTextProcessing, CustomLanguage, EventsHandling

Recognize Method of the FRDocument Object

This method performs recognition and page synthesis of all pages in the document.

Visual Basic Syntax

```
Method Recognize(
  synthesisParamsForPage As SynthesisParamsForPage,
  extractionParams       As ObjectsExtractionParams
)
```

C++ Syntax

```
HRESULT Recognize(
  ISynthesisParamsForPage*  synthesisParamsForPage,
  IObjectsExtractionParams* extractionParams
);
```

Parameters

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] The **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.
- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument
Working with Profiles

RecognizePages Method of the FRDocument Object

This method performs recognition and page synthesis of the specified pages in the document.

Visual Basic Syntax

```
Method RecognizePages(
  pageIndices            As LongsCollection,
  synthesisParamsForPage As SynthesisParamsForPage,
  extractionParams       As ObjectsExtractionParams
)
```

C++ Syntax

```
HRESULT RecognizePages(
  ILongsCollection*         pageIndices,
  ISynthesisParamsForPage*  synthesisParamsForPage,
  IObjectsExtractionParams* extractionParams
);
```

Parameters

pageIndices

[in] This parameter refers to the **LongsCollection** object that contains the numbers of pages to be recognized.

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] The **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.
- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

See also

FRDocument
Working with Profiles

Synthesize Method of the FRDocument Object

This method performs document synthesis of all pages in the document.

Visual Basic Syntax

```
Method Synthesize(
  synthesisParamsForDocument As SynthesisParamsForDocument
)
```

C++ Syntax

```
HRESULT Synthesize(
  ISynthesisParamsForDocument* synthesisParamsForDocument
);
```

Parameters

synthesisParamsForDocument

[in] The **SynthesisParamsForDocument** object that stores parameters of document synthesis. This parameter may be 0. In this case the document is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If synthesis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

See also

FRDocument
IFRDocument::SynthesizePages
Working with Profiles

SynthesizePages Method of the FRDocument Object

This method performs document synthesis of the specified pages in the document.

Visual Basic Syntax

```
Method SynthesizePages(
  pageIndices                As LongsCollection,
  synthesisParamsForDocument As SynthesisParamsForDocument
)
```

C++ Syntax

```
HRESULT SynthesizePages(
  ILongsCollection*            pageIndices,
  ISynthesisParamsForDocument* synthesisParamsForDocument
);
```

Parameters

pageIndices

[in] This parameter refers to the **LongsCollection** object that contains the indexes of pages to be synthesized.

synthesisParamsForDocument

[in] The **SynthesisParamsForDocument** object that stores parameters of document synthesis. This parameter may be 0. In this case, the pages are synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If synthesis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRDocument** object.

See also

FRDocument
IFRDocument::Synthesize
Working with Profiles

FRPages Object (IFRPages Interface)

This object is a collection of document pages. The collection is accessible via the **FRDocument** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

The **FRPages** object is a so-called "connectable object". It may be declared *WithEvents* in Visual Basic. For C++ user this fact means that it supports the **IConnectionPointContainer** interface. To receive notification events during recognition, a C++ user should create an object derived from the **IFRPagesEvents** interface, then set up the connection between it and events source implemented in **FRPages** object by standard COM means.

The methods of the **FRPages** object report the information about document processing progress through a special outgoing interfaces. These interfaces are **IFRPagesEvents** (for C++) and a dispinterface **DIFRPagesEvents** (for Visual Basic). It's worth noting that Visual

Basic users should not care for details of event interfaces implementation as this development platform provides easy means for handling them.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**FRPage**, read-only	Provides access to one page of the collection.

Methods

Name	Description
Find	Returns index of specified page in collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
Renumber	Renumbers elements of collection.
Swap	Exchanges the contents of two elements.

See also

FRDocument
FRPage
Working with Connectable Objects
Working with Properties

See sample: RecognizedTextProcessing

Find Method of the FRPages Object

This method returns index of specified page in collection. If there is no such page in the collection, -1 is returned. This method does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPages** object.

Visual Basic Syntax

```
Method Find(
  page As FRPage
)As Long
```

C++ Syntax

```
HRESULT Find(
  IFRPage* page,
  long* index
);
```

Parameters

page

[in] The **FRPage** object contains a page that must be find.

index

[out] This parameter contains the index of element which corresponds to specified page.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRPages
FRDocument
FRPage

Renumber Method of the FRPages Object

This method renumbers elements of collection. This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPages** object.

Visual Basic Syntax

```
Method Renumber(
  newOrder As LongsCollection
)
```

C++ Syntax

```
HRESULT Renumber(
  ILongsCollection* newOrder
);
```

Parameters

newOrder

[in] This parameter refers to the **LongsCollection** object that contains a new order of the pages in collection.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRPages
FRDocument

Swap Method of the FRPages Object

This method exchanges the contents of two elements. This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPages** object.

Visual Basic Syntax

```
Method Swap(
  firstIndex  As Long,
  secondIndex As Long
)
```

C++ Syntax

```
HRESULT Swap(
  long firstIndex,
  long secondIndex
);
```

Parameters

firstIndex

[in] This parameter contains the index of first element in collection for exchange.

secondIndex

[in] This parameter contains the index of second element in collection for exchange.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRPages
FRDocument

FRPage Object (IFRPage Interface)

This object corresponds to a page of a document. It provides a set of page processing methods. This object is an element of the **FRPages** collection.

The **FRPage** object is a so-called "connectable object." It may be declared *WithEvents* in Visual Basic. For a C++ user, this means that it supports the **IConnectionPointContainer** interface. To receive notification events during recognition, a C++ user should create an object derived from the **IFRPageEvents** interface, then set up the connection between it and the events source implemented in the **FRPage** object by standard COM means.

The methods of the **FRPage** object report information about page processing progress through special outgoing interfaces. These interfaces are **IFRPageEvents** (for C++) and the dispinterface **IFRPageEvents** (for Visual Basic). It should be noted that Visual Basic users should not care for details of implementing event interfaces, as this development platform provides easy means for handling them.

Properties

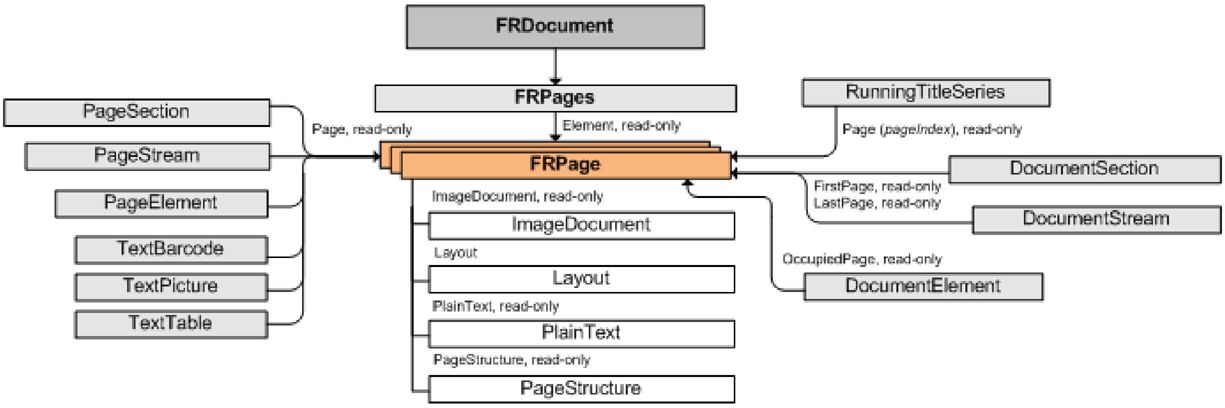
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
CanRedo	**Boolean**, read-only	Specifies whether the **Redo** method can be executed for the latest undone command which was called with the help of methods of the **FRPage** object.
CanUndo	**Boolean**, read-only	Specifies whether the **Undo** method can be executed for the latest command which was called with the help of methods of the **FRPage** object.
Document	**FRDocument**, read-only	Returns the **FRDocument** object which contains the specified page.
ImageDocument	**ImageDocument**, read-only	Returns the **ImageDocument** object for the specified page.
Layout	**Layout**	Provides access to the **Layout** object for the specified page. 📝**Note:** When you assign a **Layout** object to this property (for example, when transferring data from one page to another), the logical structure of the corresponding document becomes invalid. It is necessary to restore the document structure by calling one of the synthesis methods. However, you do not need to perform synthesis for the whole document, it is only necessary to synthesize changed pages, e.g. using the **IFRDocument::SynthesizePages** method.
PageStructure	**PageStructure**, read-only	Provides access to the logical structure and styles of the page. This property becomes meaningful only after document synthesis.
PlainText	**PlainText**, read-only	Returns the text of the page in a special "plain text" format.
UndoSupport	**Boolean**	Specifies whether the **Undo** and **Redo** methods are allowed. If the value of this property is TRUE, all the commands, which were called with the help of methods of the **FRPage** object, can be added to an undo stack. To add to the stack the commands, which were called with the help of the methods of the **FRPage** object, use the **Update** method.

Methods

Name	Description
Analyze	Analyzes the page.
AnalyzeAndRecognize	Performs layout analysis, recognition, and page synthesis of the page.
AnalyzeRegion	Analyzes layout of the image inside the specified region.
AnalyzeTable	Replaces a specified block with a table block and analyzes the structure of the table.
CleanRecognizerSession	Cleans recognizer session.
DetectOrientation	Detects page orientation.
Export	Saves a page into a file in an external format.
ExtractBarcodes	Finds and recognizes all barcode blocks. 📝**Note:** This method is obsolete and is intended to be removed in the next version of ABBYY FineReader Engine.
FindPageSplitPosition	Detects the direction of text on image and finds the position of splitting it on pages.
Flush	Unloads and saves to disk the **ImageDocument** and the **Layout** objects corresponding to the **FRPage** object if there are no references to them.
Recognize	Recognizes the page and performs page synthesis.
RecognizeBlocks	Recognizes text and performs page synthesis in an explicitly specified set of blocks.

Redo	Redoes the latest undone command which was called with the help of methods of the **FRPage** object.
RemoveGeometricalDistortions	Straightens out distorted lines on an image. Distorted lines may occur close to the binding when scanning/photographing thick books.
Undo	Undoes the latest command which was called with the help of methods of the **FRPage** object.
Update	Saves the latest changes in the **FRPage** object to the undo stack, in order these changes can be undone. This method has to be called for the changes, which were made with the help of the methods of the **FRPage** object, to add them to the stack.

Related objects



Output parameter

This object is the output parameter of the **Item** method of the **FRPages** object.

Input parameter

This object is the input parameter of the following methods:

- **FindFirstObjectOnPage** method of the **DocumentStream** object
- **Find** method of the **FRPages** object
- **OnProgress**, **OnRecognizerTip**, **OnRegionProcessed**, **OnPageProcessed** methods of the **IFRPageEvents** interface
- **PageRemoved** method of the **IFRPagesEvents** interface
- **AddPage**, **DeletePage** methods of the **RunningTitleSeries** object

See also

FRPages
Working with Connectable Objects
Working with Properties

See sample: RecognizedTextProcessing

Analyze Method of the FRPage Object

This method performs the layout analysis of the page.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

Visual Basic Syntax

```
Method Analyze(
  pageProcessingParams As PageProcessingParams
)
```

C++ Syntax

```
HRESULT Analyze(
  IPageProcessingParams* pageProcessingParams
);
```

Parameters

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

See also

FRPage
Working with Profiles

AnalyzeAndRecognize Method of the FRPage Object

This method performs layout analysis, recognition, and page synthesis of the page.

Visual Basic Syntax

```
Method AnalyzeAndRecognize(
  pageProcessingParams   As PageProcessingParams,
  synthesisParamsForPage As SynthesisParamsForPage
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognize(
  IPageProcessingParams*   pageProcessingParams,
  ISynthesisParamsForPage* synthesisParamsForPage
);
```

Parameters

pageProcessingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis and recognition. This parameter may be 0. In this case the page is processed with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis or recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

See also

FRPage
Working with Profiles

AnalyzeRegion Method of the FRPage Object

This function analyzes the layout of the image inside the specified region.

It does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

Visual Basic Syntax

```
Method AnalyzeRegion(
  region           As Region,
  processingParams As PageProcessingParams
)
```

C++ Syntax

```
HRESULT AnalyzeRegion(
  IRegion*               region,
  IPageProcessingParams* processingParams
);
```

Parameters

region

[in] This variable refers to the **Region** object that specifies the area on image that is to be analyzed. It should be set in coordinates of the deskewed black-and-white plane of the **ImageDocument**.

processingParams

[in] The **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the region is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets these parameters for layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- During the process of analysis of layout in region all the blocks that lay entirely inside the region are deleted from the **IFRPage::Layout** subobject. Zero or more new blocks may be added to the **Layout** as the result of this method call.

See also

FRPage
IDocumentAnalyzer::AnalyzeRegion
Working with Profiles

AnalyzeTable Method of the FRPage Object

This method replaces the specified block with the table block and analyzes the structure of table.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

Visual Basic Syntax

```
Method AnalyzeTable(
  blockIndex As Long,
  params     As PageProcessingParams
)
```

C++ Syntax

```
HRESULT AnalyzeTable(
  long                   blockIndex,
  IPageProcessingParams* params
);
```

Parameters

blockIndex

[in] This variable specifies the index of block in the collection of blocks which must be analyzed as table.

params

[in] The **PageProcessingParams** object that stores parameters of table layout analysis. This parameter may be 0. In this case the table is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- Table blocks always have rectangular regions; if the block was not rectangular, the new table block receives the region corresponding to bounding rectangle of the initial block.
- If the table structure cannot be analyzed, the **IFRPage::Layout** subobject is not changed.

See also

FRPage
Working with Profiles

CleanRecognizerSession Method of the FRPage Object

This method cleans recognizer session.

Recognizer session is created for recognition of each page. During this session recognizer performs a kind of self-teaching, and thus tunes itself for recognition of texts of a certain type. That is why it is good to use single recognizer instance for recognition of a number of blocks on a single page, as these blocks usually have text of similar type and therefore this improves speed and quality of recognition.

When you call this method, all the information that was received by the recognizer during this self-teaching is removed. Generally there is no need to use this method. However, you may find it useful, for example, if a page consists of two parts with extremely different types of text, then you may call this method after recognition of the first part and before the recognition of the second. This method also frees some memory.

Visual Basic Syntax

```
Method CleanRecognizerSession()
```

C++ Syntax

```
HRESULT CleanRecognizerSession();
```

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method does not report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

See also

FRPage

DetectOrientation Method of the FRPage Object

This method detects text orientation on the image. The method returns **TextOrientation** object, if orientation has been detected successfully, and NULL, if the program failed to detect orientation.

Visual Basic Syntax

```
Method DetectOrientation(
  orientationParams As OrientationDetectionParams
  extractionParams  As ObjectsExtractionParams,
  recognizerParams  As RecognizerParams
) As TextOrientation
```

C++ Syntax

```
HRESULT DetectOrientation(
  IOrientationDetectionParams* orientationParams,
  IObjectsExtractionParams*    extractionParams,
  IRecognizerParams*           recognizerParams,
  ITextOrientation**           result
);
```

Parameters

orientationParams

[in] This variable refers to the **OrientationDetectionParams** object that stores parameters of orientation detection. This parameter may be 0. In this case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

recognizerParams

[in] This variable refers to the **RecognizerParams** object that stores parameters of page recognition. This parameter may be 0. In this case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

result

[out, retval] A pointer to **ITextOrientation*** pointer variable that receives the interface pointer of the **TextOrientation** output object. This object provides access to the text orientation on the page. If orientation detection failed, NULL is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Calling this method is equivalent to the call to **IFRPage::Analyze** method with the following parameters of the input **PageProcessingParams** object: DetectOrientation = true, PerformPageAnalysis = false, RemoveGeometricalDictortions = false, DetectBarcodes = false, DetectInvertedImage = false.

See also

FRPage
IPageProcessingParams::DetectOrientation
IDocumentAnalyzer::DetectOrientation
Working with Profiles

Export Method of the FRPage Object

This method saves page into a file in an external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

Visual Basic Syntax

```
Method Export(
  exportFileName As String,
  format         As FileExportFormatEnum,
  exportParams   As Unknown
)
```

C++ Syntax

```
HRESULT Export(
  BSTR                 exportFileName,
  FileExportFormatEnum format,
  IUnknown*            exportParams
);
```

Parameters

exportFileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description for the supported file formats.

exportParams

[in] Pass the export parameters object of the type corresponding to your file format through this input parameter. For example, if you are saving the text into an RTF file, create an **RTFExportParams** object, set the necessary parameters in it, and pass it to this method as the *exportParams* input parameter. This parameter may be 0, in which case the default values for the export parameters are used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRPage
IEngine::ExportPages
IExporter::ExportPages

ExtractBarcodes Method of the FRPage Object

This method finds and recognizes all barcode blocks on an image, no other blocks are processed.

Visual Basic Syntax

```
Method ExtractBarcodes(
  barcodeParams    As BarcodeParams,
  extractionParams As ObjectsExtractionParams
)
```

C++ Syntax

```
HRESULT ExtractBarcodes(
  IBarcodeParams*          barcodeParams,
  IObjectsExtractionParams* extractionParams
);
```

Parameters

barcodeParams

[in] The **BarcodeParams** object that stores parameters of barcode recognition. This parameter may be 0. In this case the page is analyzed with default parameters (all barcode recognition parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] The **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- All existing blocks are deleted from the page.
- Calling this method is equivalent to the call to **IFRPage::AnalyzeAndRecognize** method with the following parameters of the input **PageProcessingParams** object: DetectBarcodes = true, PerformPageAnalysis = false, RemoveGeometricalDictortions = false, DetectOrientation = false, DetectInvertedImage = false.
- The method does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage**.
- This method is obsolete and is intended to be removed in the next version of ABBYY FineReader Engine.

See also

FRPage
IPageProcessingParams::DetectBarcodes
IDocumentAnalyzer::ExtractBarcodes
Working with Profiles

FindPageSplitPosition Method of the FRPage Object

This method detects the direction of text on image and finds the position of splitting it on pages, if it exists. It is used to detect the ability to split dual pages in a book.

The split position is defined by two lines, which coordinates are returned in the *startSplitPosition* and *endSplitPosition* parameters. The image area between these two lines should be removed when splitting image on pages. This area usually contains some garbage.

Visual Basic Syntax

```
Method FindPageSplitPosition(
  extractionParams   As ObjectsExtractionParams,
  splitDirection     As PageSplitDirectionEnum,
  startSplitPosition As Long,
  endSplitPosition   As Long
)
```

C++ Syntax

```
HRESULT FindPageSplitPosition(
  IObjectsExtractionParams* extractionParams,
  PageSplitDirectionEnum*   splitDirection,
  long*                     startSplitPosition,
  long*                     endSplitPosition
);
```

Parameters

extractionParams

[in] The **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

splitDirection

[out] This variable receives the type of possible split: vertical split, horizontal split, or no split. Refer to the **PageSplitDirectionEnum** description for details.

startSplitPosition

[out] The coordinate of the first line, which defines split position (if a split is possible). The meaning of this value depends on the value of the *splitDirection* variable. If the possibility of vertical split is detected, it contains the horizontal coordinate of the split line. If the possibility of horizontal split is detected, it contains the vertical coordinate of the split line. Coordinate is given against the deskewed black-and-white page of the image.

endSplitPosition

[out] The coordinate of the second line, which defines split position (if a split is possible). The meaning of this value depends on the value of the *splitDirection* variable. If the possibility of vertical split is detected, it contains the horizontal coordinate of the split line. If the possibility of horizontal split is detected, it contains the vertical coordinate of the split line. Coordinate is given against the deskewed black-and-white page of the image.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRPage
PageSplitDirectionEnum
Working with Profiles

Flush Method of the FRPage Object

This method unloads and saves to disk the **ImageDocument** and the **Layout** objects corresponding to the **FRPage** object if there are no references to them. The method is useful when processing documents of large sizes, in which case it decreases memory usage but may slow down document processing. After the operation on a separate document page is complete, free all references to the **ImageDocument** and the **Layout** objects corresponding to the page and call the **Flush** method to decrease memory usage.

Visual Basic Syntax

```
Method Flush() As Boolean
```

C++ Syntax

```
HRESULT Flush(
  VARIANT_BOOL* result
);
```

Parameters

result

[out] The variable receives the result of the unloading. The value of this parameter is TRUE if the objects were unloaded successfully. Otherwise the value is FALSE.

Return Values

This method has no specific return values. It returns the standard return codes of the ABBYY FineReader Engine functions.

See also

FRPage
IFRDocument::PageFlushingPolicy

Recognize Method of the FRPage Object

This method recognizes a page and performs page synthesis.

Visual Basic Syntax

```
Method Recognize(
   synthesisParamsForPage As SynthesisParamsForPage,
   extractionParams       As ObjectsExtractionParams
 )
```

C++ Syntax

```
HRESULT Recognize(
   ISynthesisParamsForPage*  synthesisParamsForPage,
   IObjectsExtractionParams* extractionParams
 );
```

Parameters

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] The **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

See also

FRPage
Working with Profiles

RecognizeBlocks Method of the FRPage Object

This method recognizes text and performs page synthesis in an explicitly specified set of blocks.

Visual Basic Syntax

```
Method RecognizeBlocks(
   BlockIndices           As LongsCollection,
   synthesisParamsForPage As SynthesisParamsForPage,
   extractionParams       As ObjectsExtractionParams
 )
```

C++ Syntax

```
HRESULT RecognizeBlocks(
   ILongsCollection*         BlockIndices,
   ISynthesisParamsForPage*  synthesisParamsForPage,
   IObjectsExtractionParams* extractionParams
 );
```

Parameters

BlockIndices

[in] This parameter refers to the **LongsCollection** object that contains the indices of blocks to be recognized.

synthesisParamsForPage

[in] The **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] The **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **FRPage** object.

See also

FRPage
Working with Profiles

RemoveGeometricalDistortions Method of the FRPage Object

This method straightens out distorted lines on an image. Distorted lines may occur close to the binding when scanning/photographing thick books.

We recommend calling this method after the page orientation has been corrected and a double-page spread has been split into two separate pages, if necessary. This method should be called after layout analysis, for example after the **IFRPage::Analyze** method. We recommend setting the correct recognition language before analysis, especially for texts in Chinese, Japanese and Korean.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of **FRPage**.

Visual Basic Syntax

```
Method RemoveGeometricalDistortions(
  extractionParams As ObjectsExtractionParams
)
```

C++ Syntax

```
HRESULT RemoveGeometricalDistortions(
  IObjectsExtractionParams* extractionParams
);
```

Parameters

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object corresponding to the parameters used for straightening out distorted lines on an image. This parameter may be 0, in which case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If straightening is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Calling this method is equivalent to the call to **IFRPage::Analyze** method with the following parameters of the input **PageProcessingParams** object: RemoveGeometricalDictortions = true, PerformPageAnalysis = false, DetectOrientation = false, DetectBarcodes = false, DetectInvertedImage = false.

See also

FRPage
IPageProcessingParams::RemoveGeometricalDistortions
IDocumentAnalyzer::RemoveGeometricalDistortions
Working with Profiles

Redo Method of the FRPage Object

This method redoes the latest undone command which was called with the help of the methods of the **FRPage** object. The command which was called with the help of the methods of the **FRPage** object can be redone only if it was previously added to the undo stack with the help of the **IFRPage::Update** method. The method can be executed if the value of the **IFRPage::CanRedo** property is TRUE.

Visual Basic Syntax

```
Method Redo()
```

C++ Syntax

```
HRESULT Redo();
```

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

FRPage
IFRPage::Undo
IFRPage::Update

Undo Method of the FRPage Object

This method undoes the latest command which was called with the help of methods of the **FRPage** object. The command which was called with the help of the methods of the **FRPage** object can be undone only if it was previously added to the undo stack with the help of the **IFRPage::Update** method. The method can be executed if the value of the **IFRPage::CanUndo** property is TRUE.

Visual Basic Syntax

```
Method Undo()
```

C++ Syntax

```
HRESULT Undo();
```

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

FRPage
IFRPage::Redo,
IFRPage::Update

Update Method of the FRPage Object

This method saves the latest changes in the **FRPage** object to the undo stack, in order these changes can be undone. This method has to be called for the changes, which were made with the help of the methods of the **FRPage** object, to add them to the stack.

Visual Basic Syntax

```
Method Update()
```

C++ Syntax

```
HRESULT Update();
```

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

See also

FRPage
IFRPage::Undo
IFRPage::Redo

IFRDocumentEvents Interface

This is callback interface that is used for reporting events from the **FRDocument** object to the listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to receive notifications from the **FRDocument** object should declare it *WithEvents* and implement the following Sub's:

```
Public WithEvents doc As FREngine.FRDocument
Private Sub doc_OnPageProcessed(ByRef sender As FRDocument,
                                ByVal index  As Long,
                                ByVal stage  As PageProcessingStageEnum)
...
End Sub
Private Sub doc_OnProgress(ByRef sender     As FRDocument,
                           ByVal percentage As Long,
                           ByRef cancel     As Boolean)
...
End Sub
Private Sub doc_OnRecognizerTip(ByRef sender As FRDocument,
                                ByVal tip    As String,
                                ByRef cancel As Boolean)
...
End Sub
```

An object receiving notifications through this interface's methods may do the following inside the methods' implementation:

- Process any Windows messages, which is useful in applications having User Interface, to avoid an effect that the application "is not responding" during long operations.
- Report percentage of image loading, document analysis, recognition, synthesis, and export performed.
- Report an information about document analysis, recognition, synthesis, and export completed.

Methods

Name	Description
OnPageProcessed	Delivers to the client an information about page processing completed.
OnProgress	Delivers to the client an information about approximate percentage of the current operation (image loading, analysis, recognition, and etc.).
OnRecognizerTip	Delivers to the client recognizer tips.

See also

FRDocument
Working with Connectable Objects

See sample: EventsHandling

OnPageProcessed Method of the IFRDocumentEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRDocument** object. It delivers to the client information about page processing completed. It may process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnPageProcessed(
  ByRef sender As FRDocument,
  ByVal index  As Long,
  ByVal stage  As PageProcessingStageEnum
)
```

C++ Syntax

```
HRESULT OnPageProcessed(
  IFRDocument*            sender,
  long                    index,
  PageProcessingStageEnum stage
);
```

Parameters

sender

[in] This parameter refers to the **FRDocument** object which sends notifications.

index

[in] This parameter contains the index of the processed page.

stage

[in] This variable of the **PageProcessingStageEnum** type specifies the stage of processing.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRDocumentEvents
FRDocument

OnProgress Method of the IFRDocumentEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRDocument** object. It delivers to the client an information about approximate percentage of the current operation (image loading, analysis, recognition, and etc.). Its implementation may show a progress indicator, as it is done in ABBYY FineReader. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnProgress(
  ByRef sender     As FRDocument,
  ByVal percentage As Long,
  ByRef cancel     As Boolean
)
```

C++ Syntax

```
HRESULT OnProgress(
  IFRDocument*  sender,
  long          percentage,
  VARIANT_BOOL* cancel
);
```

Parameters

sender

[in] This parameter refers to the **FRDocument** object which sends notifications.

percentage

[in] This parameter contains the percent of the work currently done. It is in the range from 0 to 100.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process should be terminated. In this case the processing function returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRDocumentEvents
FRDocument

See sample: EventsHandling

OnRecognizerTip Method of the IFRDocumentEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRDocument** object. It delivers to the client recognizer tips. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnRecognizerTip(
  ByRef sender As FRDocument,
  ByVal tip    As String,
  ByRef cancel As Boolean
)
```

C++ Syntax

```
HRESULT OnRecognizerTip(
  IFRDocument*  sender,
  BSTR          tip,
  VARIANT_BOOL* cancel
);
```

Parameters

sender

[in] This parameter refers to the **FRDocument** object which sends notifications.

tip

[in] This parameter contains the recognizer tip.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process should be terminated. In this case the processing function, that reports the tip, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRDocumentEvents
FRDocument

IFRPagesEvents Interface

This is callback interface that is used for reporting events from the **FRPages** object to the listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to receive notifications from the **FRPages** object should declare it *WithEvents* and implement the following Sub's:

```
Public WithEvents pages As FREngine.FRDocument
Private Sub pages_PageRemoved(ByRef sender As FRPages,
                              ByRef page   As FRPage,
                              ByVal index  As Long)
...
End Sub
Private Sub pages_PageAdded(ByRef sender As FRPages,
                            ByVal index  As Long)
...
End Sub
Private Sub pages_PagesRenumbered(ByRef sender As FRPages)
...
End Sub
```

An object receiving notifications through this interface's methods may do the following inside the methods' implementation:

- Process any Windows messages, which is useful in applications having User Interface, to avoid an effect that the application "is not responding" during long operations.
- Report an information about page removing and adding completed
- Report information about pages renumbering completed.

Methods

Name	Description
PageAdded	Delivers to the client information about page adding completed.
PageRemoved	Delivers to the client information about page removing completed.
PagesRenumbered	Delivers to the client information about pages renumbering completed.

See also

FRPages
Working with Connectable Objects

PageAdded Method of the IFRPagesEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPages** object. It delivers to the client information about page adding completed. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub PageAdded(
  ByRef sender As FRPages,
  ByVal index  As Long
)
```

C++ Syntax

```
HRESULT PageAdded(
  IFRPages* sender,
  long      index
);
```

Parameters

sender

[in] This parameter refers to the **FRPages** object which sends notifications.

index

[in] This parameter contains index of added page.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

See also

IFRPagesEvents
FRPages

PageRemoved Method of the IFRPagesEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPages** object. It delivers to the client information about page removing completed.

Visual Basic Syntax

```
Sub PageRemoved(
  ByRef sender As FRPages,
  ByRef page   As FRPage,
  ByVal index  As Long
)
```

C++ Syntax

```
HRESULT PageRemoved(
  IFRPages* sender,
  IFRPage*  page,
  long      index
);
```

Parameters

sender

[in] This parameter refers to the **FRPages** object which sends notifications.

page

[in] This parameter refers to the **FRPage** object which is removed.

index

[in] This parameter contains the index of removed page.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

See also

IFRPagesEvents
FRPages

PagesRenumbered Method of the IFRPagesEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPages** object. It delivers to the client an information about pages renumbering completed. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub PagesRenumbered(
  ByRef sender As FRPages
)
```

C++ Syntax

```
HRESULT PagesRenumbered(
  IFRPages* sender
);
```

Parameters

sender

[in] This parameter refers to the **FRPages** object which sends notifications.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

See also

IFRPagesEvents
FRPages

IFRPageEvents Interface

This is callback interface that is used for reporting events from the **FRPage** object to the listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to receive notifications from the **FRPage** object should declare it *WithEvents* and implement the following Sub's:

```
Public WithEvents page As FREngine.FRPage
Private Sub page_OnPageProcessed(ByRef sender   As FRPage,
                                 ByVal stage  As PageProcessingStageEnum)
...
End Sub
Private Sub page_OnProgress(ByRef sender     As FRPage,
                            ByVal percentage As Long,
                            ByRef cancel     As Boolean)
...
End Sub
Private Sub page_OnRecognizerTip(ByRef sender As FRPage,
                                 ByVal tip    As String,
                                 ByRef cancel As Boolean)
...
End Sub
Private Sub page_OnRegionProcessed(ByRef sender               As FRPage,
                                   ByVal recognizerPassNumber As Long,
                                   ByRef region               As Region,
                                   ByRef cancel               As Boolean
...
End Sub
```

An object receiving notifications through this interface's methods may do the following inside the methods' implementation:

- Process any Windows messages, which is useful in applications having User Interface, to avoid an effect that the application "is not responding" during long operations.
- Report percentage of document analysis, recognition, and export.
- Report recognizer tips to the user.

- Report information about document analysis, recognition, and export completed.

Methods

Name	Description
OnPageProcessed	Delivers to the client an information about page processing completed.
OnProgress	Delivers to the client information about approximate percentage of the current operation (analysis, recognition, and export).
OnRecognizerTip	Delivers to the client recognizer tips.
OnRegionProcessed	Delivers to the client an information about region which is processed.

See also

FRPage
Working with Connectable Objects

OnPageProcessed Method of the IFRPageEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPage** object. It delivers to the client information about page processing completed. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnPageProcessed(
  ByRef sender As FRPage,
  ByVal stage  As PageProcessingStageEnum
)
```

C++ Syntax

```
HRESULT OnPageProcessed(
  IFRPage*                sender,
  PageProcessingStageEnum stage
);
```

Parameters

sender

[in] This parameter refers to the **FRPage** object which sends notifications.

stage

[in] This variable of the **PageProcessingStageEnum** type specifies the stage of processing.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRPageEvents
FRPage

OnProgress Method of the IFRPageEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPage** object. It delivers to the client information about approximate percentage of the current operation (analysis, recognition, and export). Its implementation may show a progress indicator, as it is done in ABBYY FineReader. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnProgress(
  ByRef sender     As FRPage,
  ByVal percentage As Long,
  ByRef cancel     As Boolean
)
```

C++ Syntax

```
HRESULT OnProgress(
  IFRPage*      sender,
  long          percentage,
  VARIANT_BOOL* cancel
);
```

Parameters

sender

[in] This parameter refers to the **FRPage** object which sends notifications.

percentage

[in] This parameter contains the percent of the work currently done. It is in the range from 0 to 100.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process should be terminated. In this case the processing function, that reports the percentage, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRPageEvents
FRPage

OnRecognizerTip Method of the IFRPageEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPage** object. Its Its implementation may report recognizer tips to the user. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnRecognizerTip(
  ByRef sender As FRPage,
  ByVal tip    As String,
  ByRef cancel As Boolean
)
```

C++ Syntax

```
HRESULT OnRecognizerTip(
  IFRPage*      sender,
  BSTR          tip,
  VARIANT_BOOL* cancel
);
```

Parameters

sender

[in] This parameter refers to the **FRPage** object which sends notifications.

tip

[in] This parameter contains the recognizer tip.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process should be terminated. In this case the processing function, that reports the tip, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRPageEvents
FRPage

OnRegionProcessed Method of the IFRPageEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **FRPage** object. It delivers to the client information about processing region. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnRegionProcessed(
  ByRef sender               As FRPage,
  ByVal recognizerPassNumber As Long,
  ByRef region               As Region,
  ByRef cancel               As Boolean
)
```

C++ Syntax

```
HRESULT OnRegionProcessed(
  IFRPage*      sender,
  long          recognizerPassNumber,
  IRegion*      region,
  VARIANT_BOOL* cancel
);
```

Parameters

sender

[in] This parameter refers to the **FRPage** object which sends notifications.

recognizerPassNumber

[in] This parameter reports the number of the recognition pass. It may be 0, 1 or 2. Rectangles from different passes may be filled up with different colors as it is done in ABBYY FineReader.

region

[in] This parameter refers to the **Region** object which corresponds to the region which is processed.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process should be terminated. In this case the processing function returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that all exceptions thrown inside the method are caught and handled and no exceptions are propagated outside the method. Propagation of an exception outside the method may lead to unpredictable results (such as program termination).

See also

IFRPageEvents
FRPage

Document Synthesis Objects

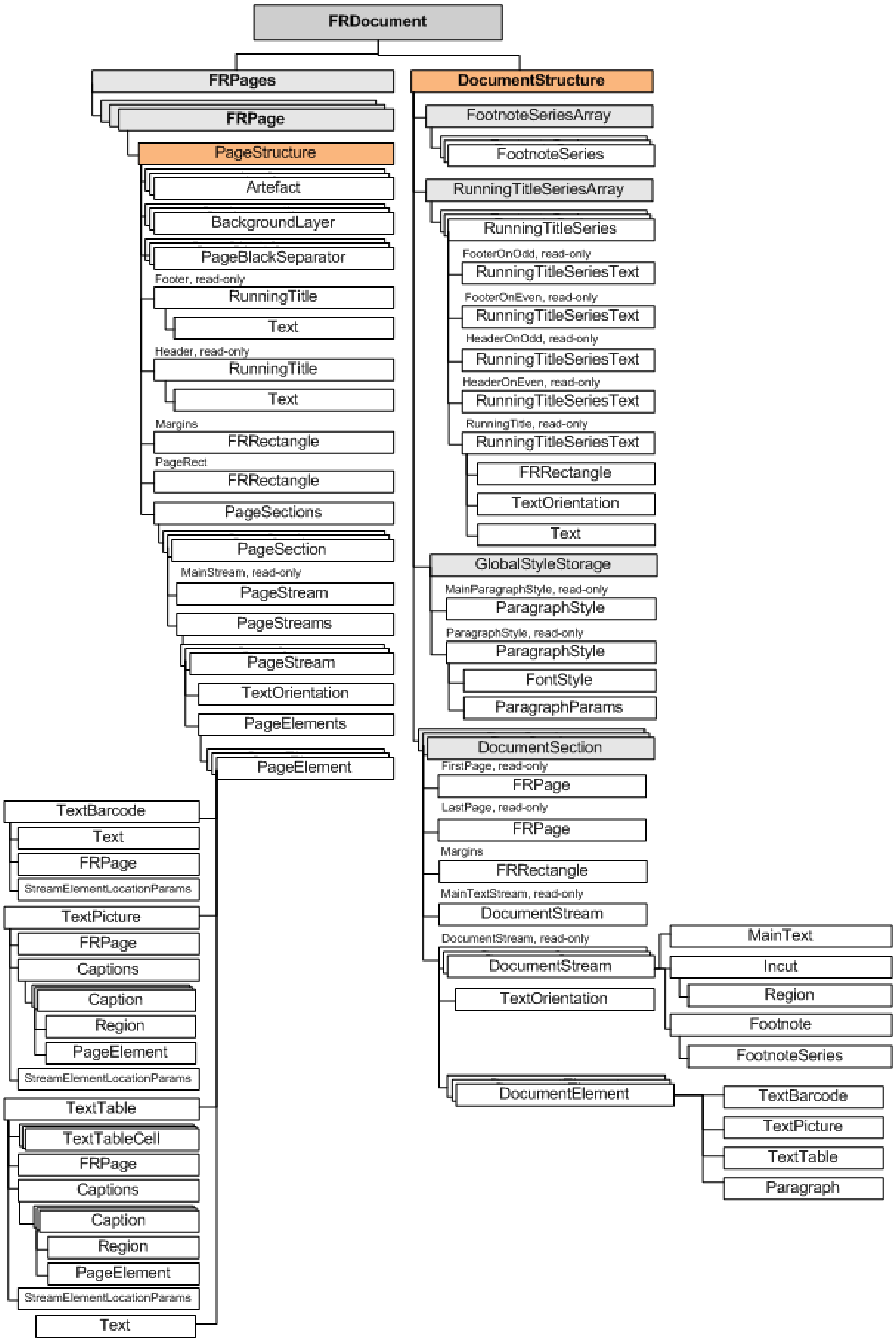
Document synthesis is performed after recognition and allows the program to recreate the logical structure of a document and formatting attributes including headers, footers, page numbers, fonts and styles and more. ABBYY FineReader Engine provides the **DocumentStructure** and **PageStructure** objects and a set of their subobjects to access the results of document and page synthesis.

This section contains descriptions of the following document synthesis objects:

- **DocumentStructure**
- **DocumentSection**
- **DocumentStream**
- **DocumentElement**
- **PageStructure**
- **PageSections**
- **PageSection**
- **PageStreams**
- **PageStream**
- **PageElements**
- **PageElement**
- **StreamElementLocationParams**
- **MainText**
- **FootnoteSeriesArray**
- **FootnoteSeries**
- **Footnote**
- **Incut**
- **Artefact**
- **TextPicture**
- **TextTable**
- **TextTableCell**
- **Captions**
- **Caption**

- **RunningTitleSeriesArray**
- **RunningTitleSeries**
- **RunningTitle**
- **RunningTitleSeriesText**
- **PageBlackSeparator**
- **BackgroundLayer**
- **GlobalStyleStorage**
- **ParagraphStyle**
- **FontStyle**

The document synthesis objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

DocumentStructure Object (IDocumentStructure Interface)

This object provides access to the logical structure of a document. Document structure is detected during document synthesis and is used for re-creation of the logical structure of a document and formatting attributes during export. The object exposes a set of methods and properties for working with logical sections and styles of the document.

⚠**Important!** Pointers to child object's interfaces are valid until the parent object exists. An attempt to access a child object after its parent object has been destroyed may result in error. Please, see for details **Working with Properties**.

Note: Document structure may cause high memory usage, e.g. when iterating through the document structure. Therefore we recommend unloading the pages of the document structure each time you have finished to work with them. Use the **UnloadUnusedPages** method for it.

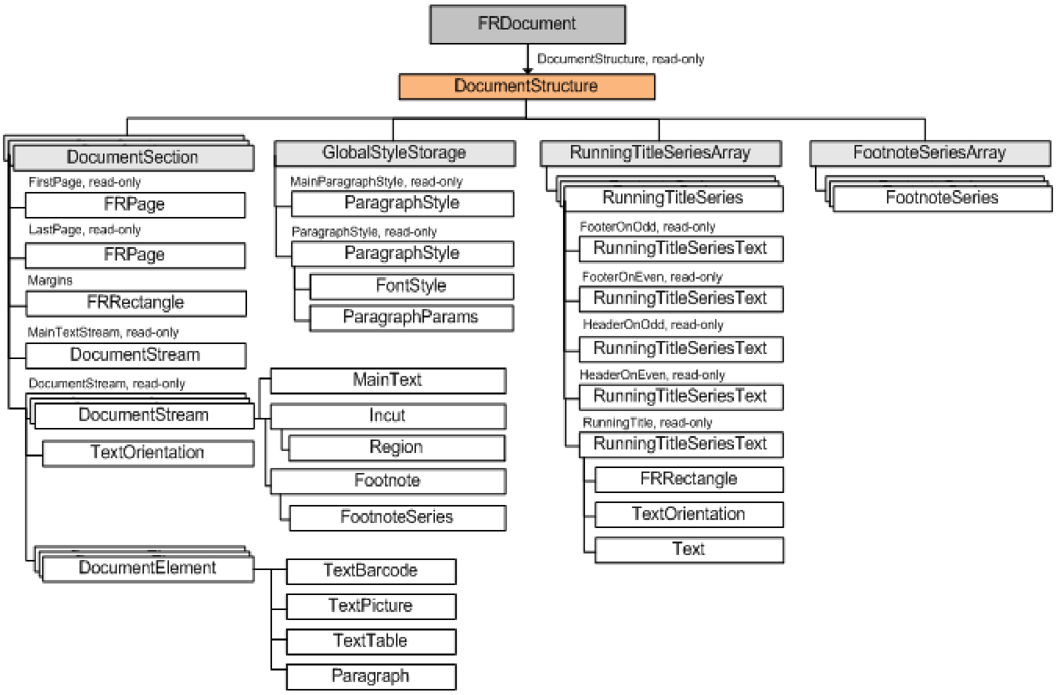
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DocumentSection	**DocumentSection**, read-only	Provides access to the document section with the specified index.
DocumentSectionsCount	**Long**, read-only	Stores the number of sections in the document.
FootnoteSeriesArray	**FootnoteSeriesArray**, read-only	Provides access to the array of footnote series of the document.
GlobalStyleStorage	**GlobalStyleStorage**, read-only	Provides access to the global style storage of the document.
RunningTitleSeriesArray	**RunningTitleSeriesArray**, read-only	Provides access to the array of running titles series of the document.

Methods

Name	Description
FindFirstSectionOnPage	Finds the first document section on the specified page.
FindFootnoteByHyperlinkTarget	Finds the footnote by the hyperlink target refers to this footnote.
GetAllFootnoteTargets	Returns the collection of hyperlink targets of the document.
UnloadAllPages	Unloads all pages of the logical structure of the document.
UnloadUnusedPages	Unloads the pages of the document structure which are not in use at the moment.

Related objects



See also

Working with the Logical Structure of a Document
FRDocument
Working with Properties

DocumentSection Property of the DocumentStructure Object

This property provides access to the document section with the specified index.

Visual Basic Syntax

```
Property DocumentSection( sectionIndex As Long ) As DocumentSection
  read-only
```

C++ Syntax

```
HRESULT get_DocumentSection(
  long               sectionIndex,
  IDocumentSection** result
);
```

Parameters

sectionIndex

[in] This variable specifies the index of the section in the internal collection of sections of the document structure. Must be in range from 0 to **IDocumentStructure::DocumentSectionsCount** -1.

result

[out, retval] A pointer to **IDocumentSection*** pointer variable that receives the interface pointer of the returned **DocumentSection** object.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStructure
Working with Properties

FindFirstSectionOnPage Method of the DocumentStructure Object

This method allows you to find the first document section on the specified page. The returned section may start from some previous page of the document and continue on the current page and following pages. If there is no section on the page, the first section found in the document after this page will be returned. If there is no such section, NULL is returned.

Visual Basic Syntax

```
Method FindFirstSectionOnPage(
  pageIndex As Long
) As DocumentSection
```

C++ Syntax

```
HRESULT FindFirstSectionOnPage(
  long               pageIndex,
  IDocumentSection** result
);
```

Parameters

pageIndex

[in] This variable specifies the index of the page in the collection of document pages.

result

[out, retval] A pointer to **IDocumentSection*** pointer variable that receives the interface pointer of the returned **DocumentSection** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStructure

FindFootnoteByHyperlinkTarget Method of the DocumentStructure Object

This method allows you to find the footnote by the hyperlink target refers to this footnote.

Visual Basic Syntax

```
Method FindFootnoteByHyperlinkTarget(
  target As String
) As DocumentStream
```

C++ Syntax

```
HRESULT FindFootnoteByHyperlinkTarget(
  BSTR              target,
  IDocumentStream** result
);
```

Parameters

target

[in] This variable specifies the hyperlink target.

result

[out, retval] A pointer to **IDocumentStream*** pointer variable that receives the interface pointer to the returned **DocumentStream** object which contains the footnote. If the footnote is not found, NULL is returned.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

Remarks

The method returns the **DocumentStream** object of the type ST_Footnote. You can receive then the **Footnote** object with the **IDocumentStream::GetAsFootnote** method.

See also

DocumentStructure

GetAllFootnoteTargets Method of the DocumentStructure Object

This method allows you to receive the collection of hyperlink targets of the document.

Visual Basic Syntax

```
Method GetAllFootnoteTargets() As StringsCollection
```

C++ Syntax

```
HRESULT GetAllFootnoteTargets(
  IStringsCollection** result
);
```

Parameters

result

[out, retval] A pointer to **IStringsCollection*** pointer variable that receives the interface pointer of the returned **StringsCollection** object, which contains the collection of hyperlink targets.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStructure

UnloadUnusedPages Method of the DocumentStructure Object

This method allows you to free the memory, which was used by the logical pages of the document. It unloads the pages, which are not in use at the moment. We recommend to use this method when working with big documents and iterating thorough the document structure. Unload the pages each time you have finished to work with them.

Visual Basic Syntax

```
Method UnloadUnusedPages()
```

C++ Syntax

```
HRESULT UnloadUnusedPages();
```

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStructure

UnloadAllPages Method of the DocumentStructure Object

This method allows you to free the memory which was used by the logical pages of the document. It unloads all pages of the logical structure of the document.

⚠**Important!** After this method call all the elements of the document structure become invalid.

Visual Basic Syntax

```
Method UnloadAllPages()
```

C++ Syntax

```
HRESULT UnloadAllPages();
```

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStructure

DocumentSection Object (IDocumentSection Interface)

This object represents one logical section of the document. For example, a book may have several parts, a magazine may include several articles. Such parts of the book, or articles of the magazine, will be detected as document sections.

A section usually contains several pages. The first and the last pages of the section can be accessed using the **FirstPage** and **LastPage** properties.

Each section consists of one or several document streams: main text, incuts, footnotes, and artefacts. The document section can have only one stream of the main text type. All the streams of the section are to be located on the pages of the section inside the page margins (**Margins** property).

Properties

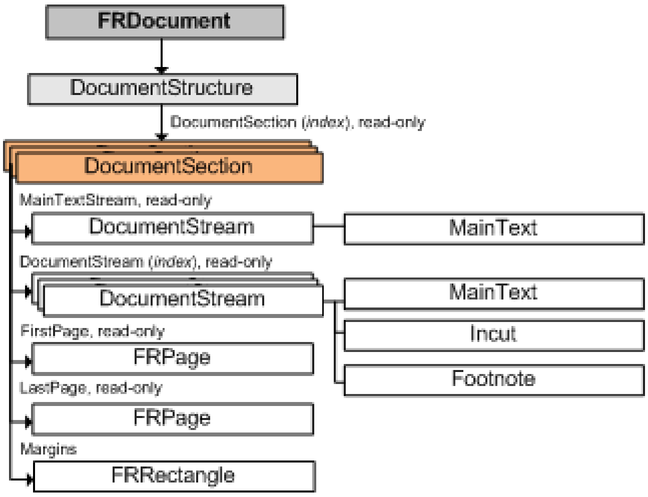
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
AreMarginsMirroredOnEvenPages	**Boolean**	Specifies whether the page margins are mirrored on even pages.
DocumentStream	**DocumentStream**, read-only	Provides access to one document stream by its index.
DocumentStreamsCount	**Long**, read-only	Stores the number of streams in the document section.
FirstPage	**FRPage**, read-only	Provides access to the first page of the document section.
LastPage	**FRPage**, read-only	Provides access to the last page of the document section.
MainTextStream	**DocumentStream**, read-only	Provides access to the main text stream of the section.
Margins	**FRRectangle**	Specifies the rectangle of page margins in the document section. Margins are measured in hundredth parts of point.
PageHeight	**Long**	Specifies the page height in the section in hundredth parts of point.
PageWidth	**Long**	Specifies the page width in the section in hundredth parts of

		point.

Methods

Name	Description
AddNewStream	Adds a new stream into the document section.

Related objects



Output parameter

This object is the output parameter of the **FindFirstSectionOnPage** method of the **DocumentStructure** object.

See also

DocumentStream
DocumentStructure
Working with Properties

DocumentStream Property of the DocumentSection Object

This property provides access to the document stream with the specified index.

Visual Basic Syntax

```
Property DocumentStream( streamIndex As Long ) As DocumentStream
  read-only
```

C++ Syntax

```
HRESULT get_DocumentStream(
  long              streamIndex,
  IDocumentStream** result
);
```

Parameters

streamIndex

[in] This variable specifies the index of the stream in the collection of document streams. Must be in range from 0 to **IDocumentSection::DocumentStreamsCount** -1.

result

[out, retval] A pointer to **IDocumentStream*** pointer variable that receives the interface pointer to the returned **DocumentStream** object.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentSection

AddNewStream Method of the DocumentSection Object

This method adds a new stream of the specified type into the document section.

Note: The document section may have only one stream of the main text type. Artefacts cannot be added to the document section.

Visual Basic Syntax

```
Method AddNewStream(
  streamType As StreamTypeEnum
) As DocumentStream
```

C++ Syntax

```
HRESULT AddNewStream(
  StreamTypeEnum    streamType,
  IDocumentStream** result
);
```

Parameters

streamType

[in] This variable specifies the type of the new stream. Use the **StreamTypeEnum** enumeration constants to specify the type of the stream. The ST_Artefact constant must not be used as the value of this parameter.

result

[out, retval] A pointer to **IDocumentStream*** pointer variable that receives the interface pointer to the returned **DocumentStream** object which contains the new stream.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentSection

DocumentStream Object (IDocumentStream Interface)

This object provides access to one document stream. Document stream is an element of the logical structure of a document. Document streams can be of several types (the **Type** property): main text, incut, and footnote. They are parts of the document section. Each document section may have only one main text stream and several incuts and footnotes. Running titles are not document streams. The **DocumentStream** object exposes a set of methods which provide access to the extended attributes of a stream of specific type.

Document stream consists of document elements: paragraphs, tables, pictures, or barcodes. You can navigate through the elements using the **FirstElement**, **LastElement**, **NextElement**, **PrevElement** properties.

Properties

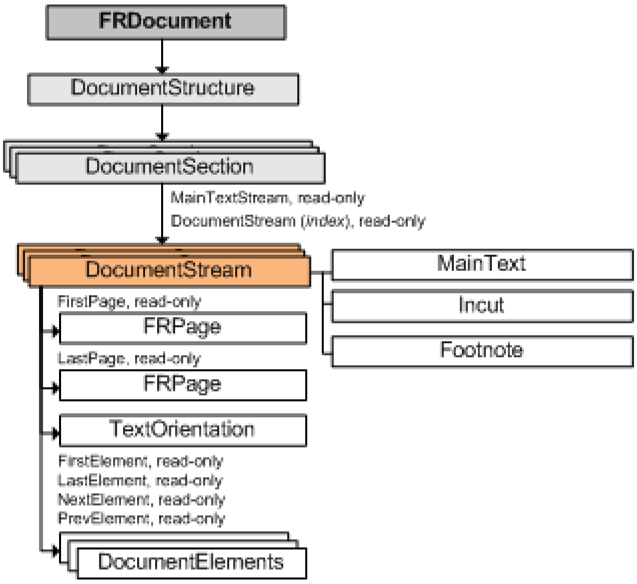
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
FirstElement	**DocumentElement**, read-only	Returns the first element in the document stream.
FirstPage	**FRPage**, read-only	Provides access to the first page of the document stream.
IsEmpty	**Boolean**, read-only	Specifies whether the stream does not contain any elements.
LastElement	**DocumentElement**, read-only	Returns the last element in the document stream.
LastPage	**FRPage**, read-only	Provides access to the last page of the document stream.
NextElement	**DocumentElement**, read-only	Returns the next element in the document stream.
PrevElement	**DocumentElement**, read-only	Returns the previous element in the document stream.
TextOrientation	**TextOrientation**, read-only	Returns text orientation in the stream. The property returns a constant object.

Type	**StreamTypeEnum**, read-only	Stores the type of the stream: main text, incut, or footnote. Document stream cannot be an artefact.

Methods

Name	**Description**
FindFirstObjectOnPage	Finds the first document element of the stream on the specified page.
GetAllPageElements	Returns all page elements of the document stream.
GetAsFootnote	Returns the document stream as the **Footnote** object. If the document stream is not a footnote, NULL is returned.
GetAsIncut	Returns the document stream as the **Incut** object. If the document stream is not an incut, NULL is returned.
GetAsMainText	Returns the document stream as the **MainText** object. If the document stream is not a main text, NULL is returned.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **FindFootnoteByHyperlinkTarget** method of the **DocumentStructure** object
- **AddNewStream** methods of the **DocumentSection** object

See also

Working with the Logical Structure of a Document
DocumentSection
DocumentStructure
Working with Properties

NextElement Property of the DocumentStream Object

This property retrieves the next element of the document stream. If there is no next element, NULL is returned.

Visual Basic Syntax

```
Property NextElement(
  element As DocumentElement
) As DocumentElement
```

C++ Syntax

```
HRESULT NextElement(
  IDocumentElement*  element,
  IDocumentElement** result
);
```

Parameters

element

[in] This parameter refers to the **DocumentElement** object which next element is to be found.

result

[out, retval] A variable of type **IDocumentElement*** that receives a pointer to the interface of the **DocumentElement** object which contains the next element.

Return Values

This property has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

DocumentStream
IDocumentStream::PrevElement
Working with Properties

PrevElement Property of the DocumentStream Object

This property retrieves the previous element of the document stream. If there is no previous element, NULL is returned.

Visual Basic Syntax

```
Property PrevElement(
  element As DocumentElement
) As DocumentElement
```

C++ Syntax

```
HRESULT PrevElement(
  IDocumentElement*  element,
  IDocumentElement** result
);
```

Parameters

element

[in] This parameter refers to the **DocumentElement** object which previous element is to be found.

result

[out, retval] A variable of type **IDocumentElement*** that receives a pointer to the interface of the **DocumentElement** object which contains the previous element.

Return Values

This property has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

DocumentStream
IDocumentStream::NextElement
Working with Properties

FindFirstObjectOnPage Method of the DocumentStream Object

This method allows you to find the first document element of the stream on the specified page. The returned element may start from some previous page of the document and continue on the current page and following pages. If there are no elements of this stream on the page, the first element of the stream found in the document after this page will be returned. If there is no such element, NULL is returned.

Visual Basic Syntax

```
Method FindFirstObjectOnPage(
  page As FRPage
) As DocumentElement
```

C++ Syntax

```
HRESULT FindFirstObjectOnPage(
  IFRPage*           page,
  IDocumentElement** result
);
```

Parameters

page

[in] This variable refers to the **FRPage** object which contains the page to find element on.

result

[out, retval] A pointer to **IDocumentElement*** pointer variable that receives the interface pointer to the returned **DocumentElement** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStream

GetAllPageElements Method of the DocumentStream Object

This method returns a collection of page elements presented in the stream.

Visual Basic Syntax

```
Method GetAllPageElements() As PageElements
```

C++ Syntax

```
HRESULT GetAllPageElements(
  IPageElements** result
);
```

Parameters

result

[out, retval] A pointer to **IPageElements*** pointer variable that receives the interface pointer to the returned **PageElements** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStream

GetAsFootnote Method of the DocumentStream Object

This method returns the document stream as the **Footnote** object. If the document stream is not a footnote, NULL is returned.

Visual Basic Syntax

```
Method GetAsFootnote() As Footnote
```

C++ Syntax

```
HRESULT GetAsFootnote(
  IFootnote** result
);
```

Parameters

result

[out, retval] A pointer to **IFootnote*** pointer variable that receives the interface pointer to the returned **Footnote** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStream
Footnote

GetAsIncut Method of the DocumentStream Object

This method returns the document stream as the **Incut** object. If the document stream is not an incut, NULL is returned.

Visual Basic Syntax

```
Method GetAsIncut() As Incut
```

C++ Syntax

```
HRESULT GetAsIncut(
  IIncut** result
);
```

Parameters

result

[out, retval] A pointer to **IIncut*** pointer variable that receives the interface pointer to the returned **Incut** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStream
Incut

GetAsMainText Method of the DocumentStream Object

This method returns the document stream as the **MainText** object. If the document stream is not a main text, NULL is returned.

Visual Basic Syntax

```
Method GetAsMainText() As MainText
```

C++ Syntax

```
HRESULT GetAsMainText(
  IMainText** result
);
```

Parameters

result

[out, retval] A pointer to **IMainText*** pointer variable that receives the interface pointer to the returned **MainText** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentStream
MainText

DocumentElement Object (IDocumentElement Interface)

This object provides access to one element of the document stream. Document element is the minimal unit of the logical structure of a document. Document elements are paragraphs, barcodes, tables, and pictures. The type of the element is defined by the **Type** property. The object exposes the set of methods, which provides access to the properties of the document element of specific type.

Location of the document element in the document is specified by the number of pages, which contains the element (the **OccupiedPagesCount** property), and the pages itself (the **OccupiedPage** property). Usually a document element is located on one page, but multi-page tables or paragraphs which start on one page and continue till the other are located on several pages.

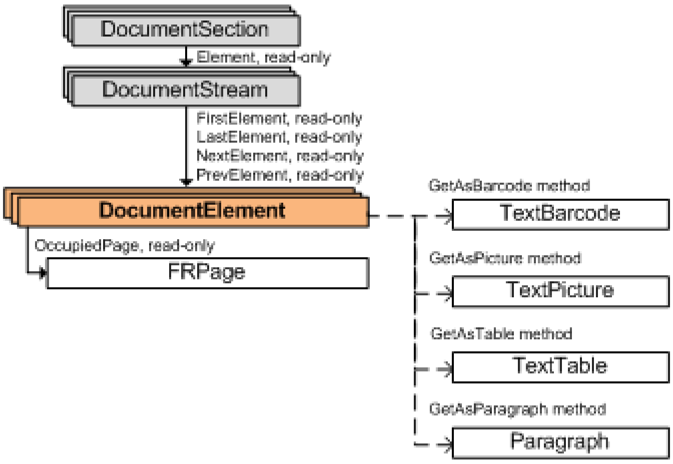
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
OccupiedPage	**FRPage**, read-only	Returns the page with the specified number from the collection of pages on which the current element is located.
OccupiedPagesCount	**Long**, read-only	Stores the number of pages on which the current element is located. In most cases, the value of this property is 1. But for multi-page tables or paragraphs which are located on several pages the value of this property may be more than 1.
Type	**DocumentElementTypeEnum**, read-only	Stores the type of the element: paragraph, barcode, table, or picture.

Methods

Name	Description
GetAsBarcode	Returns the document element as the **TextBarcode** object. If the document element is not a barcode, NULL is returned.
GetAsParagraph	Returns the document element as the **Paragraph** object. If the document element is not a paragraph, NULL is returned.
GetAsPicture	Returns the document element as the **TextPicture** object. If the document element is not a picture, NULL is returned.
GetAsTable	Returns the document element as the **TextTable** object. If the document element is not a text table, NULL is returned.

Related objects



Output parameter

This object is the output parameter of the **FindFirstObjectOnPage** method of the **DocumentStream** object

See also

Working with the Logical Structure of a Document
DocumentStream
Working with Properties

OccupiedPage Property of the DocumentElement Object

This property returns the page with the specified number from the collection of the element's pages.

Visual Basic Syntax

```
Property OccupiedPage( pageNumber As Long ) As FRPage
```

C++ Syntax

```
HRESULT get_OccupiedPage(
  long      pageNumber,
  IFRPage** result
);
```

Parameters

pageNumber

[in] This variable specifies the number of the page in the collection. The value of this parameter must be in range from 0 to **IDocumentElement:: OccupiedPagesCount** –1.

result

[out, retval] A pointer to **IFRPage*** pointer variable that receives the interface pointer to the returned **FRPage** object.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentElement
FRPage
Working with Properties

GetAsBarcode Method of the DocumentElement Object

This method returns the document element as the **TextBarcode** object. If the document element is not a barcode, NULL is returned.

Visual Basic Syntax

```
Method GetAsBarcode() As TextBarcode
```

C++ Syntax

```
HRESULT GetAsBarcode(
  ITextBarcode** result
);
```

Parameters

result

[out] A pointer to **ITextBarcode*** pointer variable that receives the interface pointer to the returned **TextBarcode** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentElement
TextBarcode

GetAsParagraph Method of the DocumentElement Object

This method returns the document element as the **Paragraph** object. If the document element is not a paragraph, NULL is returned.

Visual Basic Syntax

```
Method GetAsParagraph() As Paragraph
```

C++ Syntax

```
HRESULT GetAsParagraph(
  IParagraph** result
);
```

Parameters

result

[out] A pointer to **IParagraph*** pointer variable that receives the interface pointer to the returned **Paragraph** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentElement
Paragraph

GetAsPicture Method of the DocumentElement Object

This method returns the document element as the **TextPicture** object. If the document element is not a picture, NULL is returned.

Visual Basic Syntax

```
Method GetAsPicture() As TextPicture
```

C++ Syntax

```
HRESULT GetAsPicture(
  ITextPicture** result
);
```

Parameters

result

[out] A pointer to **ITextPicture*** pointer variable that receives the interface pointer to the returned **TextPicture** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentElement
TextPicture

GetAsTable Method of the DocumentElement Object

This method returns the document element as the **TextTable** object. If the document element is not a text table, NULL is returned.

Visual Basic Syntax

```
Method GetAsTable() As TextTable
```

C++ Syntax

```
HRESULT GetAsTable(
  ITextTable** result
);
```

Parameters

result

[out] A pointer to **ITextTable*** pointer variable that receives the interface pointer to the returned **TextTable** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

DocumentElement
TextTable

PageStructure Object (IPageStructure Interface)

This object provides access to the logical structure of the page. Page structure is detected during document synthesis and is used for re-creation of page logical structure and formatting attributes during export. The object exposes a set of methods and properties for working with logical sections and styles of the page.

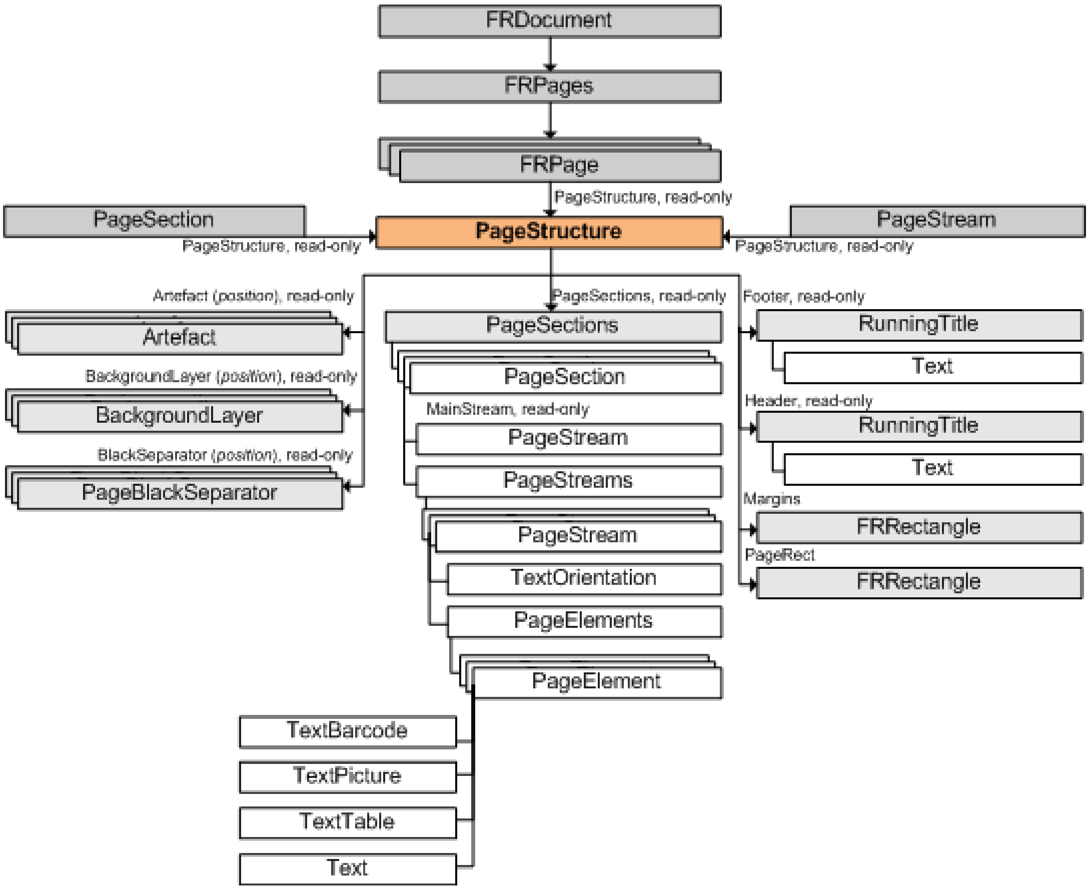
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Artefact	**Artefact**, read-only	Returns the artefact with the specified index.
ArtefactsCount	**Long**, read-only	Stores the number of artefacts on the page.
BackgroundLayer	**BackgroundLayer**, read-only	Returns the background layer with the specified index.
BackgroundLayersCount	**Long**, read-only	Stores the number of background layers on the page.
BlackSeparator	**PageBlackSeparator**, read-only	Returns the black separator with the specified index.
BlackSeparatorsCount	**Long**, read-only	Stores the number of black separators on the page.
Footer	**RunningTitle**, read-only	Stores the footer of the page.
Header	**RunningTitle**, read-only	Stores the header of the page.
IsPageOdd	**Boolean**, read-only	Specifies if the page is odd or even.
Margins	**FRRectangle**	Specifies the rectangle of the page with margins. **Note:** The property returns a constant object. To change the rectangle, you must first receive an intermediate **FRRectangle** object with the help of the **IEngine::CreateRectangle** method, change the necessary parameters, and then assign this object to the property.
PageRect	**FRRectangle**	Specifies the rectangle of the page which contains text. **Note.** The property returns a constant object. To change the rectangle, you must first receive an intermediate **FRRectangle** object with the help of the **IEngine::CreateRectangle** method, change the necessary parameters, and then assign this object to the property.
PageSections	**PageSections**, read-only	Provides access to the page sections.

Methods

Name	Description
AddArtefact	Creates an artefact on the page.
AddBackgroundLayer	Creates a background layer on the page.
AddBlackSeparator	Creates a black separator on the page.
CreateRunningTitle	Creates a header or a footer on the page.
DeleteRunningTitles	Deletes running titles from this page.
RemoveBackgroundLayer	Deletes the specified background layer from the page.
RemoveBlackSeparator	Deletes the specified black separator from the page.

Related objects



See also

Working with the Logical Structure of a Document
PageStream
Working with Properties

Artefact Property of the PageStructure Object

This property returns the artefact with the specified index.

Visual Basic Syntax

```
Property Artefact( position As Long ) As PageStream
  read-only
```

C++ Syntax

```
HRESULT get_Artefact(
  long          position,
  IPageStream** result
);
```

Parameters

position

[in] Specifies the index of the artefact in the collection of the page artefacts.

result

[out, retval] A pointer to **IPageStream*** pointer variable that receives the interface pointer to the returned **PageStream** object which contains the specified artefact.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStructure
PageStream
Working with Properties

BackgroundLayer Property of the PageStructure Object

This property returns the background layer with the specified index.

Visual Basic Syntax

```
Property BackgroundLayer( position As Long ) As BackgroundLayer
  read-only
```

C++ Syntax

```
HRESULT get_BackgroundLayer(
  long               position,
  IBackgroundLayer** result
);
```

Parameters

position

[in] Specifies the index of the background layer in the collection of the page background layers. It must be in range from 0 to **IPageStructure::BackgroundLayersCount** -1.

result

[out, retval] A pointer to **IBackgroundLayer*** pointer variable that receives the interface pointer to the returned **BackgroundLayer** object which contains the specified background layer.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStructure
BackgroundLayer
Working with Properties

BlackSeparator Property of the PageStructure Object

This property returns the black separator with the specified index.

Visual Basic Syntax

```
Property BlackSeparator( position As Long ) As PageBlackSeparator
  read-only
```

C++ Syntax

```
HRESULT get_BlackSeparator(
  long                  position,
  IPageBlackSeparator** result
);
```

Parameters

position

[in] Specifies the index of the black separator in the collection of the page black separators. It must be in range from 0 to **IPageStructure::BlackSeparatorsCount** -1.

result

[out, retval] A pointer to **IPageBlackSeparator*** pointer variable that receives the interface pointer to the returned **PageBlackSeparator** object which contains the specified artefact.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStructure
PageBlackSeparator
Working with Properties

AddArtefact Method of the PageStructure Object

This method creates an artefact on the page and adds it into the internal array of artefacts of the **PageStructure** object.

Visual Basic Syntax

```
Method AddArtefact() As PageStream
```

C++ Syntax

```
HRESULT AddArtefact(
  IPageStream** result
);
```

Parameters

result

[out, retval] A pointer to **IPageStream*** pointer variable that receives the interface pointer to the returned **PageStream** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

Remarks

The method returns the **PageStream** object of the type ST_Artefact. You can receive then the **Artefact** object with the **IPageStream::GetAsArtefact** method. The newly created object will be accessible via the **IPageStructure::Artefact** property either. The method increases the value of the **IPageStructure::ArtefactsCount** property.

See also

PageStructure
PageStream

AddBackgroundLayer Method of the PageStructure Object

This method creates a background layer on the page and adds it into the internal array of background layers of the **PageStructure** object.

Visual Basic Syntax

```
Method AddBackgroundLayer() As BackgroundLayer
```

C++ Syntax

```
HRESULT AddBackgroundLayer(
  IBackgroundLayer** result
);
```

Parameters

result

[out, retval] A pointer to **IBackgroundLayer*** pointer variable that receives the interface pointer to the returned **BackgroundLayer** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

Remarks

The newly created object will be accessible via the **IPageStructure::BackgroundLayer** property either. The method increases the value of the **IPageStructure::BackgroundLayersCount** property.

See also

PageStructure
BackgroundLayer

AddBlackSeparator Method of the PageStructure Object

This method creates a black separator on the page and adds it into the internal array of page black separators of the **PageStructure** object.

Visual Basic Syntax

```
Method AddBlackSeparator() As PageBlackSeparator
```

C++ Syntax

```
HRESULT AddBlackSeparator(
  IPageBlackSeparator** result
);
```

Parameters

result

[out, retval] A pointer to **IPageBlackSeparator*** pointer variable that receives the interface pointer to the returned **PageBlackSeparator** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

Remarks

The newly created object will be accessible via the **IPageStructure::BlackSeparator** property either. The method increases the value of the **IPageStructure::BlackSeparatorsCount** property.

See also

PageStructure
PageBlackSeparator

DeleteRunningTitles Method of the PageStructure Object

This method deletes running titles from this page.

Visual Basic Syntax

```
Method DeleteRunningTitles()
```

C++ Syntax

```
HRESULT DeleteRunningTitles();
```

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStructure

RemoveBackgroundLayer Method of the PageStructure Object

This method deletes the specified background layer from the page.

Visual Basic Syntax

```
Method RemoveBackgroundLayer(
  position As Long
)
```

C++ Syntax

```
HRESULT RemoveBackgroundLayer(
  long position
);
```

Parameters

position

[in] Specifies the index of the background layer in the collection of the page background layers. It must be in range from 0 to **IPageStructure::BackgroundLayersCount** -1.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStructure
BackgroundLayer

RemoveBlackSeparator Method of the PageStructure Object

This method deletes the specified black separator from the page.

Visual Basic Syntax

```
Method RemoveBlackSeparator(
  position As Long
)
```

C++ Syntax

```
HRESULT RemoveBlackSeparator(
  long position
);
```

Parameters

position

[in] Specifies the index of the black separator in the collection of the page black separators. It must be in range from 0 to **IPageStructure::BlackSeparatorsCount** -1.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStructure

PageSections Object (IPageSections Interface)

This object represents a collection of page sections.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

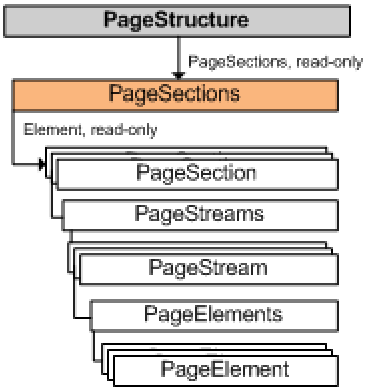
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**PageSection**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a **PageSection** object at the end of the collection.
Item	Provides access to a **PageSection** object in the collection.
RemoveAll	Removes all the elements from the collection.

Related objects



See also

Working with the Logical Structure of a Document
PageSection
PageStructure
Working with Properties

Add Method of the PageSections Object

This method adds a new element at the end of the collection of page sections.

Visual Basic Syntax

```
Method Add() As PageSection
```

C++ Syntax

```
HRESULT Add(
  IPageSection** result
);
```

Parameters

result

[out, retval] A pointer to **IPageSection*** pointer variable that receives the interface pointer to the returned **PageSection** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageSections
PageSection

PageSection Object (IPageSection Interface)

This object represents one page section. It is an element of the **PageSections** collection.

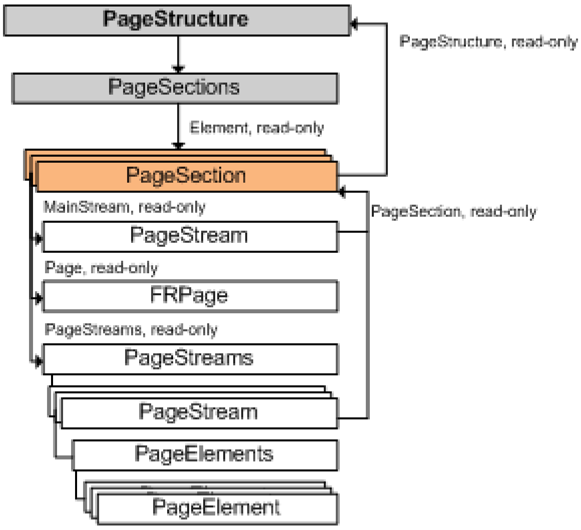
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
IsFirstOnPage	**Boolean**, read-only	Specifies if the section is the first section on page.
IsLastOnPage	**Boolean**, read-only	Specifies if the section is the last section on page.
MainStream	**PageStream**, read-only	Provides access to the main stream of this text section.
Page	**FRPage**, read-only	Provides access to the page that contains this section.
PageStreams	**PageStreams**, read-only	Provides access to the page streams of the section.
PageStructure	**PageStructure**, read-only	Provides access to the page structure, which includes this page section.

Methods

Name	Description
AddFootnote	Creates a footnote in the section.
AddIncut	Creates an incut in the section.
CreateMainStream	Creates the main text stream in the section.
RemoveMainStream	Removes main text stream.

Related objects



Output parameter

This object is the output parameter of the **Item**, **Add** methods of the **PageSections** object.

See also

Working with the Logical Structure of a Document
PageSections
Working with Properties

AddFootnote Method of the PageSection Object

This method creates a footnote in the section and adds it into the **IPageSection::PageStreams** collection.

Visual Basic Syntax

```
Method AddFootnote() As PageStream
```

C++ Syntax

```
HRESULT AddFootnote(
  IPageStream** result
);
```

Parameters

result

[out, retval] A pointer to **IPageStream*** pointer variable that receives the interface pointer to the returned **PageStream** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

Remarks

The method returns the **PageStream** object of the type ST_Footnote. You can receive then the **Footnote** object with the **IPageStream::GetAsFootnote** method.

See also

PageSection

AddIncut Method of the PageSection Object

This method creates an incut in the section and adds it into the **IPageSection::PageStreams** collection.

Visual Basic Syntax

```
Method AddIncut() As PageStream
```

C++ Syntax

```
HRESULT AddIncut(
  IPageStream** result
);
```

Parameters

result

[out, retval] A pointer to **IPageStream*** pointer variable that receives the interface pointer to the returned **PageStream** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

Remarks

The method returns the **PageStream** object of the type ST_Incut. You can receive then the **Incut** object with the **IPageStream::GetAsIncut** method.

See also

PageSection

CreateMainStream Method of the PageSection Object

This method creates the main text stream in the section.

Visual Basic Syntax

```
Method CreateMainStream() As PageStream
```

C++ Syntax

```
HRESULT CreateMainStream(
  IPageStream** result
);
```

Parameters

result

[out] A pointer to **IPageStream*** pointer variable that receives the interface pointer to the returned **PageStream** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageSection

RemoveMainStream Method of the PageSection Object

This method removes main text stream.

Visual Basic Syntax

```
Method RemoveMainStream()
```

C++ Syntax

```
HRESULT RemoveMainStream();
```

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageSection

PageStreams Object (IPageStreams Interface)

This object provides access to the collection of page streams.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

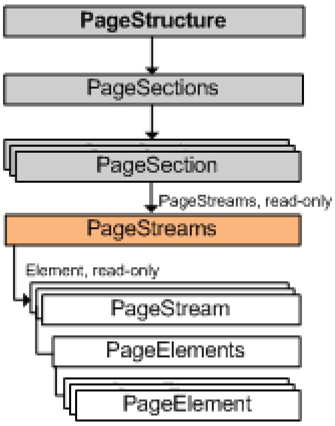
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**PageStream**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Item	Provides access to a single element of the collection.

Related objects



See also

Working with the Logical Structure of a Document
PageStream
Working with Properties

PageStream Object (IPageStream Interface)

This object represents a page stream. It is an element of the **PageStreams** collection. There are 4 types of page streams. Three of them are meaningful components of page logical structure (main text, incut, and footnote), and the excess one is artefact, which contains some garbage.

Properties

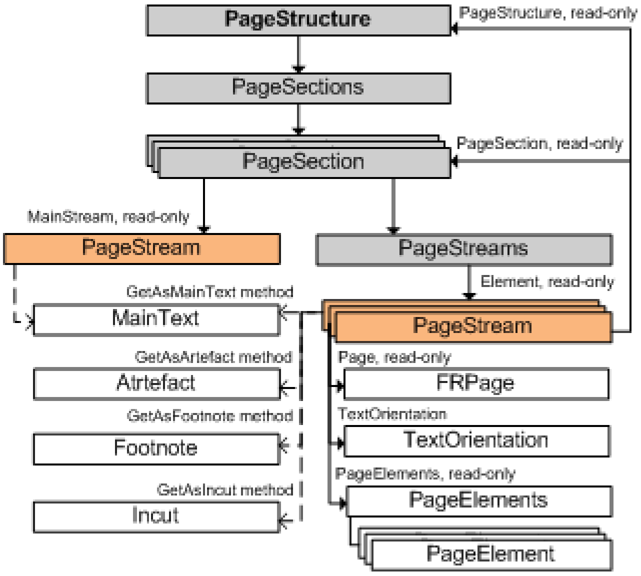
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Page	**FRPage**, read-only	Provides access to the page which contains this page stream.
PageElements	**PageElements**, read-only	Provides access to the page elements of this stream.
PageSection	**PageSection**, read-	Stores the page section, which contains this stream.

	only	
PageStructure	**PageStructure**, read-only	Stores the page structure, which contains this stream.
TextOrientation	**TextOrientation**	Stores the orientation of the text in the stream. **Note:** The property returns a constant object. To change the text orientation, you must first receive an intermediate **TextOrientation** object with the help of the **IEngine::CreateTextOrientation** method, change the necessary parameters, and then assign this object to the property.
Type	**StreamTypeEnum**, read-only	Specifies the type of the page stream.

Methods

Name	**Description**
GetAsArtefact	Returns the page stream as the **Artefact** object. If the page stream is not an artefact, NULL is returned.
GetAsFootnote	Returns the page stream as the **Footnote** object. If the page stream is not a footnote, NULL is returned.
GetAsIncut	Returns the page stream as the **Incut** object. If the page stream is not an incut, NULL is returned.
GetAsMainText	Returns the page stream as the **MainText** object. If the page stream is not the main text of the page, NULL is returned.

Related objects



Output parameter

This object is the output parameter of the following methods and properties:

- **AddArtefact**, **GetArtefact** methods of the **PageStructure** object
- **CreateMainStream**, **AddIncut**, **AddFootnote** methods of the **PageSection** object

See also

Working with the Logical Structure of a Document
PageStreams
Working with Properties

GetAsArtefact Method of the PageStream Object

This method returns the page stream as the **Artefact** object. If the page stream is not an artefact, NULL is returned.

Visual Basic Syntax

```
Method GetAsArtefact() As Artefact
```

C++ Syntax

```
HRESULT GetAsArtefact(
  IArtefact** result
);
```

Parameters

result

[out] A pointer to **IArtefact*** pointer variable that receives the interface pointer to the returned **Artefact** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStream
Artefact

GetAsFootnote Method of the PageStream Object

This method returns the page stream as the **Footnote** object. If the page stream is not a footnote, NULL is returned.

Visual Basic Syntax

```
Method GetAsFootnote() As Footnote
```

C++ Syntax

```
HRESULT GetAsFootnote(
  IFootnote** result
);
```

Parameters

result

[out] A pointer to **IFootnote*** pointer variable that receives the interface pointer to the returned **Footnote** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStream
Footnote

GetAsIncut Method of the PageStream Object

This method returns the page stream as the **Incut** object. If the page stream is not an incut, NULL is returned.

Visual Basic Syntax

```
Method GetAsIncut() As Incut
```

C++ Syntax

```
HRESULT GetAsIncut(
  IIncut** result
);
```

Parameters

result

[out] A pointer to **IIncut*** pointer variable that receives the interface pointer to the returned **Incut** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStream
Incut

GetAsMainText Method of the PageStream Object

This method returns the page stream as the **MainText** object. If the page stream is not the main text of the page, NULL is returned.

Visual Basic Syntax

```
Method GetAsMainText() As MainText
```

C++ Syntax

```
HRESULT GetAsMainText(
  IMainText** result
);
```

Parameters

result

[out] A pointer to **IMainText*** pointer variable that receives the interface pointer to the returned **MainText** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

PageStream
MainText

PageElements Object (IPageElements Interface)

This object provides access to the collection of page elements.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

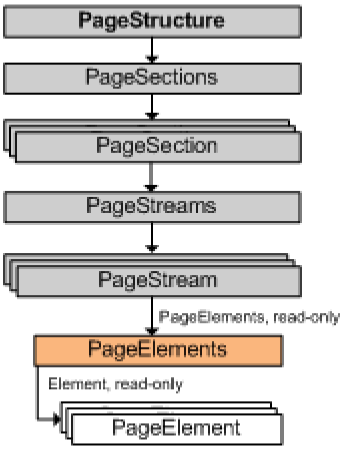
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**PageElement**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
Item	Provides access to a **PageElement** object in the collection.

Related objects



Output parameter

This object is the output parameter of the **PageElements** property of the **PageStream** object.

See also

Working with the Logical Structure of a Document
PageElement
Working with Properties

PageElement Object (IPageElement Interface)

This object represents an element of a recognized page. A page may contain several elements of different types: text, table, picture, and barcode. The type of the element is defined by the **Type** property. The **PageElement** object exposes properties for accessing extended attributes of an element of specific type. The object is an element of the **PageElements** collection.

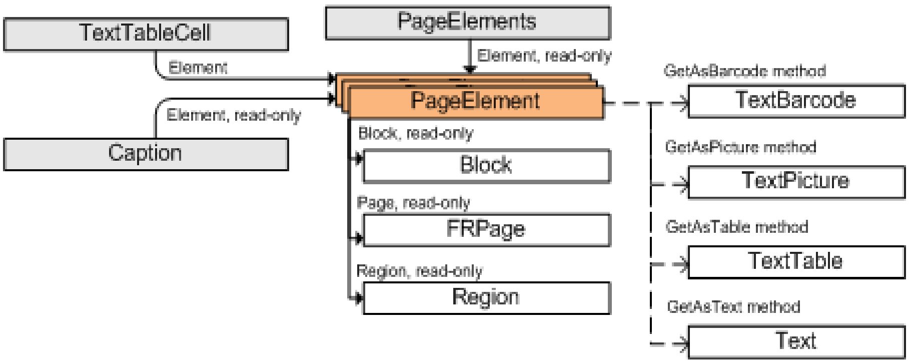
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Block	**Block**, read-only	Provides access to the block which contains this page element.
Id	**String**, read-only	Provides access to the ID of the page element.
Page	**FRPage**, read-only	Provides access to the page which contains this page element.
Region	**Region**, read-only	Stores the region of the block or of the part of block which contains this page element. For the text page element which is located in several blocks, 0 is returned. Coordinates of the region are defined in pixels and counted from the left top corner of the page image. The coordinates can be out of the block region, if the page element is a part of column.
Type	**PageElementTypeEnum**, read-only	Specifies the type of the page element.

Methods

Name	Description
GetAsBarcode	Returns the page element as the **TextBarcode** object. If the page element is not a barcode, NULL is returned.
GetAsPicture	Returns the page element as the **TextPicture** object. If the page element is not a picture, NULL is returned.
GetAsTable	Returns the page element as the **TextTable** object. If the page element is not a table, NULL is returned.
GetAsText	Returns the page element as the **Text** object. If the page element is not a text, NULL is returned.

Related objects



Output parameter

This object is the output parameter of the **Item** method of the **PageElements** object.

See also

Working with the Logical Structure of a Document
PageElements
Working with Properties

GetAsBarcode Method of the PageElement Object

This method returns the page element as the **TextBarcode** object. If the page element is not a barcode, NULL is returned.

Visual Basic Syntax

```
Method GetAsBarcode() As TextBarcode
```

C++ Syntax

```
HRESULT GetAsBarcode(
  ITextBarcode** result
);
```

Parameters

result

[out] A pointer to **ITextBarcode*** pointer variable that receives the interface pointer to the returned **TextBarcode** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Working with the Logical Structure of a Document
PageElement
TextBarcode

GetAsPicture Method of the PageElement Object

This method returns the page element as the **TextPicture** object. If the page element is not a picture, NULL is returned.

Visual Basic Syntax

```
Method GetAsPicture() As TextPicture
```

C++ Syntax

```
HRESULT GetAsPicture(
  ITextPicture** result
);
```

Parameters

result

[out] A pointer to **ITextPicture*** pointer variable that receives the interface pointer to the returned **TextPicture** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Working with the Logical Structure of a Document
PageElement
TextPicture

GetAsTable Method of the PageElement Object

This method returns the page element as the **TextTable** object. If the page element is not a table, NULL is returned.

Visual Basic Syntax

```
Method GetAsTable() As TextTable
```

C++ Syntax

```
HRESULT GetAsTable(
  ITextTable** result
);
```

Parameters

result

[out] A pointer to **ITextTable*** pointer variable that receives the interface pointer to the returned **TextTable** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Working with the Logical Structure of a Document
PageElement
TextTable

GetAsText Method of the PageElement Object

This method returns the page element as the **Text** object. If the page element is not a text, NULL is returned.

Visual Basic Syntax

```
Method GetAsText() As Text
```

C++ Syntax

```
HRESULT GetAsText(
  IText** result
);
```

Parameters

result

[out] A pointer to **IText*** pointer variable that receives the interface pointer to the returned **Text** object.

Return Values

This method has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

Working with the Logical Structure of a Document
PageElement
Text

StreamElementLocationParams Object (IStreamElementLocationParams Interface)

This object allows you to locate a document element or a page element in a column. The parameters are only applied to table, picture, or barcode elements. For text elements (the document elements of the type DET_Paragraph and page elements of the type PET_Text) use the **ParagraphParams** object for corresponding paragraphs.

Positioning of an element (table, picture, or barcode) is performed as follows:

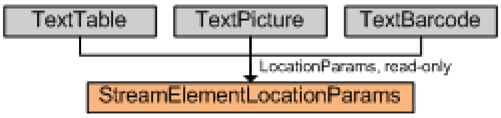
1. First, width and height of the element and element's horizontal position relative to the column are defined.
2. Then positions of all the captions for this element are defined.
3. The surrounding rectangle for the element and all its captions is defined.
4. Finally, the horizontal position of the surrounding rectangle is defined.

Properties

Name	Type	Description
Alignment	**StreamElementAlignmentEnum**	Specifies alignment of a table, picture, or barcode in the column. By default, the value of this property is SEA_None. If the value of this property is not SEA_None, the **LeftIndent** and **RightIndent** properties are set to 0.
Application	**Engine**, read-only	Returns the **Engine** object.

LeftIndent	**Long**	Specifies the indent from the left side of the column to the left side of the element in hundredth parts of point. The value of this property may be negative. By default, the value of this property is 0. If you set the value of this property to nonzero value, the **Alignment** property is set to SEA_None.
RightIndent	**Long**	Specifies the indent from the right side of the column to the right side of the element in hundredth parts of point. The value of this property may be negative. By default, the value of this property is 0. If you set the value of this property to nonzero value, the **Alignment** property is set to SEA_None.
SpaceAfter	**Long**	Specifies the space from the bottom border of the surrounding rectangle of the current element (which includes the element and all its captions) to the top border of the surrounding rectangle of the next element in the stream. The space is measured in hundredth parts of point and must be non-negative. By default, the value of this property is 0.
SpaceBefore	**Long**	Specifies the space from the bottom border of the surrounding rectangle of the previous element (which includes the object and all its captions) in the stream to the top border of the surrounding rectangle of the current element. The space is measured in hundredth parts of point and must be non-negative. By default, the value of this property is 0.

Related objects



See also

TextTable
TextPicture
TextBarcode
Working with Properties

MainText Object (IMainText Interface)

This object exposes method and properties of a main text.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
ColumnsCount	**Long**, read-only	The number of columns in the main text. By default the value of the property is 0.
HasSeparatorBefore	**Boolean**	Specifies if there is a separators before the main text. By default the value of the property is FALSE.
HasSeparatorsBetweenColumns	**Boolean**	Specifies if there are separators between columns. The property makes sense only if the number of columns (the **ColumnsCount** property) is above zero. By default the value of the property is FALSE.
IsRightToLeft	**Boolean**	Specifies if the main text has right-to-left writing direction (like for Hebrew).
LeftColumnBound	**Long**	Specifies the coordinate of the left bound of the specified column in the main text.
RightColumnBound	**Long**	Specifies the coordinate of the right bound of the specified column in the main text.
WhiteGapBefore	**Long**	Specifies the distance from the top bound of this main text to the bottom bound of the main text of the previous section. The property makes sense only if the section is not the first on the page. By default the value of this property is 0.

Methods

Name	Description
AddColumn	Adds a column into the main text.
RemoveColumn	Removes a column with the specified index.

Output parameter

This object is the output parameter of the following methods:

- **GetAsMainText** method of the **DocumentStream** object
- **GetAsMainText** method of the **PageStream** object

See also

Working with the Logical Structure of a Document

AddColumn Method of the MainText Object

This method adds a column into the main text.

Visual Basic Syntax

```
Method AddColumn(
  Long left,
  Long right
)
```

C++ Syntax

```
HRESULT AddColumn(
  long left,
  long right
);
```

Parameters

left

[in] This parameter contains the coordinate of the left bound of the newly added column.

right

[in] This parameter contains the coordinate of the right bound of the newly added column.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

The coordinates are measured in hundredth parts of point from the left border of the page (see **IPageStructure::PageRect**) — for the main text of a page section, or from the left margin of the document (see **IDocumentSection::Margins**) — for the main text of a document section.

The coordinates of the column borders should lay between left and right borders of the page rectangle (for the main text of a page section) and between left and right margins of the document (for the main text of a page section).

See also

MainText
IMainText::RemoveColumn

RemoveColumn Method of the MainText Object

This method removes a column with the specified index.

Visual Basic Syntax

```
Method RemoveColumn(
  Long index
)
```

C++ Syntax

```
HRESULT RemoveColumn(
  long index
);
```

Parameters

index

[in] This parameter contains the index of the column. It must be in range from 0 to **IMainText::ColumnsCount** -1.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

MainText
IMainText::AddColumn

FootnoteSeriesArray Object (IFootnoteSeriesArray Interface)

This object represents a collection of footnote series.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

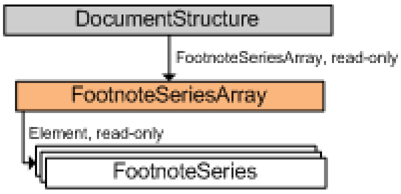
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of series in the array.
Element	**FootnoteSeries**, read-only	Provides access to one footnote series by its index.

Methods

Name	Description
CreateFootnoteSeries	Creates the **FootnoteSeries** object.
DeleteEmptySeries	Removes all empty footnote series from the array.
DeleteAll	Removes all the elements from the footnote series array.
Item	Provides access to a single element of the collection.

Related objects



See also

FootnoteSeries
DocumentStructure
Working with Properties

CreateFootnoteSeries Method of the FootnoteSeriesArray Object

This method allows you to create the **FootnoteSeries** object.

Visual Basic Syntax

```
Method CreateFootnoteSeries() As FootnoteSeries
```

C++ Syntax

```
HRESULT CreateFootnoteSeries(
  IFootnoteSeries** result
);
```

Parameters

result

[out] A pointer to **IFootnoteSeries*** pointer variable that receives the interface pointer to the created **FootnoteSeries** object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FootnoteSeriesArray

DeleteAll Method of the FootnoteSeriesArray Object

This method removes all the elements from the footnote series array.

Visual Basic Syntax

```
Method DeleteAll()
```

C++ Syntax

```
HRESULT DeleteAll();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

If you receive a reference to one of the objects which contain the results of a page syntheses (e.g. one of the subobjects of the **PageStructure** object) and then call the **IFootnoteSeriesArray::DeleteAll** method, the **IFootnote::Series** property for such page will return incorrect value. You should receive a reference to such page again after the method call.

See also

FootnoteSeriesArray
IFootnoteSeriesArray::DeleteEmptySeries

DeleteEmptySeries Method of the FootnoteSeriesArray Object

This method removes all empty footnote series from the array.

Visual Basic Syntax

```
Method DeleteEmptySeries()
```

C++ Syntax

```
HRESULT DeleteEmptySeries();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FootnoteSeriesArray
IFootnoteSeries::DeleteAll

FootnoteSeries Object (IFootnoteSeries Interface)

This object stores the parameters of one series of footnotes.

Properties

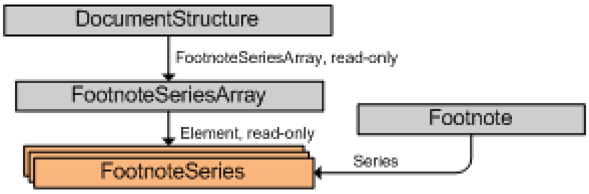
Name	Type	Description

Application	**Engine**, read-only	Returns the **Engine** object.
HasSeparator	**Boolean**	Specifies whether the footnotes are separated from the text with a horizontal line.
IsContinuousNumbering	**Boolean**	If the value of this property is TRUE, the continuous numbering is used. If the value of this property is FALSE, the footnotes numbering starts from 1 on each page.
IsNumberingWithSuperscript	**Boolean**	Specifies whether superscript characters are used for the footnote numbering.
NumberingType	**FootnoteNumberingTypeEnum**	Specifies the type of the footnote numbering.
PositionInDocument	**FootnotePositionInDocumentTypeEnum**, read-only	Returns the position of the footnote in the document. To change the footnote position, use the **SetPosition** method.
PositionOnPage	**FootnotePositionOnPageTypeEnum**, read-only	Returns the position of the footnote relative to the column with the anchor. To change the footnote position, use the **SetPosition** method.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
SetPosition	Sets the position of the footnote.

Related objects



Output parameter

This object is the output parameter of the **CreateFootnoteSeries** methods of the **FootnoteSeriesArray** object.

See also

FootnoteSeriesArray
Working with Properties

SetPosition Method of the FootnoteSeries Object

This method sets the position of the footnote. The method affects the values of the **PositionInDocument** and **PositionOnPage** properties of the **FootnoteSeries** object.

Visual Basic Syntax

```
Method SetPosition(
   position       As FootnotePositionInDocumentTypeEnum,
   columnPosition As FootnotePositionOnPageTypeEnum
 )
```

C++ Syntax

```
HRESULT SetPosition(
   FootnotePositionInDocumentTypeEnum position,
   FootnotePositionOnPageTypeEnum     columnPosition
 );
```

Parameters

position

[in] This variable contains the position of the footnote in the document. See description of the **FootnotePositionInDocumentTypeEnum**.

columnPosition

[in] This variable contains the position of the footnote relative to the column with the anchor. See description of the **FootnotePositionOnPageTypeEnum**.

Return Values

This method has no specific return values. It returns the standard return values of the ABBYY FineReader Engine functions.

Remark

Not all combinations of the **FootnotePositionInDocumentTypeEnum** and **FootnotePositionOnPageTypeEnum** constants are allowed. The following combinations are prohibited:

- FPDT_PageEnd and FPPT_CurrentColumn
- FPDT_SectionEnd and FPPT_CurrentColumn
- FPDT_DocumentEnd and FPPT_LastColumn
- FPDT_DocumentEnd and FPPT_CurrentColumn

See also

FootnoteSeries

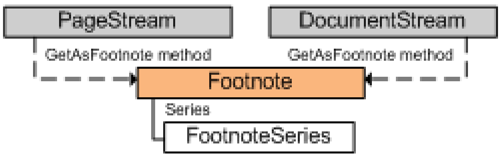
Footnote Object (IFootnote Interface)

This object exposes properties of a footnote.

Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
HasHead	**Boolean**	Specifies if the footnote is a part of another footnote located on several pages, and has the beginning on another page. The property makes sense only for a page stream. The default value is FALSE.
HasTail	**Boolean**	Specifies if the footnote is a part of another footnote located on several pages, and has the end on another page. The property makes sense only for a page stream. The default value is FALSE.
LeftBound	**Long**	Specifies the coordinate of the left bound of the column where the footnote is located. By default the value of the property is -1, which means that the coordinate is undefined. The coordinates are measured in hundredth parts of point from the left top corner of the page region (see **IPageStructure::PageRect**) — for the page stream, or left top corner of the region of the page with margins (see **IDocumentSection::Margins**) — for the document stream.
Number	**Long**	Specifies the number of the footnote in the corresponding series. Footnote numbering starts with 1. By default the value of the property is -1, which means that the number is undefined.
RightBound	**Long**	Specifies the coordinate of the right bound of the column where the footnote is located. By default the value of the property is -1, which means that the coordinate is undefined. The coordinates are measured in hundredth parts of point from the left top corner of the page region (see **IPageStructure::PageRect**) — for the page stream, or left top corner of the region of the page with margins (see **IDocumentSection::Margins**) — for the document stream.
Series	**FootnoteSeries**	Provides access to the footnote series which corresponds to this footnote. **Note:** The property returns a constant object. To change the footnote series which corresponds to this footnote, you must first receive an intermediate **FootnoteSeries** object with the help of the **IFootnoteSeriesArray::CreateFootnoteSeries** method, change the necessary parameters, and then assign this object to the property.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **GetAsFootnote** method of the **DocumentStream** object
- **GetAsFootnote** method of the **PageStream** object

See also

Working with the Logical Structure of a Document
Working with Properties

Incut Object (IIncut Interface)

This object exposes method and properties of an incut.

Properties

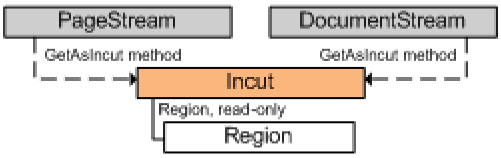
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColor	**Long**	Specifies the background color. By default the value of this property is 0xFEFFFFFF, which means that the color is transparent. **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component.
Borders	**Long**	Describes the borders of the incut frame as a bitwise OR combination of the **BF_** prefixed flags.
BottomMargin	**Long**	Specifies the margin from the bottom border of the incut frame to the text of the column below the incut (if the incut intersects a column). By default the value is LONG_MIN.
HorizontalOffset	**Long**, read-only	Stores the horizontal offset of the incut frame from some object on the page.
LeftMargin	**Long**	Specifies the margin from the left border of the incut frame to the text of the column to the left of the incut (if the incut intersects a column). By default the value is LONG_MIN.
Region	**Region**, read-only	Stores the region of the incut. The region is specified in coordinates measured in hundredth parts of point from the left top corner of the page region (see **IPageStructure::PageRect**) — for the page stream, or left top corner of the region of the page with margins (see **IDocumentSection::Margins**) — for the document stream.
RightMargin	**Long**	Specifies the margin from the right border of the incut frame to the text of the column to the right of the incut (if the incut intersects a column). By default the value is LONG_MIN.
TextWrapping	**TextWrappingEnum**	Specifies the text wrapping around the incut. By default the value is TW_Undefined.
TopMargin	**Long**	Specifies the margin from the top border of the incut frame to the text of the column above the incut (if the incut intersects a column). By default the value is LONG_MIN.
VerticalOffset	**Long**, read-only	Stores the vertical offset of the incut frame from some object on the page.

Methods

Name	Description

SetVerticalOffsetFromSectionTop	Sets the vertical offset of the incut frame from the section.
SetVerticalOffsetFromParagraph	Sets the vertical offset of the incut frame from the paragraph.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **GetAsIncut** method of the **DocumentStream** object
- **GetAsIncut** method of the **PageStream** object

See also

Working with the Logical Structure of a Document
Working with Properties

HorizontalOffset Property of the Incut Object

This property returns the horizontal offset of the incut frame from different objects on the page. The type of the object is defined by the **FrameHorizontalReferenceEnum** constant. Horizontal offset is generally measured from the left border of the object for texts with left-to-right writing direction, and from the right border — for texts with right-to-left writing direction. See details in the description of the **FrameHorizontalReferenceEnum** constants.

Visual Basic Syntax

```
Property HorizontalOffset(type As FrameHorizontalReferenceEnum) As Long
  read-only
```

C++ Syntax

```
HRESULT get_HorizontalOffset(
   FrameHorizontalReferenceEnum type,
   long*                        result
);
```

Parameters

type

[in] This parameter specifies the object on the page to measure offset from. See the description of the **FrameHorizontalReferenceEnum** constants.

result

[out, retval] A pointer to **long** variable that receives the value of this property.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Incut
Working with Properties

VerticalOffset Property of the Incut Object

This property returns the vertical offset of the incut frame from different objects on the page. The type of the object is defined by the **FrameVerticalReferenceEnum** constant.

Visual Basic Syntax

```
Property VerticalOffset(type As FrameVerticalReferenceEnum) As Long
```

```
  read-only
```

C++ Syntax

```
HRESULT get_VerticalOffset(
   FrameVerticalReferenceEnum type,
   long*                      result
);
```

Parameters

type

[in] This parameter specifies the object on the page to measure offset from. See the description of the **FrameVerticalReferenceEnum** constants.

result

[out, retval] A pointer to **long** variable that receives the value of the offset.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Incut
Working with Properties

SetVerticalOffsetFromParagraph Method of the Incut Object

This method sets the vertical offset of the incut frame from the paragraph.

Visual Basic Syntax

```
Method SetVerticalOffsetFromParagraph(
  value As Long
)
```

C++ Syntax

```
HRESULT SetVerticalOffsetFromParagraph(
  long value
);
```

Parameters

value

[in] This parameter specifies the vertical offset of the incut frame from the paragraph.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Incut

SetVerticalOffsetFromSectionTop Method of the Incut Object

This method sets the vertical offset of the incut frame from the section top.

Visual Basic Syntax

```
Method SetVerticalOffsetFromSectionTop(
  value As Long
)
```

C++ Syntax

```
HRESULT SetVerticalOffsetFromSectionTop(
  long value
);
```

Parameters

value

[in] This parameter specifies the vertical offset of the incut frame from the section top.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Incut

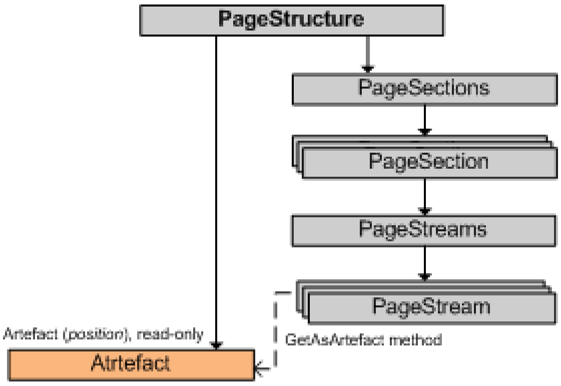
Artefact Object (IArtefact Interface)

This object exposes properties of an artefact. Artefact is an object on page which contains some garbage. Artefact is a type of page stream. Usually a page stream is assigned the artefact type, if this stream is unnecessary in page logical structure and cannot be assigned any other meaningful type.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Region	**Region**, read-only	Stores the region of the artefact in hundredth parts of point. The region is specified in coordinates relative to the left top corner of the page region.

Related objects



Output parameter

This object is the output parameter of **GetAsArtefact** method of the **PageStream** object.

See also

Working with Properties

TextPicture Object (ITextPicture Interface)

This object provides access to specific properties of a picture in a logic structure of a document.

Properties

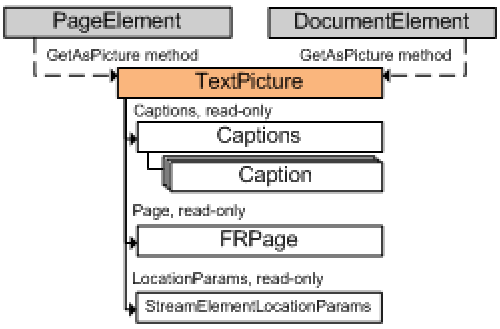
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Captions	**Captions**, read-only	Returns the collection of captions of the picture. If the picture has no captions, the property returns 0.
ColumnNumber	**Long**, read-only	Specifies the number of the column which contains this picture. For the picture located in several columns, -1 is returned.
HasCaptions	**Boolean**, read-only	Specifies if the picture has captions.
IsBackgroundPicture	**Boolean**	Specifies if the picture is a background picture. If this property is

		TRUE, actually the picture does not exist, and the object is only used as a stub for some background picture.
IsInlinePicture	**Boolean**	Indicates that the picture is embedded in text. This property may be set to TRUE during recognition of the image. It indicates that the picture is inline and need not to be displayed as a separate block. In this case the so called "*Object replacement character*" appears in the recognized text instead of this picture. Unicode for this character is 0xFFFC. Embedded pictures are displayed in **Editor** window of ABBYY FineReader in this way.
LocationParams	**StreamElementLocationParams**, read-only	Stores the parameters which define the position of the picture relative to the column (the **ColumnNumber** property).
Page	**FRPage**, read-only	Returns a reference to the page that contains picture.

Methods

Name	Description
DeleteCaptions	Deletes all the captions of the picture.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **GetAsPicture** method of the **DocumentElement** object
- **GetAsPicture** method of the **PageElement** object

See also

Working with the Logical Structure of a Document
PageElement
DocumentElement
Working with Properties

DeleteCaptions Method of the TextPicture Object

This method deletes all the captions of the picture.

Visual Basic Syntax

```
Method DeleteCaptions()
```

C++ Syntax

```
HRESULT DeleteCaptions();
```

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextPicture

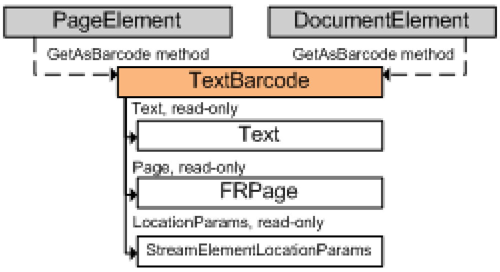
TextBarcode Object (ITextBarcode Interface)

This object provides access to specific properties of a barcode in a logic structure of a document.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
ColumnNumber	**Long**, read-only	Specifies the number of the column which contains this barcode. For the barcode located in several columns, -1 is returned.
LocationParams	**StreamElementLocationParams**, read-only	Stores the parameters which define the position of the barcode relative to the column (the **ColumnNumber** property).
Page	**FRPage**, read-only	Returns a reference to the page that contains the barcode.
Text	**Text**, read-only	Stores the text of the barcode. **Note:** We recommend working with text of barcodes via the layout (the **IBarcodeBlock::BarcodeText** or **IBarcodeBlock::Text** property) as this way is more suitable for barcodes and does not require synthesis.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **GetAsBarcode** method of the **PageElement** object
- **GetAsBarcode** method of the **DocumentElement** object

See also

Working with the Logical Structure of a Document
PageElement
DocumentElement

TextTable Object (ITextTable Interface)

This object provides access to specific properties of a table in a logic structure of a document.

Properties

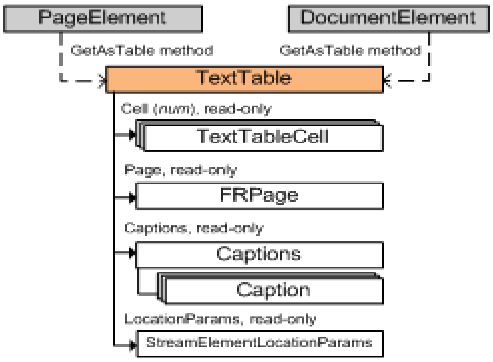
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Captions	**Captions**, read-only	Returns the collection of captions of the table. If the table has no captions, the property returns 0.
Cell	**TextTableCell**, read-only	Provides access to the cell by its index.
CellsCount	**Long**, read-only	Stores the number of cells in the table.
ColumnsCount	**Long**, read-only	Stores the number of columns in the table.
ColumnNumber	**Long**, read-only	Specifies the number of the text column which contains this table. For the table located in several columns, -1 is returned.
HasCaptions	**Boolean**, read-only	Specifies if the table has captions.

Height	**Long**, read-only	Stores the height of the table in in hundredth parts of point.
HSeparatorPos	**Long**, read-only	Returns the position of the specified horizontal separator. The position is a distance from the upper border of the table to the specified separator measured in hundredth parts of point.
HSeparatorType	**TextTableSeparatorTypeEnum**, read-only	Returns the type of the specified horizontal separator.
HSeparatorWidth	**Long**, read-only	Returns the width of the specified horizontal separator.
LocationParams	**StreamElementLocationParams**, read-only	Stores the parameters which define the position of the table relative to the text column (the **ColumnNumber** property).
Page	**FRPage**, read-only	Returns a reference to the page that contains table.
RowsCount	**Long**, read-only	Stores the number of rows in the table.
VSeparatorPos	**Long**, read-only	Returns the position of the specified vertical separator. The position is a distance from the left border of the table to the specified separator measured in hundredth parts of point.
VSeparatorType	**TextTableSeparatorTypeEnum**, read-only	Returns the type of the specified vertical separator.
VSeparatorWidth	**Long**, read-only	Returns the width of the specified vertical separator.
Width	**Long**, read-only	Stores the width of the table in in hundredth parts of point.

Methods

Name	**Description**
CreateCell	Returns a reference to the **TextTableCell** object.
DeleteCaptions	Deletes all the captions of the table.
DeleteHSeparator	Deletes the specified horizontal separator.
DeleteVSeparator	Deletes the specified vertical separator.
GetCellByPos	Returns a cell that corresponds to the specified point in base coordinates of the table grid.
GetCellIndexByPos	Returns an index of the cell that corresponds to the specified point in base coordinates of the table grid.
InsertHSeparator	Adds a new horizontal separator and splits the specified row into two rows.
InsertVSeparator	Adds a new vertical separator and splits the specified column into two columns.
SetHSeparator	Sets the width and type of the specified horizontal separator.
SetHSeparatorPos	Sets the position of the specified horizontal separator.
SetVSeparator	Sets the width and type of the specified vertical separator.
SetVSeparatorPos	Sets the position of the specified vertical separator.

Related objects



Output parameter

This object is the output parameter of the following methods:

- **GetAsTable** method of the **DocumentElement** object
- **GetAsTable** method of the **PageElement** object

See also

Working with the Logical Structure of a Document
Working with Properties

Cell Property of the TextTable Object

This property provides access to the cell by its index.

Visual Basic Syntax

```
Property Cell(num As Long) As TextTableCell
  read-only
```

C++ Syntax

```
HRESULT get_Cell(
   long            num,
   ITextTableCell** result
);
```

Parameters

num

[in] This variable contains the number of the cell in the table. Cells are numbered in the order of creation. Must be in range from 0 to **ITextTable::CellsCount**-1.

result

[out, retval] A pointer to **ITextTableCell*** pointer variable that receives the interface pointer of the output **TextTableCell** object.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

HSeparatorPos Property of the TextTable Object

This property returns the position of the specified horizontal separator. The position is a distance from the upper border of the table to the specified separator measured in hundredth parts of point.

Visual Basic Syntax

```
Property HSeparatorPos(
  row    As Long
) As Long
  read-only
```

C++ Syntax

```
HRESULT get_HSeparatorPos(
  long row,
  long result
);
```

Parameters

row

[in] This variable specifies the index of the row which upper border separator position is requested. Must be in range from 0 to **ITextTable::RowsCount**.

result

[out, retval] This variable returns the distance from the upper border of the table to the specified separator measured in hundredth parts of point.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

HSeparatorType Property of the TextTable Object

This property returns the type of the specified horizontal separator.

Visual Basic Syntax

```
Property HSeparatorType(
  column As Long,
  row    As Long,
) As TextTableSeparatorTypeEnum
  read-only
```

C++ Syntax

```
HRESULT get_HSeparatorType(
  long                       column,
  long                       row,
  TextTableSeparatorTypeEnum result
);
```

Parameters

column

[in] This variable specifies the index of the column which contains the separator. Must be in range from 0 to **ITextTable::ColumnsCount**-1.

row

[in] This variable specifies the index of the row which upper border separator is requested. Must be in range from 0 to **ITextTable::RowsCount**.

result

[out, retval] This variable returns the type of the separator. It may be one of the constants from the **TextTableSeparatorTypeEnum** enumeration.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

HSeparatorWidth Property of the TextTable Object

This property returns the width of the specified horizontal separator.

Visual Basic Syntax

```
Property HSeparatorWidth(
  column As Long,
  row    As Long,
) As Long
```

```
  read-only
```

C++ Syntax

```
HRESULT get_HSeparatorWidth(
  long column,
  long row,
  long result
);
```

Parameters

column

[in] This variable specifies the index of the column which contains the separator. Must be in range from 0 to **ITextTable::ColumnsCount**-1.

row

[in] This variable specifies the index of the row which upper border separator is requested. Must be in range from 0 to **ITextTable::RowsCount**.

result

[out, retval] This variable returns the width of the separator.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

VSeparatorPos Property of the TextTable Object

This property returns the position of the specified vertical separator. The position is a distance from the left border of the table to the specified separator measured in hundredth parts of point.

Visual Basic Syntax

```
Property VSeparatorPos(
  column As Long
) As Long
  read-only
```

C++ Syntax

```
HRESULT get_VSeparatorPos(
  long column,
  long result
);
```

Parameters

column

[in] This variable specifies the index of the column which left border separator position is requested. Must be in range from 0 to **ITextTable::ColumnsCount**.

result

[out, retval] This variable returns the distance from the left border of the table to the specified separator measured in hundredth parts of point.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

VSeparatorType Property of the TextTable Object

This property returns the type of the specified vertical separator.

Visual Basic Syntax

```
Property VSeparatorType(
  column As Long,
  row    As Long,
) As TextTableSeparatorTypeEnum
  read-only
```

C++ Syntax

```
HRESULT get_VSeparatorType(
  long                       column,
  long                       row,
  TextTableSeparatorTypeEnum result
);
```

Parameters

column

[in] This variable specifies the index of the column which left border separator is to be set. Must be in range from 0 to **ITextTable::ColumnsCount**.

row

[in] This variable specifies the index of the row which contains the separator to be set. Must be in range from 0 to **ITextTable::RowsCount**-1.

result

[out, retval] This variable returns the type of the separator. It may be one of the constants from the **TextTableSeparatorTypeEnum** enumeration.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

VSeparatorWidth Property of the TextTable Object

This property returns the width of the specified vertical separator.

Visual Basic Syntax

```
Property VSeparatorWidth(
  column As Long,
  row    As Long,
) As Long
  read-only
```

C++ Syntax

```
HRESULT get_VSeparatorWidth(
  long column,
  long row,
  long result
);
```

Parameters

column

[in] This variable specifies the index of the column which left border separator is to be set. Must be in range from 0 to **ITextTable::ColumnsCount**.

row

[in] This variable specifies the index of the row which contains the separator to be set. Must be in range from 0 to **ITextTable::RowsCount**-1.

result

[out, retval] This variable returns the width of the separator.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable
Working with Properties

CreateCell Method of the TextTable Object

This method returns a reference to the **TextTableCell** object.

Visual Basic Syntax

```
Method CreateCell(
  position As FRRectangle,
  result   As TextTableCell
)
```

C++ Syntax

```
HRESULT SeparateHorz(
  IFRRectangle*    position,
  ITextTableCell** result
);
```

Parameters

position

[in] This variable refers to **FRRectangle** object which contains the rectangle of a newly created table cell in a base grid.

result

[out, retval] A pointer to the **ITextTableCell*** pointer variable that receives the interface pointer to the **TextTableCell** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Cell coordinates in a base grid are the coordinates of its left top corner in that grid. By the base grid here we assume the grid formed by table borders and separators. Each vertical separator increments the horizontal coordinate by one, and each horizontal separator increments the vertical coordinate by one. Coordinate axes are oriented from left to right and from top to bottom. Pixel coordinates relative to image must lay inside the table block's region otherwise base coordinate value returned will be -1.

See also

TextTable

DeleteCaptions Method of the TextTable Object

This method deletes all the captions of the table.

Visual Basic Syntax

```
Method DeleteCaptions()
```

C++ Syntax

```
HRESULT DeleteCaptions();
```

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

DeleteHSeparator Method of the TextTable Object

This method deletes the specified horizontal separator. All the segments of the separator must have the TTST_CellSeparator type, i.e. each cell above the separator must be merged with the corresponding cell under the separator.

Visual Basic Syntax

```
Method DeleteHSeparator(
  row    As Long,
  result As Boolean
)
```

C++ Syntax

```
HRESULT DeleteHSeparator(
  long          row,
  VARIANT_BOOL* result
);
```

Parameters

row

[in] This variable specifies the index of the row which upper border separator is to be deleted. Must be in range from 1 to **ITextTable::RowsCount**.

result

[out, retval] This variable returns TRUE, if the separator successfully deleted. It returns FALSE, if not all the segments of the separator have the TTST_CellSeparator type, and therefore the separator cannot be deleted.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

DeleteVSeparator Method of the TextTable Object

This method deletes the specified vertical separator. All the segments of the separator must have the TTST_CellSeparator type, i.e. each cell to the left of the separator must be merged with the corresponding cell to the right of the separator.

Visual Basic Syntax

```
Method DeleteVSeparator(
  column As Long,
  result As Boolean
)
```

C++ Syntax

```
HRESULT DeleteVSeparator(
  long          column,
  VARIANT_BOOL* result
);
```

Parameters

column

[in] This variable specifies the index of the column which left border separator is to be deleted. Must be in range from 1 to **ITextTable::ColumnsCount**.

result

[out, retval] This variable returns TRUE, if the separator successfully deleted. It returns FALSE, if not all the segments of the separator have the TTST_CellSeparator type, and therefore the separator cannot be deleted.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

GetCellIndexByPos Method of the TextTable Object

This method returns an index of the cell that corresponds to the specified point in base coordinates of the table grid.

Visual Basic Syntax

```
Method FindCellIndex(
  x As Long,
  y As Long
) As Long
```

C++ Syntax

```
HRESULT FindCellIndex(
  long  x,
  long  y,
  long* result
);
```

Parameters

x

[in] This variable specifies horizontal coordinate of the point (defined on vertical separators).

y

[in] This variable specifies vertical coordinate of the point (defined on horizontal separators).

result

[out, retval] A pointer to **long** variable that receives the index of the cell.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Cell coordinates in a base grid are the coordinates of its left top corner in that grid. By the base grid here we assume the grid formed by table borders and separators. Each vertical separator increments the horizontal coordinate by one, and each horizontal separator increments the vertical coordinate by one. Coordinate axes are oriented from left to right and from top to bottom.

The point specified should not exceed the table grid otherwise an error code is returned.

See also

TextTable

GetCellByPos Method of the TextTable Object

This method returns a cell that corresponds to the specified point in base coordinates of the table grid.

Visual Basic Syntax

```
Method GetCellByPos(
  x As Long,
  y As Long
) As TextTableCell
```

C++ Syntax

```
HRESULT GetCellByPos(
  long  x,
  long  y,
  ITextTableCell** result
);
```

Parameters

x

[in] This variable specifies horizontal coordinate of the point (defined on vertical separators).

y

[in] This variable specifies vertical coordinate of the point (defined on horizontal separators).

result

[out, retval] A pointer to the **ITextTableCell*** pointer variable that receives the interface pointer to the output **TextTableCell** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Cell coordinates in a base grid are the coordinates of its left top corner in that grid. By the base grid here we assume the grid formed by table borders and separators. Each vertical separator increments the horizontal coordinate by one, and each horizontal separator increments the vertical coordinate by one. Coordinate axes are oriented from left to right and from top to bottom. Pixel coordinates relative to image must lay inside the table block's region otherwise base coordinate value returned will be -1.

The point specified should not exceed the table grid otherwise an error code is returned.

See also

TextTable

InsertHSeparator Method of the TextTable Object

This method adds a new horizontal separator into the collection of horizontal separators and splits the specified row into two rows. The newly added separator has the TTST_CellSeparator type. Positions of the cells in the base grid are recalculated.

Visual Basic Syntax

```
Method InsertHSeparator(
  row As Long,
  pos As Long
)
```

C++ Syntax

```
HRESULT InsertHSeparator(
  long row,
  long pos
);
```

Parameters

row

[in] This variable specifies the index of the row which is to be split. Must be in range from 0 to **ITextTable::RowsCount** - 1.

pos

[in] This variable specifies the distance from the upper border of the table to the newly added separator measured in hundredth parts of point. The distance must be above than **ITextTable::HSeparatorPos**(*row*) and less than **ITextTable::HSeparatorPos**(*row* + 1).

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

InsertVSeparator Method of the TextTable Object

This method adds a new vertical separator into the collection of vertical separators and splits the specified column into two columns. The newly added separator has the TTST_CellSeparator type. Positions of the cells in the base grid are recalculated.

Visual Basic Syntax

```
Method InsertVSeparator(
  column As Long
  pos    As Long
)
```

C++ Syntax

```
HRESULT InsertVSeparator(
  long column,
  long pos
);
```

Parameters

column

[in] This variable specifies the index of the column which is to be split. Must be in range from 0 to **ITextTable::ColumnsCount** - 1.

pos

[in] This variable specifies the distance from the left border of the table to the newly added separator measured in hundredth parts of point. The distance must be above than **ITextTable::VSeparatorPos**(*column*) and less than **ITextTable::VSeparatorPos**(*column* + 1).

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

SetHSeparator Method of the TextTable Object

This method allows you to set the width and type of the specified horizontal separator.

Visual Basic Syntax

```
Method SetHSeparator(
  column         As Integer,
  row            As Integer,
  type           As TextTableSeparatorTypeEnum,
  separatorWidth As Long
)
```

C++ Syntax

```
HRESULT SetHSeparator(
  int                        column,
  int                        row,
  TextTableSeparatorTypeEnum type,
  long                       separatorWidth
);
```

Parameters

column

[in] This variable specifies the index of the column which contains the separator to be set. Must be in range from 0 to **ITextTable::ColumnsCount**-1.

row

[in] This variable specifies the index of the row which upper border separator is to be set. Must be in range from 0 to **ITextTable::RowsCount**.

type

[in] This variable specifies the new type of the separator. It may be set to one of the constants from the **TextTableSeparatorTypeEnum** enumeration.

separatorWidth

[in] This variable specifies the new width of the separator.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

SetHSeparatorPos Method of the TextTable Object

This method sets the position of the specified horizontal separator. The position is a distance from the upper border of the table to the specified separator measured in hundredth parts of point.

Visual Basic Syntax

```
Method SetHSeparatorPos(
  row As Long,
  pos As Long
)
```

C++ Syntax

```
HRESULT SetHSeparatorPos(
  long row,
  long pos
);
```

Parameters

row

[in] This variable specifies the index of the row which upper border separator position is set. Must be in range from 1 to **ITextTable::RowsCount**-1.

pos

[in] This variable specifies the distance from the upper border of the table to the specified separator measured in hundredth parts of point. The distance must be above than **ITextTable::HSeparatorPos**(*row* - 1) and less than **ITextTable::HSeparatorPos**(*row* + 1).

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

SetVSeparator Method of the TextTable Object

This method allows you to set the width and type of the specified vertical separator.

Visual Basic Syntax

```
Method SetVSeparator(
  column         As Integer,
  row            As Integer,
  type           As TextTableSeparatorTypeEnum,
  separatorWidth As Long
)
```

C++ Syntax

```
HRESULT SetVSeparator(
  int                        column,
  int                         row,
  TextTableSeparatorTypeEnum type,
  long                        separatorWidth
);
```

Parameters

column

[in] This variable specifies the index of the column which left border separator is to be set. Must be in range from 0 to **ITextTable::ColumnsCount**.

row

[in] This variable specifies the index of the row which contains the separator to be set. Must be in range from 0 to **ITextTable::RowsCount**-1.

type

[in] This variable specifies the new type of the separator. It may be set to one of the constants from the **TextTableSeparatorTypeEnum** enumeration.

separatorWidth

[in] This variable specifies the new width of the separator.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

SetVSeparatorPos Method of the TextTable Object

This method sets the position of the specified vertical separator. The position is a distance from the left border of the table to the specified separator measured in hundredth parts of point.

Visual Basic Syntax

```
Method SetVSeparatorPos(
  column As Long
  pos    As Long
)
```

C++ Syntax

```
HRESULT SetVSeparatorPos(
  long column,
  long pos
);
```

Parameters

column

[in] This variable specifies the index of the column which left border separator position is requested. Must be in range from 1 to **ITextTable::ColumnsCount** - 1.

pos

[in] This variable specifies the distance from the left border of the table to the specified separator measured in hundredth parts of point. The distance must be above than **ITextTable::VSeparatorPos**(*column* - 1) and less than **ITextTable::VSeparatorPos**(*column* + 1).

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

TextTable

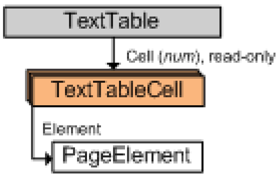
TextTableCell Object (ITextTableCell Interface)

This object provides access to specific properties of a table cell in a logic structure of a document.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColor	**Long**	Specifies the background color of the cell. **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component.
Element	**PageElement**	Provides access to the page element which is contained in the cell.
EndColumnNumber	**Long**, read-only	Stores the number of the last column, which contains the cell. If the cell is not merged, this property is equal to the **StartColumnNumber** property.
EndRowNumber	**Long**, read-only	Stores the number of the last row, which contains the cell. If the cell is not merged, this property is equal to the **StartRowNumber** property.
StartColumnNumber	**Long**, read-only	Stores the number of the first column, which contains the cell. If the cell is not merged, this property is equal to the **EndColumnNumber** property.
StartRowNumber	**Long**, read-only	Stores the number of the first row, which contains the cell. If the cell is not merged, this property is equal to the **EndRowNumber** property.
VertAlignment	**TableCellVertAlignmentEnum**	Specifies the vertical alignment of the table cell.

Related objects



Output parameter

This object is the output parameter of the **CreateCell**, **GetCellByPos** methods of the **TextTable** object.

See also

Working with Layout and Blocks
Working with Properties

Captions Object (ICaptions Interface)

This object provides access to the collection of captions of a table or picture. Besides standard collection methods and properties, it contains the **CreateCaption** method that allows you to create a new caption to the object.

Important! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

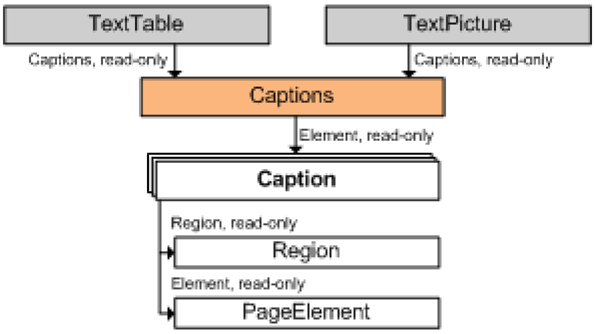
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**Caption**, read-only	Provides access to a single element of the collection.

Methods

Name	Description

CreateCaption	Creates a new caption to the object.
DeleteAll	Deletes all the captions from the collection.
Item	Provides access to a single element of the collection.

Related objects



See also

Working with the Logical Structure of a Document
Caption
TextTable
TextPicture
Working with Properties

CreateCaption Method of the Captions Object

This method creates a new caption to the object (table or picture).

Visual Basic Syntax

```
Method CreateCaption(
  position   As CaptionPositionEnum,
  text       As PageElement,
) As Caption
```

C++ Syntax

```
HRESULT CreateCaption(
  CaptionPositionEnum position,
  IPageElement*       text,
  ICaption**          result
);
```

Parameters

position

[in] This variable of **CaptionPositionEnum** type specifies the position of the new caption relative to the object.

text

[in] This variable refers to the **PageElement** object which contains the text of the new caption. The type of the page element must be PET_Text.

result

[out, retval] A pointer to **ICaption*** pointer variable that receives the interface pointer of the newly created **Caption** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Captions
Caption

DeleteAll Method of the Captions Object

This method deletes all the captions from the collection.

Visual Basic Syntax

```
Method DeleteAll()
```

C++ Syntax

```
HRESULT DeleteAll();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Captions
Caption

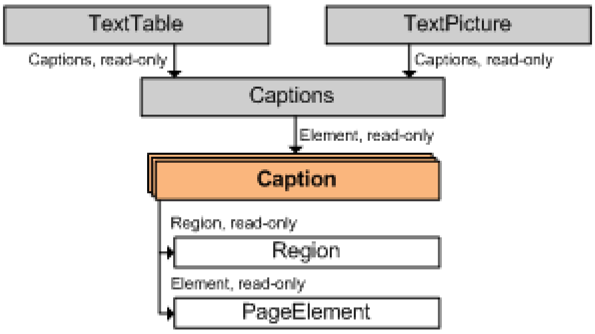
Caption Object (ICaption Interface)

This object provides access to specific properties of a table or picture caption. It is an element of the collection of captions (**Captions** object).

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Element	**PageElement**, read-only	Provides access to the text of the caption. The text is stored as the page element of type PET_Text.
Position	CaptionPositionEnum, read-only	Stores the position of the caption relative to the table or picture which has this caption.
Region	**Region**, read-only	Stores the region of the caption in hundredth parts of point. The region is specified in coordinates relative to the left top corner of the surrounding rectangle of the object which has this caption. Surrounding rectangle includes the object and all its captions.

Related objects



Output parameter

This object is the output parameter of the **Item**, **CreateCaption** methods of the **Captions** object.

See also

Working with the Logical Structure of a Document
Captions
Working with Properties

RunningTitleSeriesArray Object (IRunningTitleSeriesArray Interface)

This object represents a collection of running title series.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

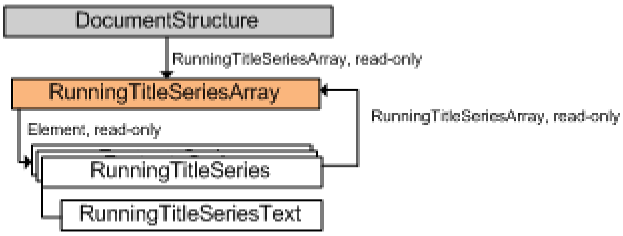
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of series in the array.
Element	**RunningTitleSeries**, read-only	Provides access to one running title series by its index.

Methods

Name	Description
CreateRunningTitlesSeries	Creates the **RunningTitleSeries** object.
DeleteEmptySeries	Removes all empty running title series from the array.
DeleteAll	Removes all the elements from the running title series array.
Item	Provides access to a single element of the collection.

Related objects



See also

Working with the Logical Structure of a Document
RunningTitleSeries
DocumentStructure
Working with Properties

CreateRunningTitleSeries Method of the RunningTitleSeriesArray Object

This method allows you to create the **RunningTitleSeries** object.

Visual Basic Syntax

```
Method CreateRunningTitleSeries() As RunningTitleSeries
```

C++ Syntax

```
HRESULT CreateRunningTitleSeries(
  IRunningTitleSeries** result
);
```

Parameters

result

[out] A pointer to **IRunningTitleSeries*** pointer variable that receives the interface pointer to the created **RunningTitleSeries** object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeriesArray

DeleteAll Method of the RunningTitleSeriesArray Object

This method removes all the elements from the running title series array.

Visual Basic Syntax

```
Method DeleteAll()
```

C++ Syntax

```
HRESULT DeleteAll();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeriesArray
IRunningTitleSeriesArray::DeleteEmptySeries

DeleteEmptySeries Method of the RunningTitleSeriesArray Object

This method removes all empty running title series from the array.

Visual Basic Syntax

```
Method DeleteEmptySeries()
```

C++ Syntax

```
HRESULT DeleteEmptySeries();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeriesArray

RunningTitleSeries Object (IRunningTitleSeries Interface)

This object stores the parameters of one series of running titles. It is an element of the **RunningTitleSeriesArray** collection.

Properties

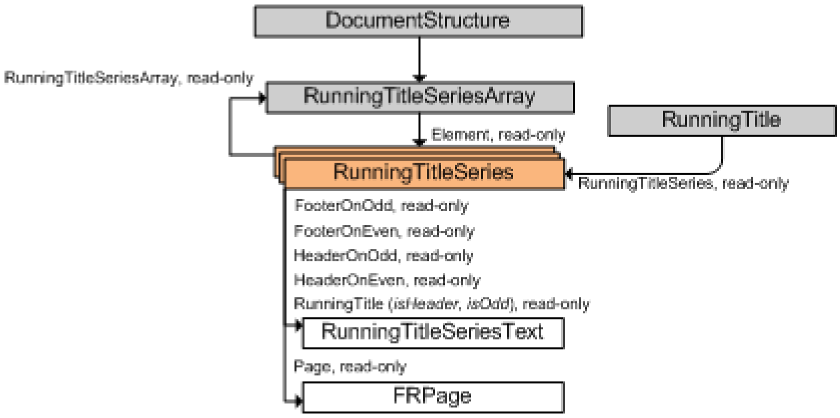
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
FooterOnEven	**RunningTitleSeriesText**, read-only	Provides access to the footer for even pages of this series.
FooterOnOdd	**RunningTitleSeriesText**, read-only	Provides access to the footer for odd pages of this series.
HeaderOnEven	**RunningTitleSeriesText**, read-only	Provides access to the header for even pages of this series.
HeaderOnOdd	**RunningTitleSeriesText**, read-only	Provides access to the header for odd pages of this series.
IsEqualOddAndEven	**Boolean**, read-only	Specifies whether this running title series is the same for odd and even pages.
Page	**FRPage**, read-only	Returns the page with the specified index from the collection of pages containing running titles of this series.
PagesCount	**Long**, read-only	Stores the number of pages with running titles of this series.
RunningTitle	**RunningTitleSeriesText**, read-only	Provides access to the text of the specified running title series.
RunningTitleSeriesArray	**RunningTitleSeriesArray**, read-only	Returns the running titles array which contains this object.

Methods

Name	Description
AddPage	Adds page for the series.

CreateFooter	Creates the same footer for odd and even pages of this series.
CreateFooterOnEven	Creates a footer for even pages of this series.
CreateFooterOnOdd	Creates a footer for odd pages of this series.
CreateHeader	Creates the same header for odd and even pages of this series.
CreateHeaderOnEven	Creates a header for even pages of this series.
CreateHeaderOnOdd	Creates a header for odd pages of this series.
DeletePage	Deletes page from the series.

Related objects



Output parameter

This object is the output parameter of the **CreateRunningTitlesSeries** method of the **RunningTitleSeriesArray** object.

See also

RunningTitleSeriesArray
Working with Properties

Page Property of the RunningTitleSeries Object

This property returns the page with the specified index from the collection of pages containing running titles of this series.

Visual Basic Syntax

```
Property Page(
   pageIndex As Long
) As FRPage
  read-only
```

C++ Syntax

```
HRESULT get_Page(
   long      pageIndex ,
   IFRPage** result
);
```

Parameters

pageIndex

[in] This variable contains the index of the page.

result

[out] A pointer to **IFRPage*** pointer variable that receives the interface pointer of the **FRPage** output object.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
FRPage
Working with Properties

RunningTitle Property of the RunningTitleSeries Object

This property provides access to the text of the specified running title series.

Visual Basic Syntax

```
Property RunningTitle(
   isHeader As Boolean,
   isOdd    As Boolean,
) As RunningTitleSeriesText
  read-only
```

C++ Syntax

```
HRESULT get_RunningTitle(
   VARIANT_BOOL              isHeader,
   VARIANT_BOOL              isOdd,
   IRunningTitleSeriesText** result
);
```

Parameters

isHeader

[in] This variable specifies if requested running title is a header.

isOdd

[in] This variable specifies if requested running title is on an odd page.

result

[out] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText
Working with Properties

AddPage Method of the RunningTitleSeries Object

This method adds a page into the internal collection of pages containing running titles of this series. After this method call the page will have the footer and header defined by this running title series.

Visual Basic Syntax

```
Method AddPage(
  page As FRPage
)
```

C++ Syntax

```
HRESULT AddPage(
   IFRPage* page
);
```

Parameters

page

[in] This variable refers to the **FRPage** object to be added.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
FRPage

CreateFooter Method of the RunningTitleSeries Object

This method creates the same footer for odd and even pages of this series. Each page of the running title series can have no more than one footer.

Visual Basic Syntax

```
Method CreateFooter() As RunningTitleSeriesText
```

C++ Syntax

```
HRESULT CreateFooter(
   IRunningTitleSeriesText** result
);
```

Parameters

result

[out, retval] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText

CreateFooterOnEven Method of the RunningTitleSeries Object

This method creates a footer for even pages of this series. Each page of the running title series can have no more than one footer.

Visual Basic Syntax

```
Method CreateFooterOnEven() As RunningTitleSeriesText
```

C++ Syntax

```
HRESULT CreateFooterOnEven(
   IRunningTitleSeriesText** result
);
```

Parameters

result

[out, retval] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText

CreateFooterOnOdd Method of the RunningTitleSeries Object

This method creates a footer for odd pages of this series. Each page of the running title series can have no more than one footer.

Visual Basic Syntax

```
Method CreateFooterOnOdd() As RunningTitleSeriesText
```

C++ Syntax

```
HRESULT CreateFooterOnOdd(
   IRunningTitleSeriesText** result
);
```

Parameters

result

[out, retval] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText

CreateHeader Method of the RunningTitleSeries Object

This method creates the same header for odd and even pages of this series. Each page of the running title series can have no more than one header.

Visual Basic Syntax

```
Method CreateHeader() As RunningTitleSeriesText
```

C++ Syntax

```
HRESULT CreateHeader(
   IRunningTitleSeriesText** result
);
```

Parameters

result

[out, retval] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText

CreateHeaderOnEven Method of the RunningTitleSeries Object

This method creates a header for even pages of this series. Each page of the running title series can have no more than one header.

Visual Basic Syntax

```
Method CreateHeaderOnEven() As RunningTitleSeriesText
```

C++ Syntax

```
HRESULT CreateHeaderOnEven(
   IRunningTitleSeriesText** result
);
```

Parameters

result

[out, retval] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText

CreateHeaderOnOdd Method of the RunningTitleSeries Object

This method creates a header for odd pages of this series. Each page of the running title series can have no more than one header.

Visual Basic Syntax

```
Method CreateHeaderOnOdd() As RunningTitleSeriesText
```

C++ Syntax

```
HRESULT CreateHeaderOnOdd(
   IRunningTitleSeriesText** result
);
```

Parameters

result

[out, retval] A pointer to **IRunningTitleSeriesText*** pointer variable that receives the interface pointer of the **RunningTitleSeriesText** output object. This object exposes properties of the running title text.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
RunningTitleSeriesText

DeletePage Method of the RunningTitleSeries Object

This method deletes page from the series.

Visual Basic Syntax

```
Method DeletePage(
  page As FRPage
)
```

C++ Syntax

```
HRESULT DeletePage(
   IFRPage* page
);
```

Parameters

page

[in] This variable refers to the **FRPage** object to be deleted.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

RunningTitleSeries
FRPage

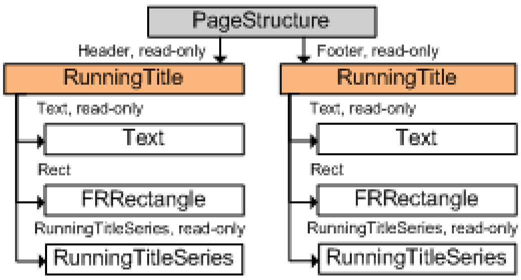
RunningTitle Object (IRunningTitle Interface)

This object provides access to a single header or footer on a page.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
IsHeader	**Boolean**, read-only	Specifies if the running title is a header or footer.
RunningTitleSeries	**RunningTitleSeries**, read-only	Provides access to the running title series to which this running title belongs.
Rect	**FRRectangle**	Specifies the rectangle of the running title in hundredth parts of point. The coordinates are counted from the left top corner of the page (see **IPageStructure::PageRect**). **Notes:** • The property returns a constant object. To change the rectangle, you must first receive an intermediate **FRRectangle** object with the help of the **IEngine::CreateRectangle** method, change the necessary parameters, and then assign this object to the property. • This rectangle may differ from the rectangle returned by the **IRunningTitleSeriesText::Rect** property as the latter contains an average position of all running title rectangles in the series, and the current rectangle defines the real position of a single header or footer.
Text	**Text**, read-only	Provides access to the text of the running title.

Related objects



See also

Working with the Logical Structure of a Document
PageStructure
Working with Properties

RunningTitleSeriesText Object (IRunningTitleSeriesText Interface)

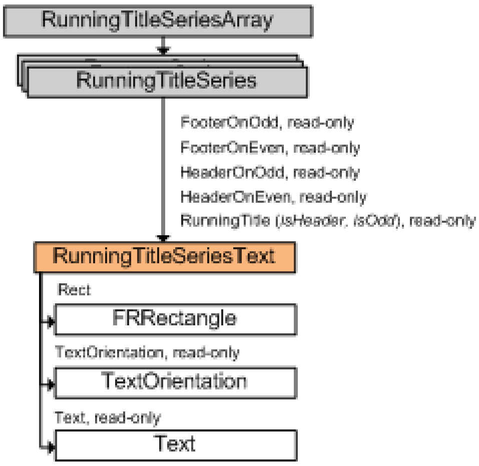
This object provides access to additional properties of the running title series concerning its text. The object defines an average position of all running title rectangles in the series, orientation of the text, and provides access to all text of the running title series, not to the text of a single running title. To work with the text of a single running title, receive the **RunningTitle** object using the **IPageStructure::Header** or **IPageStructure::Footer** property, and then use the **Text** property of the **RunningTitle** object.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
IsInverted	**Boolean**	Specifies if the running title text is inverted.
HasSeparator	**Boolean**	Specifies whether the running title text is separated from the main text with a horizontal line. A separator may be below a header or above a footer.
Rect	**FRRectangle**	Specifies an average position of all running title rectangles in the series. The coordinates are defined in hundredth parts of point and counted from the left top corner of the page. **Notes:** • To view the position of a certain header or footer, use the **Rect** property of

		the corresponding **RunningTitle** object. • The property returns a constant object. To change the rectangle, you must first receive an intermediate **FRRectangle** object with the help of the **IEngine::CreateRectangle** method, change the necessary parameters, and then assign this object to the property.
Text	**Text**, read-only	Contains all text of the running title series. This text has the TR_AbstractText role (**IText::TextRole** property).
TextOrientation	**TextOrientation**, read-only	Specifies the text orientation in the running title series. The property returns a constant object.

Related objects



Output parameter

This object is output parameter of the CreateHeader, CreateHeaderOnOdd, CreateHeaderOnEven, CreateFooter, CreateFooterOnOdd, CreateFooterOnEven methods of the RunningTitleSeries object.

See also

Working with the Logical Structure of a Document
RunningTitleSeries
Working with Properties

PageBlackSeparator Object (IPageBlackSeparator Interface)

This object represents a single page black separator in the logical structure of a document. Black separator is a line on an image. It can be horizontal, vertical, or slanting.

Black separators are found during page layout analysis, then during document synthesis their additional attributes are detected. The separators are used during export to reconstruct the initial page layout. This object works with the separator properties received after document synthesis. To work with the separators in the layout, use the **SeparatorBlock** and **SeparatorGroup** objects of the **ILayout::BlackSeparators** collection.

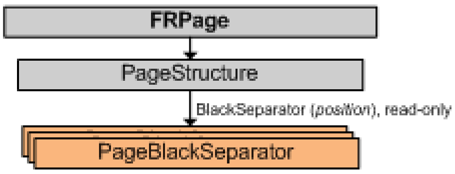
The **PageBlackSeparator** object allows you to get different black separator geometrical properties together with its type and direction. The **PageStructure** object provides access to a set of these objects. The number of separators on a page is accessible via the **IPageStructure::BlackSeparatorsCount** property, while the separator with the specified index can be accessed via the **IPageStructure::BlackSeparator** property.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
EndX	**Long**	Stores the horizontal coordinate of the end point of separator. The coordinates are measured in hundredth parts of point relative to the left top corner of the page region.

EndY	**Long**	Stores the vertical coordinate of the end point of separator. The coordinates are measured in hundredth parts of point relative to the left top corner of the page region.
Role	**PageBlackSeparatorRoleEnum**	Specifies the role of the black separator in the document. It may be a part of a table, may separate main text from some additional text, etc.
StartX	**Long**	Stores the horizontal coordinate of the start point of separator. The coordinates are measured in hundredth parts of point relative to the left top corner of the page region.
StartY	**Long**	Stores the vertical coordinate of the start point of separator. The coordinates are measured in hundredth parts of point relative to the left top corner of the page region.
Thickness	**Long**	Stores the precise width of the black separator in hundredth parts of point.
Type	**PageBlackSeparatorTypeEnum**	Stores the type of the separator.

Related objects



Output parameter

This object is the output parameter of the **AddBlackSeparator** method of the **PageStructure** object.

See also

PageStructure
Working with Properties

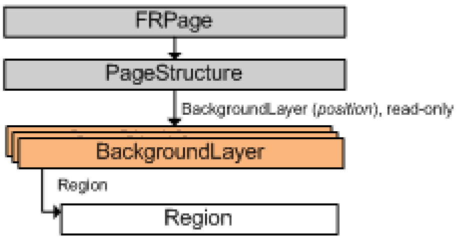
BackgroundLayer Object (IBackgroundLayer Interface)

This object exposes properties of a background layer of a page.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Color	**Long**	Specifies the background color. If the background layer is a picture (the **IsPicture** value is TRUE), the value of this property is 0xFEFFFFFF, which means that the color is transparent. By default the background color is white or RGB(255,255,255). **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color white equals 16777215.
IsPicture	**Boolean**, read-only	Specifies if the background layer is a picture. By default, the value is FALSE.
Region	**Region**	Stores the region of the background layer in pixels. The region is specified in coordinates relative to the left top corner of the image. The region is allowed to overlap the logical page rectangle (the **IPageStructure::Margins** rectangle). **Note:** The property returns a constant object. To change the background layer region, you must first receive an intermediate **Region** object with the help of the **IEngine::CreateRegion** method, change the necessary parameters, and then assign this object to the property.

Related objects



Output parameter

This object is the output parameter of the **AddBackgroundLayer** method of the **PageStructure** object

See also

PageStructure
Working with Properties

GlobalStyleStorage Object (IGlobalStyleStorafge Interface)

This object provides access to the styles of the document.

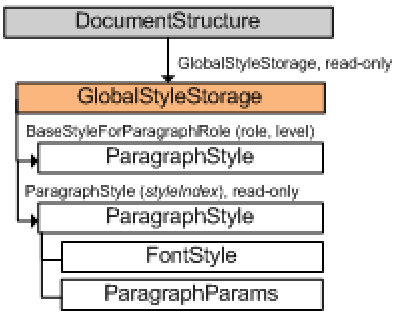
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BaseStyleForParagraphRole	**ParagraphStyle**	Provides access to the base style of paragraphs with the specified role. 📝 **Note:** The property returns a constant object. To change the base paragraph style, you must first receive an intermediate **ParagraphStyle** object with the help of the **IGlobalStyleStorage::CreateParagraphStyle** method, change the necessary parameters, and then assign this object to the property.
ParagraphStyle	**ParagraphStyle**, read-only	Provides access to the paragraph style with the specified index.
ParagraphStylesCount	**Long**, read-only	Stores the number of paragraph styles in the document.

Methods

Name	Description
Clean	Cleans the global style storage.
CreateParagraphStyle	Creates a new paragraph style (the **ParagraphStyle** object).
DeleteAllStyles	Deletes all styles in the global style storage.
DeleteAndReplaceParagraphStyle	Deletes the specified style from the global style storage and replaces this style with another style.

Related objects


DocumentStructure
GlobalStyleStorage, read-only
GlobalStyleStorage
BaseStyleForParagraphRole (role, level)
ParagraphStyle
ParagraphStyle (styleIndex), read-only
ParagraphStyle
FontStyle
ParagraphParams

See also

DocumentStructure
Working with Properties

BaseStyleForParagraphRole Property of the GlobalStyleStorage Object

This property provides access to the base style of paragraphs with the specified role.

Note: The property returns a constant object. To change the base paragraph style, you must first receive an intermediate **ParagraphStyle** object with the help of the **IGlobalStyleStorage::CreateParagraphStyle** method, change the necessary parameters, and then assign this object to the property.

Visual Basic Syntax

```
Property BaseStyleForParagraphRole(
  role As ParagraphRoleEnum,
  level As Long
) As ParagraphStyle
```

C++ Syntax

```
HRESULT get_BaseStyleForParagraphRole(
  ParagraphRoleEnum role,
  long              level,
  IParagraphStyle** result
);
HRESULT put_BaseStyleForParagraphRole(
  ParagraphRoleEnum role,
  long              level,
  IParagraphStyle*  style
);
```

Parameters

role

[in] This variable specifies the role of the paragraphs which base style is to be found. It may be set to one of the constants from the **ParagraphRoleEnum** enumeration.

level

[in] This variable specifies the level of the heading. You need to specify this parameter, only if the *role* is PR_Heading. Otherwise it is set to -1.

result

[out, retval] A variable of type **IParagraphStyle*** that receives a pointer to the interface of the **ParagraphStyle** object which contains the base style. *result* must not be NULL. **result* is guaranteed to be non-NULL after a successful call.

style

[in] This variable refers to the **ParagraphStyle** object which contains the base style to be set.

Return Values

This Property has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

GlobalStyleStorage

ParagraphStyle Property of the GlobalStyleStorage Object

This property provides access to the paragraph style with the specified index.

Visual Basic Syntax

```
Property ParagraphStyle( styleIndex As Long ) As ParagraphStyle
  read-only
```

C++ Syntax

```
HRESULT get_ParagraphStyle(
  long              styleIndex,
  IParagraphStyle** result
);
```

Parameters

styleIndex

[in] This variable specifies the index of the paragraph style in the collection of document styles. Must be in range from 0 to **IGlobalStyleStorage::ParagraphStylesCount** -1.

result

[out] A pointer to **IParagraphStyle*** pointer variable that receives the interface pointer to the returned **ParagraphStyle** object.

Return Values

This property has no specific return values. It returns standard return codes of ABBYY FineReader Engine functions.

See also

GlobalStyleStorage

Clean Method of the GlobalStyleStorage Object

This method cleans the global style storage: deletes all the styles and lists from the global style storage.

Visual Basic Syntax

```
Method Clean()
```

C++ Syntax

```
HRESULT Clean();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

GlobalStyleStorage
IGlobalStyleStorage::DeleteAllStyles

CreateParagraphStyle Method of the GlobalStyleStorage Object

This method allows you to create a new paragraph style (the **ParagraphStyle** object).

Visual Basic Syntax

```
Method CreateParagraphStyle() As ParagraphStyle
```

C++ Syntax

```
HRESULT CreateParagraphStyle(
  IParagraphStyle** result
);
```

Parameters

result

[out] A pointer to **IParagraphStyle*** pointer variable that receives the interface pointer to the created **ParagraphStyle** object. *result* must not be NULL. **result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

GlobalStyleStorage
ParagraphStyle

DeleteAllStyles Method of the GlobalStyleStorage Object

This method deletes all styles in the global style storage.

Visual Basic Syntax

```
Method DeleteAllStyles()
```

C++ Syntax

```
HRESULT DeleteAllStyles();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

GlobalStyleStorage
IGlobalStyleStorage::DeleteAndReplaceParagraphStyle
IGlobalStyleStorage::Clean

DeleteAndReplaceParagraphStyle Method of the GlobalStyleStorage Object

This method deletes the specified style from the global style storage and replaces this style in the document with another style.

Visual Basic Syntax

```
Method DeleteAndReplaceParagraphStyle(
  deleted    As ParagraphStyle,
  substitute As ParagraphStyle
)
```

C++ Syntax

```
HRESULT DeleteAndReplaceParagraphStyle(
  IParagraphStyle*  deleted,
  IParagraphStyle*  substitute
);
```

Parameters

deleted

[in] This variable refers to the **ParagraphStyle** object that should be replaced.

substitute

[in] This variable refers to the **ParagraphStyle** object that contains the new style for the paragraphs.

Remarks

If the *deleted* paragraph style is the main style for paragraphs with some role and role level and the *substitute* style has the same role and role level, the *substitute* paragraph style will be used as the main style for the paragraphs with this style and role level. Otherwise the main style for the paragraphs with this role and role level will be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

GlobalStyleStorage

ParagraphStyle Object (IParagraphStyle Interface)

This object exposes properties of the paragraph style.

⚠**Important!** If you wish to work with the style of a single paragraph, you must first call any of the functions that perform synthesis (e.g. the **Process** or **Synthesize** method of the **FRDocument** object), as these properties become meaningful only after synthesis.

Properties

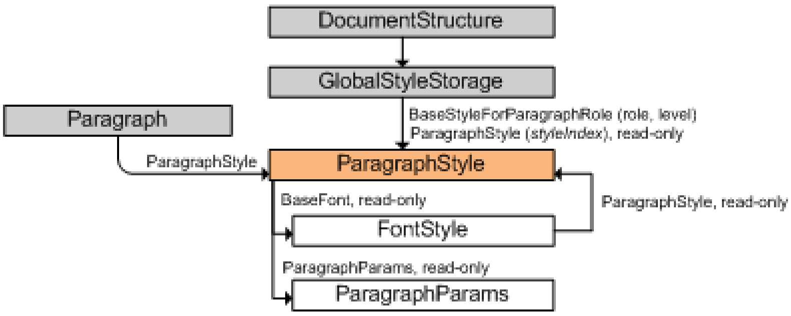
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BaseFont	**FontStyle**, read-only	Stores the base font style of this style paragraphs.
HeadingLevel	**Long**	Specifies the level of the heading. The property only makes sense, if the value of the **ParagraphRole** property is PR_Heading.

Name	**String**	Specifies the user-defined name of the paragraph style. By default, the value of this property is an empty string. You can generate the name of the style on base of the paragraph role (the **ParagraphRole** property).
ParagraphParams	**ParagraphParams**, read-only	Stores the parameters of the paragraph of this style: alignment, left and right indent, space before and after the paragraph.
ParagraphRole	**ParagraphRoleEnum**	Specifies the role of this style paragraphs in the logic structure of the document. This property can be used to generate a suitable name for the style.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.

Related objects


DocumentStructure
GlobalStyleStorage
Paragraph
BaseStyleForParagraphRole (role, level)
ParagraphStyle (styleIndex), read-only
ParagraphStyle
ParagraphStyle
BaseFont, read-only
FontStyle
ParagraphStyle, read-only
ParagraphParams, read-only
ParagraphParams

Output parameter

This object is the output parameter of the **CreateParagraphStyle** method of the **GlobalStyleStorage** object.

Input parameter

This object is the input parameter of the **DeleteAndReplaceParagraphStyle** method of the **GlobalStyleStorage** object.

See also

GlobalStyleStorage
Working with Properties

FontStyle Object (IFontStyle Interface)

This object exposes properties of a font style.

Properties

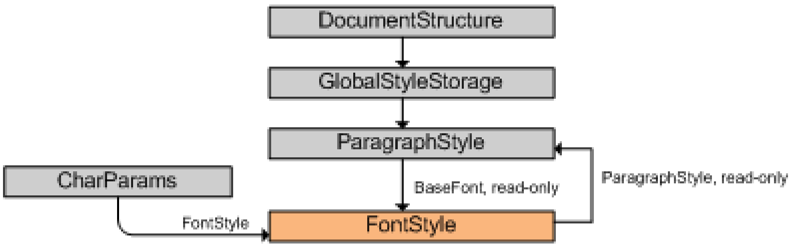
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BaseLine	**Long**	Sets the shift of a character from the base line of the string in pixels. The base line of the string is defined by the **IParagraphLine::BaseLine** property. This property is mainly used for the pictures embedded in text.
Color	**Long**	Sets the RGB value of the color for the font. By default the font color is black or RGB(0,0,0). **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color black equals 0.
IsBaseFont	**Boolean**, read-only	Specifies whether this font style is the base font style of the paragraph style.
IsBold	**Boolean**	Specifies whether the font is bold.
IsItalic	**Boolean**	Specifies whether the font is italic.
IsSmallCaps	**Boolean**	Specifies whether the font has "small caps" style. This means that the small

		characters are displayed as small capitals.
IsStrikeout	**Boolean**	Specifies whether the font is strikeout.
IsUnderlined	**Boolean**	Specifies whether the font is underlined.
FontName	**String**, read-only	Stores the name of the font. By default this value is "Times New Roman". This property cannot be changed directly but via the **SetFont** method.
FontSize	**Long**	Specifies the height of the font in twips. Twip is 1/20 of point, and point is 1/72". Default value of this property corresponds to 10 points or 200 twips.
FontType	**FontTypeEnum**, read-only	Stores the type of the font. By default this value is FT_Serif. This property cannot be changed directly but via the **SetFont** method.
HorizontalScale	**Long**	Stores horizontal scaling the font style in 1/1000. Default for this property is 1000, which corresponds to no scaling.
OverriddenStyleParams	**Long**, read-only	Describes the set of the style parameters overridden in this style against the base font style of the paragraph. The property is an OR superposition of the **StyleParamsEnum** constants.
ParagraphStyle	**ParagraphStyle**, read-only	Provides access to the paragraph style which contains this font style. The property returns a constant object.
Spacing	**Long**	Specifies additional spacing between characters in twips. Twip is 1/20 of point, and point is 1/72". Default value of this property is 0.

Methods

Name	Description
SetFont	Sets the new font name and font type.

Related objects



See also

ParagraphStyle
Working with Properties

SetFont Method of the FontStyle Object

This method allows you to set a new font name and font type for the font style. This method affects the **IFontStyle::FontName** and **IFontStyle::FontType** properties.

Visual Basic Syntax

```
Method SetFont(
  fontName As String,
  fontType As FontTypeEnum
)
```

C++ Syntax

```
HRESULT SetFont(
  BSTR         fontName,
  FontTypeEnum fontType
);
```

Parameters

fontName

[in] This variable specifies the name of the new font.

fontType

[in] This variable specifies the type of the new font. It may be set to one of the constants from the **FontTypeEnum** enumeration.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FontStyle
FontTypeEnum

List Object (IList Interface)

This object represents one list. It is a collection of list levels.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**ListLevel**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
AddLevel	Adds a new level with the next index to the collection.
RemoveAll	Deletes all levels from the collection.
Item	Provides access to a single element of the collection.

Output parameter

This object is the output parameter of the **List** property of the **ListParams** object.

See also

ListLevel
Working with Properties

AddLevel Method of the List Object

This method adds a new level with the next index to the collection. The maximum level index is 9.

Visual Basic Syntax

```
Method AddLevel() As ListLevel
```

C++ Syntax

```
HRESULT AddLevel(
  IListLevel** result
);
```

Parameters

result

[out, retval] A pointer to **IListLevel*** pointer variable that receives the interface pointer of the newly added **ListLevel** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

List
ListLevel

ListLevel Object (IListLevel Interface)

This object provides access to the parameters of one level of a list. It is an element of the **List** collection.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BulletSymbol	**Short**	Contains the bullet of the unordered list level. The property only makes sense, if the value of the **NumberingStyle** property is NS_Bullet. By default, the bullet symbol is ● (U+2022).
LevelIndex	**Long**, read-only	Stores the index of the level. The value of this property is in range from 0 to 9.
NumberingStyle	**NumberingStyleEnum**	Specifies the numbering style of the list level.
RestartNumberingOnUpperListLevelOccurance	**Long**	Specifies the upper level of the list. If this level appears in the list, the numbering in the next appearance of the current level should be restarted. By default, the value of this property is -1, which means that the numbering is not restarted.
StartNumber	**Long**	Specifies the start number of the list level. The value of this property is nonnegative. The non-numeric elements are counted from 1.
TemplateText	**String**	Specifies the template text of the list level. The text may include elements *%0*, *%1*, ..., *%n*, where *n* is the index of the current level. When viewing the document these elements will be replaced with actual values. "%%" is used for the normal character "%". The other elements are displayed as is. For example, "Section %0.%1" means that the current numbering of the level with index 0 and 1 is used in the text of the current level.

Output parameter

This object is the output parameter of the **Item**, **AddLevel** methods and **Element** property of the **List** object.

See also

List
Working with Properties

ListParams Object (IListParams Interface)

This object provides access to the parameters of the list to which a paragraph belongs. This object is a subobject of the **Paragraph** object.

Properties

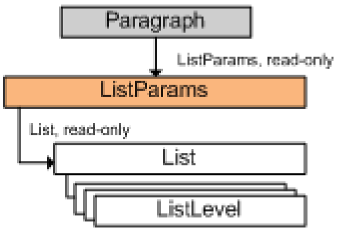
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
List	**List**, read-only	Stores the **List** object which corresponds to the list to which the paragraph belongs.
ListLevel	**Long**	Specifies the level of the paragraph in the list. It may be in range from 0 to 8. As well the value

		of the property may be -1, if the paragraph is not in the list.
OrdinalNumber	**Long**, read-only	Specifies the paragraph's number in the list.

Methods

Name	Description
AddToList	Adds the paragraph to the list.
RemoveFromList	Removes the paragraph from the list.

Related objects



See also

Paragraph
List
Working with Properties

AddToList Method of the ListParams Object

This method adds the paragraph to the list.

Visual Basic Syntax

```
Method AddToList(
   listParams As ListParams
 )
```

C++ Syntax

```
HRESULT AddToList(
   IListParams* listParams
 );
```

Parameters

listParams

[in] This parameter refers to the **ListParams** object, which contains the parameters of the list to which the paragraph should be added.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ListParams

RemoveFromList Method of the ListParams Object

This method removes the paragraph from the list.

Visual Basic Syntax

```
Method RemoveFromList()
```

C++ Syntax

```
HRESULT RemoveFromList();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ListParams

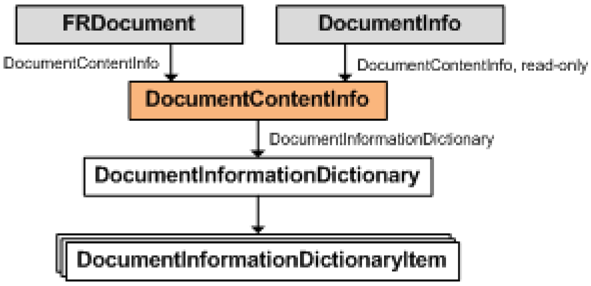
Document Supplementary Objects

Some additional information which is stored in the document, such as information about the author, keywords, subject, title of the document, can be accessed using these objects.

This section contains descriptions of the following supplementary objects:

- **DocumentContentInfo**
- **DocumentInformationDictionary**
- **DocumentInformationDictionaryItem**

The document structure objects hierarchy


FRDocument
DocumentInfo
DocumentContentInfo
DocumentContentInfo, read-only
DocumentContentInfo
DocumentInformationDictionary
DocumentInformationDictionary
DocumentInformationDictionaryItem

For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

DocumentContentInfo Object (IDocumentContentInfo Interface)

This object contains information about the author, keywords, subject, and title of the document and stores the document information dictionary. You can access content information of the certain document using the **DocumentContentInfo** property of the corresponding **FRDocument** object.

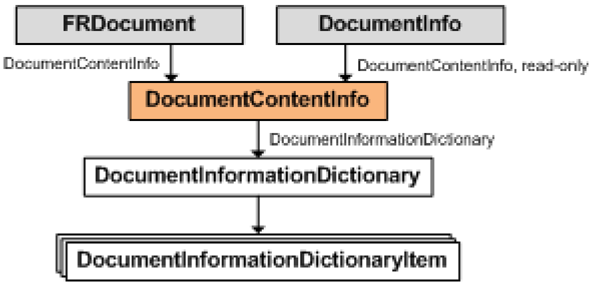
If you want the values of the properties to be written into the output file, set the corresponding properties of the needed format export parameters to TRUE (see the descriptions of the **RTFExportParams**, **HTMLExportParams**, **XLExportParams**, **PPTExportParams, PDFExportParamsOld**, **PDFAExportParamsOld**).

Properties

Name	Type	Description
Application	**Engine**, read–only	Returns the **Engine** object.
Author	**String**	Stores the name of the author of the document. You may set this property to the name of the user. The default value of this property is an empty string.
Creator	**String**	Stores the creator of the document. The default value of this property is "ABBYY FineReader Engine 10". The value of this property is used as the Creator attribute of PDF, PDF/A file.
Keywords	**String**	Stores the keywords of the document. The default value of this property is an empty string.
Producer	**String**	Stores the producer of the document. The default value of this property is an empty string. The value of this property is used as the Producer attribute of PDF, PDF/A file.
Subject	**String**	Stores the subject of the document. The default value of this property is an empty string.
Title	**String**	Stores the title of the document. The default value of this property is an empty string.

DocumentInformationDictionary	**DocumentInformationDictionary**	Stores the document information dictionary. This property is only relevant for documents in PDF and PDF/A formats. If the values of the **Author**, **Keywords**, **Subject** or **Title** properties are not empty strings, the values of these properties are used instead of the corresponding items of the **DocumentInformationDictionary** object. **Note:** The property returns a constant object. To change the document information dictionary, you must first receive an intermediate **DocumentInformationDictionary** object with the help of the **IEngine::CreateDocumentInformationDictionary** method, change the necessary parameters, and then assign this object to the property.

Related objects


FRDocument
DocumentInfo
DocumentContentInfo
DocumentContentInfo, read-only
DocumentContentInfo
DocumentInformationDictionary
DocumentInformationDictionary
DocumentInformationDictionaryItem

See also

FRDocument
DocumentInfo
Working with Properties

DocumentInformationDictionary Object (IDocumentInformationDictionary Interface)

This object represents a document information dictionary which contains metadata from the PDF file. The **DocumentInformationDictionary** may exist either as an independent object or as a sub-object of the **DocumentContentInfo** object. This object provides the standard collection functionality.

Note: If the values of the **Author**, **Keywords**, **Subject** or **Title** properties of the **DocumentContentInfo** object are not empty strings, the values of these properties are used instead of the corresponding items of the **DocumentInformationDictionary** object.

Important! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

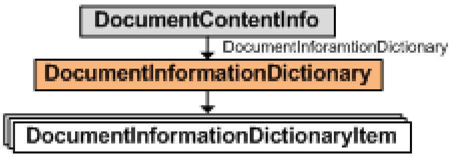
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Value	**String**	Provides access to a metadata value by its name.

Methods

Name	Description
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
CreateDocumentInformationDictionaryItem	Creates the DocumentInformationDictionaryItem object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection (the **DocumentInformationDictionaryItem** object).

Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Related objects


DocumentContentInfo
DocumentInforamtionDictionary
DocumentInformationDictionary
DocumentInformationDictionaryItem

Output parameter

This object is the output parameter of the **CreateDocumentInformationDictionary** method of the **Engine** object.

See also

DocumentInformationDictionaryItem
DocumentContentInfo
Working with Properties

Value Property of the DocumentInformationDictionary Object

This property provides access to the value of the document information dictionary element by its name.

Visual Basic Syntax

```
Property Value(
   Name As String
)As String
```

C++ Syntax

```
HRESULT get_Value(
   BSTR  Name,
   BSTR* Result
);
HRESULT put_Value(
   BSTR Name,
   BSTR Value
);
```

Parameters

Name

[in] This variable contains the name of the element in the document information dictionary.

Result

[out, retval] This variable contains the value which corresponds to the name.

Value

[in] This variable contains a new value of the element in the document information dictionary.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

DocumentInformationDictionary
Working with Properties

DocumentInformationDictionaryItem Object (IDocumentInformationDictionaryItem Interface)

This object is an element of a document information dictionary which contains metadata from the PDF file. It represents a key-value pair.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Name	**String**	Stores the name of the element which is used as key.
Value	**String**	Stores the value of the element.

Output parameter

This object is the output parameter of the following methods:

- **CreateDocumentInformationDictionaryItem** method of the **Engine** object.
- **Item** method of the **DocumentInformationDictionary** object.

Input parameter

This object is the input parameter of the **Insert**, **Add** methods of the **DocumentInformationDictionary** object.

See also

DocumentInformationDictionary
Working with Properties

Mechanism Objects

There are three objects in this group: **DocumentAnalyzer**, **Exporter** and **ScanManager**. These objects provide methods for layout analysis and recognition; for the recognized text export; and for scanning respectively. The methods for layout analysis and recognition and for recognized text export provided by the **DocumentAnalyzer** and **Exporter** objects are also called internally by similar methods of the **Engine** object, and have some additional features.

This section contains descriptions of the following objects and events and callback interfaces:

- **DocumentAnalyzer**
- **Exporter**
- **ScanManager**
- **IDocumentAnalyzerEvents**
- **IExporterEvents**
- **IScanManagerEvents**

The mechanism objects hierarchy

For more details about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**

DocumentAnalyzer Object (IDocumentAnalyzer Interface)

This object exposes a set of analysis and recognition functions analogous to those exposed by the **Engine** object. These methods are different in that the information about recognition progress is reported through special outgoing interfaces. These interfaces are **IDocumentAnalyzerEvents** (for C++) and a dispinterface **DIDocumentAnalyzerEvents** (for Visual Basic).

It is worth noting that Visual Basic users should not care for details of event interfaces implementation as this development platform provides easy means for handling them. This object may be declared *WithEvents* in Visual Basic.

For C++ user this fact means that it supports the **IConnectionPointContainer** interface. To receive notification events during recognition, a C++ user should create an object derived from the **IDocumentAnalyzerEvents** interface, then set up the connection between it and events source implemented in **DocumentAnalyzer** object by standard COM means.

The document analyzer allows you to use its cache dictionary during processing. The cache dictionary is a small dictionary (about a hundred words) which can be changed easily during processing. The cache dictionaries can be used when it is possible to select a dictionary more precisely, if you found new information about the document during processing.

Note: It is not recommended to recognize more than one page with the use of a single instance of the **DocumentAnalyzer** object, as it may lead to unpredictable effects. The **DocumentAnalyzer** performs a kind of self-teaching during analysis and recognition, and thus tunes itself for recognition of text of a certain type. That is why it is good to use the **DocumentAnalyzer** object instance for recognition of a number of blocks on a single page, as this improves speed and quality of recognition.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.

Methods

Name	Description
AddWordsToCacheDictionary	Adds a group of words to the cache dictionary.
AddWordToCacheDictionary	Adds one word to the cache dictionary.
AnalyzeAndRecognizePage	Performs layout analysis, recognition, and page synthesis of the specified image.
AnalyzeAndRecognizePages	Performs layout analysis, recognition, and page synthesis of an images collection.
AnalyzePage	Performs layout analysis of an image.
AnalyzePages	Performs layout analysis of an images collection.
AnalyzeRegion	Analyzes layout of the image inside the specified region.
AnalyzeTable	Replaces the specified block with the table block and analyzes the structure of table.
CleanCacheDictionary	Deletes all words from the cache dictionary.
DetectOrientation	Detects text orientation on the image.
ExtractBarcodes	Finds and recognizes all barcode blocks. **Note:** This method is obsolete and is intended to be removed in the next version of ABBYY FineReader Engine.
FindPageSplitPosition	Detects the direction of text on image and finds the position of splitting it on pages.
RecognizeBlocks	Recognizes text in an explicitly specified set of blocks and performs page synthesis.
RecognizeImageDocumentAsPlainText	Recognizes an image and returns recognized text in a special "plain text" format.
RecognizePage	Recognizes parts of the specified image that lay inside the blocks in the specified layout and performs page synthesis.
RecognizePages	Recognizes those parts of the images from the collection that lay inside the blocks of the specified layout collection and performs page synthesis.
RemoveGeometricalDistortions	Straightens out distorted lines on an image. Distorted lines may occur close to the binding when scanning/photographing thick books.

Output parameter

This object is the output parameter of the **CreateDocumentAnalyzer** method of the **Engine** object.

See also

IDocumentAnalyzerEvents
Working with Connectable Objects

AddWordsToCacheDictionary Method of the DocumentAnalyzer Object

This method adds a group of words to the cache dictionary. The cache dictionary is a small dictionary (about a hundred words) which can be changed easily during processing. The cache dictionaries can be used when it is possible to select a dictionary more precisely, if you found new information about the document during processing.

Visual Basic Syntax

```
Method AddWordsToCacheDictionary(
  params  As RecognizerParams,
  words   As StringsCollection,
  weights As LongsCollection
)
```

C++ Syntax

```
HRESULT AddWordsToCacheDictionary(
  IRecognizerParams*  params,
  IStringsCollection* words,
  ILongsCollection*   weights
);
```

Parameters

params

[in] The **RecognizerParams** object that stores parameters of page processing.

words

[in] This parameter of the **StringsCollection** type contains the collection of the newly added words.

weights

[in] This parameter of the **LongsCollection** type that must have the same size as the collection of words, is used to pass information about the weights for the newly added words. The weights for the words must be in the range from 1 to 200. You may pass 0 for this parameter in which case all the words will be included in the dictionary with default weights of 100. The weight assigned to the word in the dictionary may have a set of discrete values only. These values are 25, 50, 100, 200. The value passed in this parameter is rounded to the nearest of the discrete set of values.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

For more efficient operation it is recommended to pre-sort the added words in alphabetical order.

For adding one word into the cache dictionary, you can use the **IDocumentAnalyzer::AddWordToCacheDictionary** method.

See also

DocumentAnalyzer
IDocumentAnalyzer::AddWordToCacheDictionary
Working with Dictionaries

AddWordToCacheDictionary Method of the DocumentAnalyzer Object

This method adds one word to the cache dictionary. The cache dictionary is a small dictionary (about a hundred words) which can be changed easily during processing. The cache dictionaries can be used when it is possible to select a dictionary more precisely, if you found new information about the document during processing.

Visual Basic Syntax

```
Method AddWordToCacheDictionary(
  params As RecognizerParams,
  word   As String,
  weight As Long
)
```

C++ Syntax

```
HRESULT AddWordToCacheDictionary(
  IRecognizerParams* params,
  BSTR               word,
  long               weight
);
```

Parameters

params

[in] The **RecognizerParams** object that stores parameters of page processing.

word

[in] This parameter contains the newly added word.

weight

[in] The weight assigned to the word in the dictionary. Must be in the range from 1 to 200. The higher the weight for a word is, the more likely this word will be taken as a variant during recognition. The normal value for this parameter is 100. Visual Basic users see this parameter as having default value of 100. The weight assigned to the word in the dictionary may have a set of discrete values only. These values are 25, 50, 100, 200. The value passed in this parameter is rounded to the nearest of the discrete set of values.

Remarks

For adding a group of words into the cache dictionary, use the **IDocumentAnalyzer::AddWordsToCacheDictionary** method instead.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

DocumentAnalyzer
IDocumentAnalyzer::AddWordsToCacheDictionary
Working with Dictionaries

AnalyzeAndRecognizePage Method of the DocumentAnalyzer Object

This method performs layout analysis, recognition, and page synthesis of the image specified.

Visual Basic Syntax

```
Method AnalyzeAndRecognizePage(
  imageDocument    As ImageDocument,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage,
  layout           As Layout,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognizePage(
  IImageDocument*          imageDocument,
  IPageProcessingParams*   processingParams,
  ISynthesisParamsForPage* synthesisParams,
  ILayout*                 layout,
  IDocumentInfo*           documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be processed.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores the parameters of the layout analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters have their properties set to default values, and the recognition language is English) or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. After this method is done, it contains the results of the layout analysis and recognition.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same DocumentInfo object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets the size and resolution of the layout to be equal to those of the deskewed black-and-white page of **ImageDocument**.
- All existing blocks are deleted from *layout*.
- Calling this method is not equivalent to successive calls to **IDocumentAnalyzer::AnalyzePage** and **IDocumentAnalyzer::RecognizePage** methods, as the recognition information is used for more accurate layout analysis.
- The method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **DocumentAnalyzer**.

See also

DocumentAnalyzer
IEngine::AnalyzeAndRecognizePage
IFRPage::AnalyzeAndRecognize
Working with Profiles

AnalyzeAndRecognizePages Method of the DocumentAnalyzer Object

This method performs layout analysis, recognition, and page synthesis of an images collection.

Visual Basic Syntax

```
Method AnalyzeAndRecognizePages(
  imageDocuments   As ImageDocumentsCollection,
  layouts          As LayoutsCollection,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzeAndRecognizePages(
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  IPageProcessingParams*     processingParams,
  ISynthesisParamsForPage*   synthesisParams,
  IDocumentInfo*             documentInfo
);
```

Parameters

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be processed. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layouts

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. After this method is done, it contains the results of layout analysis and recognition.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores the parameters of the layout analysis and recognition. This parameter may be 0. In this case the page is analyzed and recognized using the default parameters (all page processing parameters have their properties set to default values, and the recognition language is English), or, if a profile has been loaded, the parameters set by this profile are used.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same DocumentInfo object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

- The method may report events to the listeners attached to the IConnectionPointContainer interface of the **DocumentAnalyzer**.

See also

DocumentAnalyzer
IEngine::AnalyzeAndRecognizePages
IFRDocument::AnalyzeAndRecognizePages
Working with Profiles

AnalyzePage Method of the DocumentAnalyzer Object

This method performs layout analysis of an image.

The method may report events to the listeners attached to the **IConnectionPointContainer** interface of the **DocumentAnalyzer**.

Visual Basic Syntax

```
Method AnalyzePage(
  imageDocument    As ImageDocument,
  processingParams As PageProcessingParams,
  layout           As Layout
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzePage(
  IImageDocument*        imageDocument,
  IPageProcessingParams* processingParams,
  ILayout*               layout,
  IDocumentInfo*         documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be analyzed.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. After analysis it contains the results of layout analysis.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets these parameters for layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- All existing blocks are deleted from *layout.*

See also

DocumentAnalyzer
IEngine::AnalyzePage
IFRPage::Analyze
Working with Profiles

AnalyzePages Method of the DocumentAnalyzer Object

This method performs layout analysis of an images collection.

Visual Basic Syntax

```
Method AnalyzePages(
  imageDocuments   As ImageDocumentsCollection,
  layouts          As LayoutsCollection,
  processingParams As PageProcessingParams,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzePages(
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  IPageProcessingParams*     processingParams,
  IDocumentInfo*             documentInfo
);
```

Parameters

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be analyzed. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layouts

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. After this method is done, it contains the results of layout analysis.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the page is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same DocumentInfo object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is

optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

- The method may report events to the listeners attached to the IConnectionPointContainer interface of the **DocumentAnalyzer**.

See also

DocumentAnalyzer
IEngine::AnalyzePages
IFRDocument::AnalyzePages
Working with Profiles

AnalyzeRegion Method of the DocumentAnalyzer Object

This function analyzes the layout of the image inside the specified region.

It does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **DocumentAnalyzer**.

Visual Basic Syntax

```
Method AnalyzeRegion(
  imageDocument    As ImageDocument,
  region           As Region,
  processingParams As PageProcessingParams,
  layout           As Layout,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzeRegion(
  IImageDocument*        imageDocument,
  IRegion*               region,
  IPageProcessingParams* processingParams,
  ILayout*               layout,
  IDocumentInfo*         documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object for which the *layout* and *region* are defined.

region

[in] This variable refers to the **Region** object that specifies the area on image that is to be analyzed. It should be set in coordinates of the deskewed black-and-white plane of the **ImageDocument**.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores parameters of layout analysis. This parameter may be 0. In this case the region is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. After analysis it contains the results of layout analysis.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets these parameters for layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.

- During the layout analysis in region all the blocks that lay entirely inside the region are deleted from the **Layout** specified. Zero or more new blocks may be added to the **Layout** as the result of this method call.

See also

DocumentAnalyzer
IFRPage::AnalyzeRegion
Working with Profiles

AnalyzeTable Method of the DocumentAnalyzer Object

This method replaces the specified block with the table block and analyzes the structure of the table.

It does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **DocumentAnalyzer**.

Visual Basic Syntax

```
Method AnalyzeTable(
  imageDocument    As ImageDocument,
  processingParams As PageProcessingParams,
  layout           As Layout,
  blockIndex       As Long,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT AnalyzeTable(
  IImageDocument*        imageDocument,
  IPageProcessingParams* processingParams,
  ILayout*               layout,
  long                   blockIndex,
  IDocumentInfo*         documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image on which the layout is defined.

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores parameters of table layout analysis. This parameter may be 0. In this case the table is analyzed with default parameters (all page processing parameters are set to default values), or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. It should contain the block with the index specified by the *blockIndex* variable. It is this block that will be analyzed as table.

blockIndex

[in] This variable specifies the index of block in the collection of blocks that belongs to *layout*.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If layout analysis is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets these parameters for layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- Table blocks always have rectangular regions; if the block was not rectangular, the new table block receives the region corresponding to bounding rectangle of the initial block.
- If the table structure cannot be analyzed, the *layout* is not changed.

See also

DocumentAnalyzer
IFRPage::AnalyzeTable
Working with Profiles

CleanCacheDictionary Method of the DocumentAnalyzer Object

This method deletes all words from the cache dictionary. The cache dictionary is a small dictionary (about a hundred words) which can be changed easily during processing. The cache dictionaries can be used when it is possible to select a dictionary more precisely, if you found new information about the document during processing.

Visual Basic Syntax

```
Method CleanCacheDictionary(
  params As RecognizerParams
)
```

C++ Syntax

```
HRESULT CleanCacheDictionary(
  IRecognizerParams* params
);
```

Parameters

params

[in] The **RecognizerParams** object that stores parameters of page processing.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

DocumentAnalyzer
IDocumentAnalyzer::AddWordToCacheDictionary
IDocumentAnalyzer::AddWordsToCacheDictionary
Working with Dictionaries

DetectOrientation Method of the DocumentAnalyzer Object

This method detects text orientation on the image. The method returns **TextOrientation** object, if orientation has been detected successfully, and NULL, if the program failed to detect orientation.

It does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **DocumentAnalyzer**.

Visual Basic Syntax

```
Method DetectOrientation(
  imageDocument     As ImageDocument,
  orientationParams As OrientationDetectionParams
  extractionParams  As ObjectsExtractionParams,
  recognizerParams  As RecognizerParams
) As TextOrientation
```

C++ Syntax

```
HRESULT DetectOrientation(
  IImageDocument*              imageDocument,
  IOrientationDetectionParams* orientationParams,
  IObjectsExtractionParams*    extractionParams,
  IRecognizerParams*           recognizerParams,
  ITextOrientation**           result
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image, on which text orientation is to be detected.

orientationParams

[in] This variable refers to the **OrientationDetectionParams** object that stores parameters of orientation detection. This parameter may be 0. In this case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

recognizerParams

[in] This variable refers to the **RecognizerParams** object that stores parameters of page recognition. This parameter may be 0. In this case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

result

[out, retval] A pointer to **ITextOrientation*** pointer variable that receives the interface pointer of the **TextOrientation** output object. This object provides access to the text orientation on the page. If orientation detection failed, NULL is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Calling this method is equivalent to the call to **IDocumentAnalyzer::AnalyzePage** method with the following parameters of the input **PageProcessingParams** object: DetectOrientation = true, PerformPageAnalysis = false, RemoveGeometricalDictortions = false, DetectBarcodes = false, DetectInvertedImage = false.

See also

DocumentAnalyzer
IFRPage::DetectOrientation
IPageProcessingParams::DetectOrientation
Working with Profiles

ExtractBarcodes Method of the DocumentAnalyzer Object

This method finds and recognizes all barcode blocks on an image, no other blocks are processed.

The method does not report any events to the listeners attached to the **IConnectionPointContainer** interface of the **DocumentAnalyzer**.

Visual Basic Syntax

```
Method ExtractBarcodes(
  imageDocument    As ImageDocument,
  barcodeParams    As BarcodeParams,
  extractionParams As ObjectsExtractionParams,
  layout           As Layout,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT ExtractBarcodes(
  IImageDocument*           imageDocument,
  IBarcodeParams*           barcodeParams,
  IObjectsExtractionParams* extractionParams,
  ILayout*                  layout,
  IDocumentInfo*            documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be processed.

barcodeParams

[in] This variable refers to the **BarcodeParams** object that stores parameters of barcode processing. This parameter may be 0. In this case all barcode processing parameters are set to default values, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the parameters are set to default values, or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. After analysis it contains the results of layout analysis.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets these parameters for layout to be equal to those of the deskewed black-and-white page of the **ImageDocument**.
- All existing blocks are deleted from *layout*.
- Calling this method is equivalent to the call to **IDocumentAnalyzer::AnalyzeAndRecognizePage** method with the following parameters of the input **PageProcessingParams** object: DetectBarcodes = true, PerformPageAnalysis = false, RemoveGeometricalDictortions = false, DetectOrientation = false, DetectInvertedImage = false.
- This method is obsolete and is intended to be removed in the next version of ABBYY FineReader Engine.

See also

DocumentAnalyzer
IFRPage::ExtractBarcodes
IPageProcessingParams::DetectBarcodes
Working with Profiles

FindPageSplitPosition Method of the DocumentAnalyzer Object

This method detects the direction of text on image and finds the position of splitting it on pages, if it exists. It is used to detect the ability to split dual pages in a book.

The split position is defined by two lines, which coordinates are returned in the *startSplitPosition* and *endSplitPosition* parameters. The image area between these two lines should be removed when splitting image on pages. This area usually contains some garbage.

Visual Basic Syntax

```
Method FindPageSplitPosition(
  imageDocument      As ImageDocument,
  extractionParams   As ObjectsExtractionParams,
  splitDirection     As PageSplitDirectionEnum,
  startSplitPosition As Long,
  endSplitPosition   As Long
)
```

C++ Syntax

```
HRESULT FindPageSplitPosition(
  IImageDocument*           imageDocument,
  IObjectsExtractionParams* extractionParams,
  PageSplitDirectionEnum*   splitDirection,
  long*                     startSplitPosition,
  long*                     endSplitPosition
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be split on pages.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the default values are used, or, if a profile has been loaded, the parameters set by this profile are used.

splitDirection

[out] This variable receives the type of possible split: vertical split, horizontal split, or no split. Refer to the **PageSplitDirectionEnum** description for details.

startSplitPosition

[out] The coordinate of the first line, which defines split position (if a split is possible). The meaning of this value depends on the value of the *splitDirection* variable. If the possibility of vertical split is detected, it contains the horizontal coordinate of the split line. If the possibility of horizontal split is detected, it contains the vertical coordinate of the split line. Coordinate is given against the deskewed black-and-white page of the image.

endSplitPosition

[out] The coordinate of the second line, which defines split position (if a split is possible). The meaning of this value depends on the value of the *splitDirection* variable. If the possibility of vertical split is detected, it contains the horizontal coordinate of the split line. If the possibility of horizontal split is detected, it contains the vertical coordinate of the split line. Coordinate is given against the deskewed black-and-white page of the image.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Try to find dual pages among the images and split them into pages
 BOOL ProcessImages( FREngine::IStringsCollection* images )
 {
   BOOL bSplitWasFound = FALSE;
   int i;
   for( i = 0; i < images->Count; i++ ) {
     _bstr_t image = images->Item( i );
     // Let the images are in ABBYY FineReader Engine internal format
     // we may open them directly
     FREngine::IImageDocumentPtr imageDoc = Engine->OpenImage( image );
     FREngine::IDocumentAnalyzerPtr documentAnalyzer = Engine-
>CreateDocumentAnalyzer();
     FREngine::PageSplitDirectionEnum splitDirection;
     long position1;
     long position2;
     // Try to find the dual page split
     documentAnalyzer->FindPageSplitPosition( imageDoc, 0, &splitDirection, &position1,
&position2 );
     switch( splitDirection ) {
       case FREngine::PSD_NoSplit:
            continue; // No split is possible
       case FREngine::PSD_HorizontalSplit:
            bSplitWasFound = TRUE;
            // make the horizontal split
            DoHorizontalSplit( imageDoc, position1, position2 );
            break;
       case FREngine::PSD_VerticalSplit:
            bSplitWasFound = TRUE;
            // make the vertical split
            DoVerticalSplit( imageDoc, position1, position2 );
            break;
     }
   }
   return bSplitWasFound;
 }
```

Visual Basic code

```
' Try to find dual pages among the images and split them into parts
 Function ProcessImages(Images As FREngine.StringsCollection) As Boolean
   Dim SplitFound As Boolean
   SplitFound = False
   Dim i As Long
   For i = 0 To Images.Count - 1
   ' Let the images are in ABBYY FineReader Engine internal format
   ' we may open them directly
     Dim Image As String
     Image = Images.Item(i)
     Dim ImageDoc As FREngine.ImageDocument
     Set ImageDoc = Engine.OpenImage(Image)
     Dim documentAnalyzer As FREngine.DocumentAnalyzer
     Set documentAnalyzer = Engine.CreateDocumentAnalyzer
     Dim SplitDirection As FREngine.PageSplitDirectionEnum
     Dim Position1 As Long
     Position1 = 0
     Dim Position2 As Long
     Position2 = 0
     ' Try to find the dual page split
     documentAnalyzer.FindPageSplitPosition ImageDoc, Nothing, SplitDirection,
Position1, Position2
     Select Case SplitDirection
       Case PSD_HorizontalSplit
            SplitFound = True
            ' make the horizontal split
            DoHorizontalSplit ImageDoc, Position1, Position2
       Case PSD_VerticalSplit
            SplitFound = True
            ' make the vertical split
            DoVerticalSplit ImageDoc, Position1, Position2
     End Select
   Next
   ProcessImages = SplitFound
 End Function
```

See also

DocumentAnalyzer
IFRPage::FindPageSplitPosition
Working with Profiles

RecognizeBlocks Method of the DocumentAnalyzer Object

This method recognizes text in an explicitly specified set of blocks and performs page synthesis.

Visual Basic Syntax

```
Method RecognizeBlocks(
  imageDocument    As ImageDocument,
  synthesisParams  As SynthesisParamsForPage,
  extractionParams As ObjectsExtractionParams,
  layout           As Layout,
  blocks           As LayoutBlocks,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT RecognizeBlocks(
  IImageDocument*           imageDocument,
  ISynthesisParamsForPage*  synthesisParams,
  IObjectsExtractionParams* extractionParams,
  ILayout*                  layout,
  ILayoutBlocks*            blocks,
  IDocumentInfo*            documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be processed.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. The blocks in the layout should be created before calling the method. After recognition these blocks will contain the recognized text.

blocks

[in] This variable refers to the **LayoutBlocks** object, specifies the set of blocks to be recognized. All these blocks should belong to the *layout*, otherwise an error code is returned.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same DocumentInfo object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets the size and resolution of the layout to be equal to those of the deskewed black-and-white page of **ImageDocument**.

- This method is analogous to the **IDocumentAnalyzer::RecognizePage**, except that only the specified blocks from layout are recognized.

See also

DocumentAnalyzer
IFRPage::RecognizeBlocks
Working with Profiles

RecognizeImageDocumentAsPlainText Method of the DocumentAnalyzer Object

This method recognizes an image and returns recognized text in a special "plain text" format. This format only contains information about recognized text symbols, recognition confidence and positions of these symbols as relative to the recognized image.

Visual Basic Syntax

```
Method RecognizeImageDocumentAsPlainText(
  image            As ImageDocument,
  processingParams As PageProcessingParams,
  synthesisParams  As SynthesisParamsForPage,
  documentInfo     As DocumentInfo
) As PlainText
```

C++ Syntax

```
HRESULT RecognizeImageDocumentAsPlainText(
  IImageDocument*          image,
  IPageProcessingParams*   processingParams,
  ISynthesisParamsForPage* synthesisParams,
  IDocumentInfo*           documentInfo,
  IPlainText**             Result
);
```

Parameters

image

[in] This variable refers to the **ImageDocument** object corresponding to the image to be recognized

processingParams

[in] This variable refers to the **PageProcessingParams** object that stores the parameters of analysis and recognition. This parameter is optional and may be 0. In this case the page is analyzed and recognized using the default parameters — all page processing parameters are set to default values, and the recognition language is English.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter is optional and may be 0. In this case the page is synthesized with default parameters.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same **DocumentInfo** object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Result

[out, retval] A pointer to **IPlainText*** pointer variable that receives the interface pointer of the **PlainText** output object. This object provides information about recognized symbols and positions of these symbols relative to the recognized image.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

See also

DocumentAnalyzer
IEngine::RecognizeImageAsPlainText
IEngine::RecognizeImageDocumentAsPlainText
PlainText

RecognizePage Method of the DocumentAnalyzer Object

This method recognizes parts of the specified image that lay inside the blocks of the specified **Layout** object and performs page synthesis.

Visual Basic Syntax

```
Method RecognizePage(
  imageDocument    As ImageDocument,
  synthesisParams  As SynthesisParamsForPage,
  extractionParams As ObjectsExtractionParams,
  layout           As Layout,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT RecognizePage(
  IImageDocument*           imageDocument,
  ISynthesisParamsForPage*  synthesisParams,
  IObjectsExtractionParams* extractionParams,
  ILayout*                  layout,
  IDocumentInfo*            documentInfo
);
```

Parameters

imageDocument

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be recognized.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

layout

[in] This variable refers to the **Layout** object corresponding to the page layout. The blocks in the layout should be created before calling the method. After recognition these blocks will contain the recognized text.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same DocumentInfo object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- If the sizes and resolutions of the image and layout do not match, this method sets the size and resolution of the layout to be equal to those of the deskewed black-and-white page of **ImageDocument**.

- Call this method after you have analyzed or created the layout of the page manually. The old text from blocks, if there is any, is deleted. If the layout contains any table blocks with non-analyzed structure, they will be recognized as containing a single cell corresponding to the whole table. Only text, table and barcode blocks are recognized.

See also

DocumentAnalyzer
IEngine::RecognizePage

IFRPage::Recognize
Working with Profiles

RecognizePages Method of the DocumentAnalyzer Object

This method recognizes those parts of the images from the collection that lay inside the blocks of the specified layout collection and performs page synthesis.

Visual Basic Syntax

```
Method RecognizePages(
  imageDocuments   As ImageDocumentsCollection,
  layouts          As LayoutsCollection,
  synthesisParams  As SynthesisParamsForPage,
  extractionParams As ObjectsExtractionParams,
  documentInfo     As DocumentInfo
)
```

C++ Syntax

```
HRESULT RecognizePages(
  IImageDocumentsCollection* imageDocuments,
  ILayoutsCollection*        layouts,
  ISynthesisParamsForPage*   synthesisParams,
  IObjectsExtractionParams*  extractionParams,
  IDocumentInfo*             documentInfo
);
```

Parameters

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object corresponding to the images collection that is to be recognized. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the layouts.

layouts

[in] This variable refers to the **LayoutsCollection** object corresponding to the collection of the page layouts. The blocks in the layouts should be created before calling the method. After recognition these blocks will contain the recognized text.

synthesisParams

[in] This variable refers to the **SynthesisParamsForPage** object that stores parameters of page synthesis. This parameter may be 0. In this case the page is synthesized with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

extractionParams

[in] This variable refers to the **ObjectsExtractionParams** object that stores parameters of objects extraction. This parameter may be 0. In this case the objects are extracted with default parameters, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object that stores service information about the open PDF file. You should use the same DocumentInfo object, which was used as a parameter during image preparation (e.g. in **IEngine::PrepareImage** method). In this case, all information about the image which was received during preparation is used during analysis and recognition. This parameter is optional and may be set to 0, which means either that this information need not be used or that a file other than PDF is being processed.

Return Values

If recognition is interrupted by the user, this method will return E_ABORT. If pattern training is interrupted by the user, this method will return FREN_E_PATTERN_TRAINING_ABORTED. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

- Call this method after you have analyzed or created the layouts of the pages manually. The old text from blocks, if there is any, is deleted. If the layouts contain any table blocks with non-analyzed structure, they will be recognized as containing a single cell corresponding to the whole table. Only text, table and barcode blocks are recognized.

- Depending on the value of the **IEngine::MultiProcessingParams** property, ABBYY FineReader Engine can distribute analysis and recognition of multi-page documents to CPU cores.

- This method may report events to the listeners attached to the IConnectionPointContainer interface of the **DocumentAnalyzer**.

See also

DocumentAnalyzer
IEngine::RecognizePages
IFRDocument::RecognizePages
Working with Profiles

RemoveGeometricalDistortions Method of the DocumentAnalyzer Object

This method straightens out distorted lines on an image. Distorted lines may occur close to the binding when scanning/photographing thick books.

We recommend to call this method after the page orientation has been corrected and a double-page spread has been split into two separate pages, if necessary. This method should be called after layout analysis, for example after the **IDocumentAnalyzer::AnalyzePage** method. We recommend to set the correct recognition language before analysis, especially for texts in Chinese, Japanese and Korean.

This method may report events to the listeners attached to the **IConnectionPointContainer** interface of **DocumentAnalyzer**.

Visual Basic Syntax

```
Method RemoveGeometricalDistortions(
  image  As ImageDocument,
  params As ObjectsExtractionParams
)
```

C++ Syntax

```
HRESULT RemoveGeometricalDistortions(
  IImageDocument*           image,
  IObjectsExtractionParams* params
);
```

Parameters

image

[in] This variable refers to the **ImageDocument** object corresponding to the image that is to be preprocessed.

params

[in] This variable refers to the **ObjectsExtractionParams** object corresponding to the parameters used for straightening out distorted lines on an image. This parameter may be 0, in which case the default parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

Return Values

If straightening is interrupted by the user, this method will return E_ABORT. It also returns the standard return codes of the ABBYY FineReader Engine functions.

Remarks

Calling this method is equivalent to the call to **IDocumentAnalyzer::AnalyzePage** method with the following parameters of the input **PageProcessingParams** object: RemoveGeometricalDictortions = true, PerformPageAnalysis = false, DetectOrientation = false, DetectBarcodes = false, DetectInvertedImage = false.

See also

DocumentAnalyzer
IFRPage::RemoveGeometricalDistortions
IPageProcessingParams::RemoveGeometricalDistortions
Working with Profiles

Exporter Object (IExporter Interface)

This object provides tools for saving recognized text into files in external formats. Its methods provide more advanced features than similar methods of the **Engine** object. The latter use the functionality of the **Exporter** object internally, simplifying the procedure of saving recognized text in a file at the same time. The features that the **Exporter** object provides, compared to the **Engine** object, are:

- You may get a list of additional files that were generated during export (e.g., pictures for HTML format).
- Information about export progress is reported through special outgoing interfaces. These interfaces are **IExporterEvents** (for C++), and a dispinterface **DIExporterEvents** (for Visual Basic). But it's worth noting that Visual Basic users should not care for details of event interfaces implementation as this development platform provides easy means for handling them.

This object may be declared *WithEvents* in Visual Basic. For C++ user this fact means that it supports the **IConnectionPointContainer** interface. To receive notification events during recognition, a C++ user should create an object derived from the **IExporterEvents** interface, then set up the connection between it and events source implemented in **Exporter** object by standard COM means.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.

Methods

Name	Description
ExportPages	Saves recognized text from several pages into a file in external format.
ExportPagesEx	Saves recognized text from several pages into a file in external format. This method is optimized from the point of view of memory consumption.

Output parameter

This object is the output parameter of the **CreateExporter** method of the **Engine** object.

See also

IExporterEvents
IEngine::ExportPages
IFRDocument::ExportPages
Working with Connectable Objects

ExportPages Method of the Exporter Object

This method saves recognized text from several pages into a file in an external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants.

This method reports events to the listeners attached to the **IConnectionPointContainer** interface supported by the **Exporter** object.

Visual Basic Syntax

```
Method ExportPages(
  format                 As FileExportFormatEnum,
  fileName               As String,
  imageDocuments         As ImageDocumentsCollection,
  layouts                As LayoutsCollection,
  exportParams           As Unknown,
  documentInfo           As DocumentInfo,
  additionalFiles        As StringsCollection,
  additionalDirectories  As StringsCollection
)
```

C++ Syntax

```
HRESULT ExportPages(
  FileExportFormatEnum        format,
  BSTR                        fileName,
  IImageDocumentsCollection*  imageDocuments,
  ILayoutsCollection*         layouts,
  IUnknown*                   exportParams,
  IDocumentInfo*              documentInfo,
  IStringsCollection**        additionalFiles,
  IStringsCollection**       additionalDirectories
);
```

Parameters

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description for the supported file formats.

fileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

imageDocuments

[in] This variable refers to the **ImageDocumentsCollection** object that corresponds to the set of images that belong to the exported pages. The number of images in the collection must correspond to the number of **Layout** objects in the collection of the exported layouts. This parameter must not be 0.

layouts

[in] This variable refers to the **LayoutsCollection** object containing the set of layouts that belong to the exported pages. This parameter may be 0 when exporting pages to PDF (PDF/A) format using PEM_ImageOnly mode.

exportParams

[in] Pass the export parameters object of the type corresponding to your file format through this input parameter. For example, if you are creating an RTF file, create the **RTFExportParams** object, set the necessary parameters in it, and pass it to this method as the *exportParams* input parameter. This parameter may be 0, in which case the default values for the export parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object. You should use the same **DocumentInfo** object, which was used as a parameter in the **SynthesizePages** or **SynthesizePagesEx** methods of the **Engine** object. In this case, all the information about document which was received during synthesis is used during export. This parameter may be 0, in which case the text attributes which were detected during synthesis are not available.

additionalFiles

[out] A pointer to the **IStringsCollection*** pointer variable that receives the interface pointer of the **StringsCollection** object. **additionalFiles* should not refer to any valid object. The **StringsCollection** is created internally by this method. This object contains the list of full paths to the additional files that were generated during export. Must not be NULL.

additionalDirectories

[out] A pointer to the **IStringsCollection*** pointer variable that receives the interface pointer of the **StringsCollection** object. **additionalDirectories* should not refer to any valid object. The **StringsCollection** is created internally by this method. This object contains the list of full paths to the additional directories that were generated during export. Must not be NULL.

Return Values

If export was interrupted by the user, this method returns E_ABORT. It may also return standard return values of ABBYY FineReader Engine functions.

Remarks

- This method returns two lists of full paths to the additional files and additional directories that were generated during export. For example, when an HTML file is written, additional picture files may appear together with it. The list of files does not include the exported file itself.

- To analyze and recognize pages that will be exported into a single file, specify identical values for all the properties of the **SynthesisParamsForPage** object except for the properties **CorrectDynamicRange**, **DetectBackgroundColor**, **DetectTextColor**.

See also

Exporter
IEngine::ExportPages
IExporter::ExportPagesEx
IFRDocument::ExportPages
Working with Profiles

ExportPagesEx Method of the Exporter Object

This method saves recognized text from several pages into a file in external format. Available file formats are represented by the **FileExportFormatEnum** enumeration constants. This method differs from the **IExporter::ExportPages** in that it is optimized by memory consumption. It requires interface of user-implemented object of type **RecognizedPages**, as its input parameter. This object allows you to pass recognized texts and images of the exported pages one-by-one rather than as the batch, and thus requires memory for only one recognized page at a time.

This method reports events to listeners attached to the **IConnectionPointContainer** interface supported by the **Exporter** object.

Visual Basic Syntax

```
Method ExportPagesEx(
  format                As FileExportFormatEnum,
  fileName              As String,
  recognizedPages       As RecognizedPages,
  exportParams          As Unknown,
  documentInfo          As DocumentInfo
  additionalFiles       As StringsCollection,
  additionalDirectories As StringsCollection
) As StringsCollection
```

C++ Syntax

```
HRESULT ExportPagesEx(
  FileExportFormatEnum format,
  BSTR                 fileName,
  IRecognizedPages*    recognizedPages,
  IUnknown*            exportParams,
  IDocumentInfo*       documentInfo,
  IStringsCollection** additionalFiles,
  IStringsCollection** additionalDirectories
);
```

Parameters

format

[in] This variable specifies the format of the output file. See the **FileExportFormatEnum** description to find out the supported file formats.

fileName

[in] This variable contains the full path to the output file. If this file already exists, it is overwritten without prompt.

recognizedPages

[in] This variable refers to the interface of the user-implemented object of the type **RecognizedPages** which is used to pass recognized texts and images of the exported pages one-by-one.

exportParams

[in] Pass the export parameters object of type corresponding to your file format through this input parameter. For example, if you are creating an RTF file, create the **RTFExportParams** object, set necessary parameters in it, and pass to this method as the *exportParams* input parameter. This parameter may be 0, in which case default values for the export parameters are used, or, if a profile has been loaded, the parameters set by this profile are used.

documentInfo

[in] This variable refers to the **DocumentInfo** object. You should use the same **DocumentInfo** object, which was used as a parameter in the **SynthesizePages** or **SynthesizePagesEx** methods of the **Engine** object. In this case, all the information about document which was received during synthesis is used during export. This parameter may be 0, in which case the text attributes which were detected during synthesis are not available.

additionalFiles

[out] A pointer to the **IStringsCollection*** pointer variable that receives the interface pointer of the **StringsCollection** object. **additionalFiles* should not refer to any valid object. The **StringsCollection** is created internally by this method. This object contains the list of full paths to additional files that were generated during export. Must not be NULL.

additionalDirectories

[out] A pointer to the **IStringsCollection*** pointer variable that receives the interface pointer of the **StringsCollection** object. **additionalDirectories* should not refer to any valid object. The **StringsCollection** is created internally by this method. This object contains the list of full paths to the additional directories that were generated during export. Must not be NULL.

Return Values

If export was interrupted by user, this method returns E_ABORT. It may also return standard return values of ABBYY FineReader Engine functions.

Remarks

- This method returns two lists of full paths to the additional files and additional directories that were generated during export. For example, when an HTML file is written, additional picture files may appear together with it. The list of files does not include the exported file itself.

- To analyze and recognize pages that will be exported into a single file, specify identical values for all the properties of the **SynthesisParamsForPage** object except for the properties **CorrectDynamicRange**, **DetectBackgroundColor**, **DetectTextColor**.

See also

Exporter
IExporter::ExportPages
Working with Profiles

ScanManager Object (IScanManager Interface)

This object exposes a set of properties and methods required to perform scanning.

This object may be declared *WithEvents* in Visual Basic. For C++ user this fact means that it supports the **IConnectionPointContainer** interface. To receive notification events during scanning, a C++ user should create an object derived from the **IScanManagerEvents** interface, then set up the connection between it and events source implemented in **ScanManager** object by standard COM means.

It is worth noting that this object requires a special implementation of the **IScanManagerEvents** interface methods which should process Windows messages. This issue is described in detail in the **IScanManagerEvents** article.

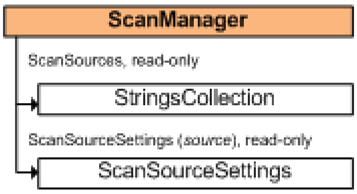
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
ScanOptionsInterfaceType	**ScanOptionsInterfaceTypeEnum**	Specifies the interface type for the scanning options. By default, this property is set to SOIT_Twain.
ScanSources	**StringsCollection**, read-only	Returns a list of scan sources available on the current workstation.
ScanSourceSettings	**ScanSourceSettings**	Provides access to the scanning options of a source (scanner). See for details Setting up Scanning Options.

Methods

Name	Description
Scan	Scans one or several images using the specified scan source into the specified folder on the disk.

Related objects



Output parameter

This object is the output parameter of the **CreateScanManager** method of the **Engine** object.

See also

Setting up Scanning Options
Working with Connectable Objects
IScanManagerEvents
Working with Properties

ScanSourceSettings Property of the ScanManager Object

This property of the **ScanManager** object provides access to the scanning options of a source (scanner). The name of the source is passed as the parameter.

Visual Basic Syntax

```
Property ScanSourceSettings(
   source As String
)As ScanSourceSettings
```

C++ Syntax

```
HRESULT get_ScanSourceSettings(
   BSTR                  source
   IScanSourceSettings** settings

);
HRESULT put_ScanSourceSettings(
   BSTR                 source
   IScanSourceSettings* newSet
);
```

Parameters

source

[in] This variable contains available scan source.

settings

[out] A pointer to the **IScanSourceSettings*** pointer variable that receives the interface pointer of the **ScanSourceSettings** object that contains the scan settings.

newSet

[in] This variable refers to the **ScanSourceSettings** object that contains the scan settings.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

ScanManager
ScanSourceSettings
Setting up Scanning Options
Working with Properties

Scan Method of the ScanManager Object

This method performs scanning of one or several images with the specified scanning source into the specified folder on the disk. The **IScanManagerEvents** interface methods may be called for this method, if there are any listeners attached to the **IConnectionPointContainer** interface supported by the **ScanManager** object.

Visual Basic Syntax

```
Method Scan(
  scanSource     As String,
  destFolder     As String,
  multiplePages  As Boolean
) As StringsCollection
```

C++ Syntax

```
HRESULT Scan(
  BSTR                 scanSource,
  BSTR                 destFolder,
  VARIANT_BOOL         multiplePages,
  IStringsCollection** scannedImages
);
```

Parameters

scanSource

[in] This variable specifies the name of scanning source with which to perform scanning. This name should be one of the collection returned by the **IScanManager::ScanSources** property.

destFolder

[in] This variable contains the full path to the output folder, where scanned images should be stored.

multiplePages

[in] This variable of the Boolean type specifies whether multiple pages should be scanned at a time.

scannedImages

[out] A pointer to **IStringsCollection*** pointer variable that receives the interface pointer of the **StringsCollection** object. **pVal* should not refer to any valid object. The **StringsCollection** is created internally by this method. This object contains the list of full paths to the image files that were received from the scanner.

Return Values

This method may return standard return values of ABBYY FineReader Engine functions.

Remarks

If no listeners are attached to the **IConnectionPointContainer** interface supported by the **ScanManager** object, Windows messages are processed internally by ABBYY FineReader Engine. This is needed to avoid an effect that the application "is not responding" during scanning. It is recommended to use ABBYY FineReader Engine internal Windows message processing, if an ability to break scanning is needed.

The image files created by this method after scanning are saved in one of the formats which is supported both by the scanner and ABBYY FineReader Engine (commonly it is BMP format).

The Scan Manager may show message boxes if an error occurs (no paper in the scanner, wrong resolution, bad TWAIN dll version, etc).

See also

IScanManagerEvents

ScanSourceSettings Object (IScanSourceSettings Interface)

This object provides access to the scanning settings of a source.

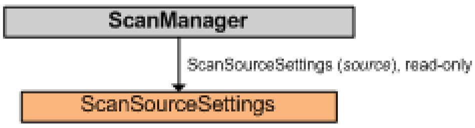
Note: By default, the scanning area rectangle is not set (all the properties **PaperBottom**, **PaperLeft**, **PaperRight**, **PaperTop** are set to 0). In this case, the scanning area will be selected by the scanner. In most cases it will be the whole available scanning area.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Brightness	**Long**	Specifies the brightness of the scanned image (0-100). It is only valid when the **BrightnessControl** property is set to SBC_Manual. By default, this property is set to 50.
BrightnessControl	**ScanBrightnessControlEnum**	Sets brightness control mode. By default, this property is set to SBC_Fine.
Delay	**Long**	Sets the pause between pages in sec. It is only valid if **PauseBetweenPages** = TRUE. By default, this property is set to 5 sec.
DuplexMode	**Boolean**	Specifies whether duplex scanning must be used. By default, this property is set to FALSE.

PaperBottom	**Long**	Sets the coordinate of the bottom border of the scanning area rectangle (in milli-inch). It is only valid if **PaperSize** = SPS_Custom. By default, this property is set to 0.
PaperLeft	**Long**	Sets the coordinate of the left border of the scanning area rectangle (in milli-inch). It is only valid if **PaperSize** = SPS_Custom. By default, this property is set to 0.
PaperRight	**Long**	Sets the coordinate of the right border of the scanning area rectangle (in milli-inch). It is only valid if **PaperSize** = SPS_Custom. By default, this property is set to 0.
PaperSize	**ScanPaperSizeEnum**	Sets paper size. By default, this property is set to SPS_None.
PaperTop	**Long**	Sets the coordinate of the top border of the scanning area rectangle (in milli-inch). It is only valid if **PaperSize** = SPS_Custom. By default, this property is set to 0.
PauseBetweenPages	**Boolean**	Specifies whether the program must pause between pages during scanning. The length of the pause can be specified in the **Delay** property. By default, this property is set to FALSE.
PictureMode	**ScanPictureModeEnum**	Sets image type (black-and-white, gray, color). By default, this property is set to SPM_BlackAndWhite.
Resolution	**Long**	Sets image resolution (from 200 to 600). The resolution must be set to a number divisible by 100. By default, this property is set to 300.
RotationAngle	**ScanPageRotationAngleEnum**	Sets image rotation angle (once the image has been scanned). By default, this property is set to SPRA_Rotation0.
StopBetweenPages	**Boolean**	Specifies whether the program must stop between pages during scanning. By default, this property is set to FALSE. This property cannot be set to TRUE, if the **IScanManager::ScanOptionsInterfaceType** is set to SOIT_None.
UseFeeder	**Boolean**	Specifies whether an automatic document feeder must be used. By default, this property is set to FALSE.

Related objects



See also

ScanManager
Setting up Scanning Options
Working with Properties

IDocumentAnalyzerEvents Interface

This is callback interface that is used for reporting events from the **DocumentAnalyzer** object to the listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to receive notifications from the **DocumentAnalyzer** object should declare it *WithEvents* and implement the following Sub's:

```
Public WithEvents da As FREngine.DocumentAnalyzer
Private Sub da_OnProgress(ByVal Percentage As Long,
                          ByRef Cancel     As Boolean)
...
End Sub
Private Sub da_OnRecognizerTip(ByVal Tip    As String,
                               ByRef Cancel As Boolean)
...
End Sub
```

```
Private Sub da_OnRegionProcessed(ByVal RecognitionPassNumber As Long,
                                 ByRef Region                As Region,
                                 ByRef Cancel                As Boolean)
...
End Sub
```

An object receiving notifications through this interface's methods may do the following inside the methods' implementation:

- Process any Windows messages, which is useful in applications having User Interface, to avoid an effect that the application "is not responding" during long operations.
- Report percentage of recognition and/or fill up the already recognized parts of the image being recognized, with a color, as it is done in ABBYY FineReader.
- Report recognizer tips to the user.

Methods

Name	Description
OnProgress	Delivers to the client information about approximate percentage of analysis or recognition.
OnRecognizerTip	Delivers to the client recognizer tips.
OnRegionProcessed	Delivers to the client information about a rectangle on image that has been recognized.

See also

DocumentAnalyzer
Working with Connectable Objects

OnProgress Method of the IDocumentAnalyzerEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **DocumentAnalyzer** object. It delivers to the client an information about approximate percentage of analysis or recognition. Its implementation may show a progress indicator, as it is done in ABBYY FineReader. It may also process any Windows messages to avoid an effect that the application "is not responding" during long analysis or recognition operations.

Visual Basic Syntax

```
Sub OnProgress(
  ByVal percentage As Long,
  ByRef cancel     As Boolean
)
```

C++ Syntax

```
HRESULT OnProgress(
  long          percentage,
  VARIANT_BOOL* cancel
);
```

Parameters

percentage

[in] This parameter contains the percent of the work currently done. It is in the range from 0 to 100.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process should be terminated. In this case the processing function, that reports the percentage, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *Cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

See also

DocumentAnalyzer
DocumentAnalyzerEvents

OnRegionProcessed Method of the IDocumentAnalyzerEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **DocumentAnalyzer** object. It delivers to the client information about a rectangle on image that has been analyzed or recognized. Its implementation may fill up parts of images with color, as it is done in ABBYY FineReader. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnRegionProcessed(
  ByVal recognitionPassNumber As Long,
  ByRef region                As Region,
  ByRef cancel                As Boolean
)
```

C++ Syntax

```
HRESULT OnRegionProcessed(
  long          recognitionPassNumber,
  IRegion*      region,
  VARIANT_BOOL* cancel
);
```

Parameters

recognitionPassNumber

[in] This parameter reports the number of the recognition pass. It may be either 1 or 2. Rectangles from different passes may be filled up with different colors as it is done in ABBYY FineReader.

region

[in] This parameter contains coordinates of the rectangle that has been recognized. These coordinates relate to the deskewed black-and-white plane of the image.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process of recognition should be terminated. In this case the processing function, that reports the rectangle, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *Cancel* parameter is ignored.

Remarks

- The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.
- This method reports the rectangle that was recognized since the last call to this method, and not a cumulative recognized rectangle.

See also

DocumentAnalyzer
IDocumentAnalyzerEvents

OnRecognizerTip Method of the IDocumentAnalyzerEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for some of the methods of the **DocumentAnalyzer** object. Its implementation may report recognizer tips to the user. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub OnRecognizerTip(
  ByVal tip    As String,
  ByRef cancel As Boolean
)
```

C++ Syntax

```
HRESULT OnRecognizerTip(
  BSTR          tip,
  VARIANT_BOOL* cancel
);
```

Parameters

tip

[in] This parameter contains the recognizer tip.

cancel

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process of recognition should be terminated. In this case the processing function, that reports the tip, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *Cancel* parameter is not taken into account.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

See also

DocumentAnalyzer
IDocumentAnalyzerEvents

IExporterEvents Interface

This is callback interface that is used for reporting events from the **Exporter** object to the listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to receive notifications from the **Exporter** object should declare it *WithEvents* and implement the following Sub:

```
Public WithEvents exporter As FREngine.Exporter
Private Sub exporter_ReportPercentage(ByVal percentage      As Long,
                                      ByRef shouldTerminate As Boolean)
...
End Sub
```

An object receiving notifications through this interface's methods may do the following inside the methods' implementation:

- Process any Windows messages, which is useful in applications having user interface, to avoid an effect that the application "is not responding" during long operations.
- Report percentage of export, as it is done in ABBYY FineReader.

Methods

Name	Description
ReportPercentage	Delivers to the client information about percentage of the export performed.

See also

Working with Connectable Objects
Exporter

ReportPercentage Method of the IExporterEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for the **IExporter::ExportPages** method. It delivers to the client information about the percentage of export performed. It is called once for each exported page. Its implementation may show a progress indicator, as it is done in ABBYY FineReader. It may also process any Windows messages to avoid an effect that the application "is not responding" during long operations.

Visual Basic Syntax

```
Sub ReportPercentage(
  ByVal percentage      As Long,
  ByRef shouldTerminate As Boolean
)
```

C++ Syntax

```
HRESULT ReportPercentage(
  long          percentage,
  VARIANT_BOOL* shouldTerminate
);
```

Parameters

percentage

[in] This parameter contains the percent of the work currently done. It is in the range from 0 to 100.

shouldTerminate

[in, out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the process of export should be terminated. In this case the **IExporter::ExportPages** method, that reports the percentage, returns E_ABORT.

Return Values

[C++ only] If this method returns a value other than S_OK, it indicates that an error occurred on the client side, and in this case the value of the *shouldTerminate* parameter is ignored.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results.

See also

IExporterEvents
Exporter

IScanManagerEvents Interface

This is a callback interface that is used for interaction of the **ScanManager** object with its listeners. This interface is implemented on the client side. As it derives from the **IUnknown** interface, the client object should also implement the **IUnknown** methods. This interface is designed primarily for using in C++. Visual Basic users that want to implement listeners for the **ScanManager** object should declare it *WithEvents* and implement the following Sub:

```
Public WithEvents scanManager As FREngine.ScanManager
Private Sub scanManager_NewImage(ByVal scannedImage As String)
   DoEvents
   ...
End Sub
Private Sub scanManager_ScanStopped()
   DoEvents
   ...
End Sub
```

An object connected to this callback interface should process Windows messages. This is absolutely necessary, or the scanning will not function. When no listeners are connected to the **IScanManagerEvents** interface, the Windows messages are processed internally by ABBYY FineReader Engine. When any listeners are attached, they should process Windows messages themselves. In Visual Basic it is done by the **DoEvents** statement. For C++ users the following special kind of message loop is recommended:

```
   MSG msg;
   while(::PeekMessage(&msg, 0, 0, 0, PM_REMOVE | PM_NOYIELD)) {
      if(WM_MOUSEFIRST <= msg.message && msg.message <= WM_MOUSELAST) {
         continue;
      }
      if(WM_NCMOUSEMOVE <= msg.message && msg.message <= WM_NCMBUTTONDBLCLK) {
         continue;
      }
      if(WM_KEYFIRST <= msg.message && msg.message <= WM_KEYLAST) {
         continue;
      }
      ::DispatchMessage(&msg);
   }
```

Note that all messages are removed from the message queue, but not all of them are processed. In particular, all user input (both from keyboard and from the mouse) is not processed. This eliminates the possibility that user may do something wrong with our window during the scanning (close it or choose something from the menu for example).

Methods

Name	Description
NewImage	Provides information about the name of the file with the scanned image. Allows you to stop multipage scanning.
ScanStopped	Provides information about whether the scanning was stopped.

See also

ScanManager
Working with Connectable Objects

NewImage Method of the IScanManagerEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for the **IScanManager::Scan** method. Its implementation should process Windows messages or the scanning will not function correctly. This method allows a user to receive a name of the file with the scanned image and to break multipage scanning.

Visual Basic Syntax

```
Sub NewImage(
  ByVal scannedImage   As String,
  ByRef CancelScanning As Boolean
)
```

C++ Syntax

```
HRESULT NewImage(
  BSTR          scannedImage,
  VARIANT_BOOL* CancelScanning
);
```

Parameters

scannedImage

[in] This parameter contains the name of the file with the scanned image.

CancelScanning

[out] You may set this variable to TRUE (VARIANT_TRUE) to indicate that the scanning process should be terminated. In this case the scanning function, that reports the tip, returns E_ABORT.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results. The client implementation should process Windows messages. In Visual Basic this is done via the **DoEvents** statement, while for C++ users it is recommended to use the special form of message loop described on the **IScanManagerEvents** page.

See also

ScanManager
IScanManagerEvents

ScanStopped Method of the IScanManagerEvents Interface

This method is implemented on the client side. It is called by ABBYY FineReader Engine for the **IScanManager::Scan** method. Its implementation should process Windows messages or the scanning will not function correctly. This method provides to the client an information about whether the scanning was stopped.

Visual Basic Syntax

```
Sub ScanStopped()
```

C++ Syntax

```
HRESULT ScanStopped();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The client implementation of this method must assure that no exceptions are thrown inside it, as it may lead to unpredictable results. The client implementation should process Windows messages. In Visual Basic this is done via the **DoEvents** statement, while for C++ users it is recommended to use the special form of message loop described on the **IScanManagerEvents** page.

See also

ScanManager
IScanManagerEvents

Parameter Objects

The setting up of parameters of layout analysis, recognition, synthesis and export is performed via the so called ABBYY FineReader Engine parameter objects.

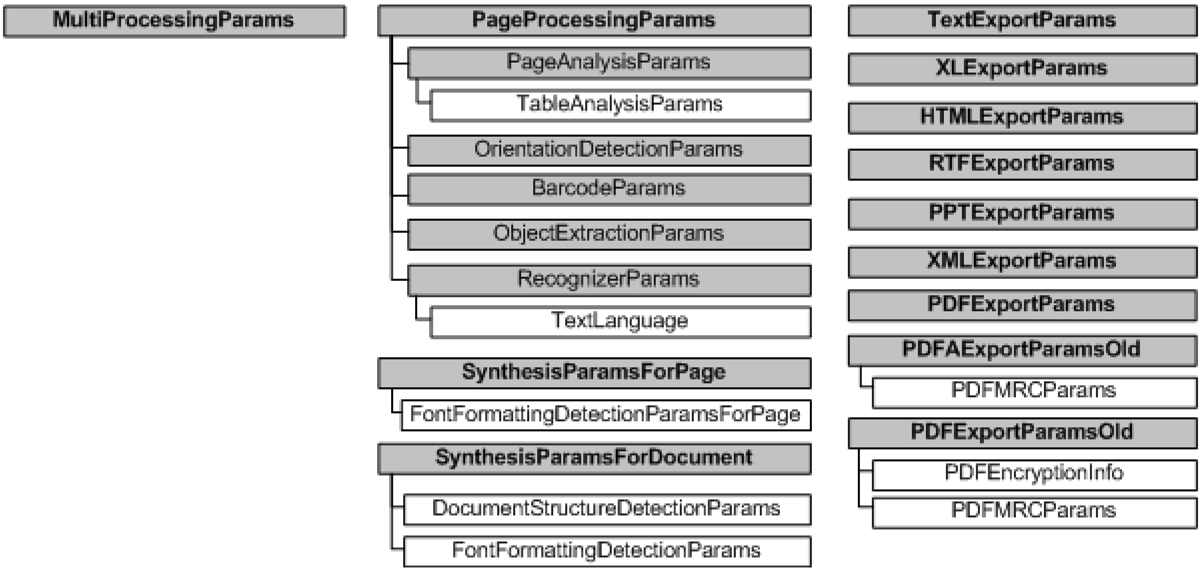
This section contains descriptions of the following parameter objects:

- **MultiProcessingParams**
- Analysis, recognition, and synthesis parameters:
  - **PageProcessingParams**
  - **PageAnalysisParams**
  - **TableAnalysisParams**
  - **BarcodeParams**
  - **RecognizerParams**
  - **ObjectsExtractionParams**
  - **OrientationDetectionParams**
  - **SynthesisParamsForDocument**
  - **DocumentStructureDetectionParams**
  - **FontFormattingDetectionParams**
  - **SynthesisParamsForPage**
  - **FontFormattingDetectionParamsForPage**
- Export parameters:
  - **HTMLExportParams**
  - **PPTExportParams**

- **RTFExportParams**
- **TextExportParams**
- **XLExportParams**
- **XMLExportParams**
- **PDFExportParams**
- **PDFAExportParamsOld**
- **PDFExportParamsOld**
- **PDFEncryptionInfo**
- **PDFMRCParams**

The additional information you can find in the Tuning Analysis, Recognition, and Synthesis Parameters and Tuning Export Parameters sections.

The parameter objects hierarchy



For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

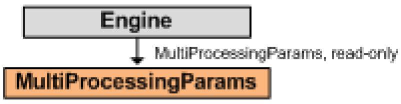
MultiProcessingParams Object (IMultiProcessingParams Interface)

This object provides access to the parameters of multiprocessing and multiple CPU cores usage. The main parameter which defines whether multiprocessing is to be used is the **MultiProcessingMode** property. All other properties regulate the number of processes and CPU cores and are taken into account only if the **MultiProcessingMode** property is set to MPM_Auto or MPM_Parallel.

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
MultiProcessingMode	**MultiProcessingModeEnum**	Specifies whether ABBYY FineReader Engine should distribute analysis and recognition of multi-page documents to CPU cores. The maximum number of processes which can be run equals to the value of the **RecognitionProcessesCount** property. By default the property is set to MPM_Auto.
RecognitionProcessesCount	**Long**	Specifies the number of processes which can be run. The maximal possible value of the property is 32. By default this property is 0 which means that the number of processes will be equal to the minimum of the following values: • number of available physical or logical CPU cores (depending on the value of the

		UseOnlyPhysicalCPUCores property), • number of free CPU cores available in the license, • number of pages in the processing document. If you change the value of this property, ABBYY FineReader Engine immediately allocates CPU cores of the license and loads the FineReader Engine Processor module. If the value of this property is 0, CPU cores allocation and loading of the FineReader Engine Processor module will be performed when it will be necessary.
SharedCPUCoresMask	**Long**	Specifies the CPU cores, which can be used in shared mode of CPU cores usage, as an affinity mask. The property makes sense only if the value of the **SharedCPUCoresMode** property is TRUE. By default all detected CPU cores are used.
SharedCPUCoresMode	**Boolean**	Specifies whether the CPU cores are used in shared mode. There are two modes of CPU cores usage: separate and shared. In separate mode ABBYY FineReader Engine uses no more processes than it is allowed by the license. In shared mode any number of processes can be run, but all these processes will use only the CPU cores specified by the **SharedCPUCoresMask** property. By default the property is set to FALSE, which means that the separate mode is used.
UseOnlyPhysicalCPUCores	**Boolean**	Specifies whether only physical CPU cores or physical and logical CPU cores are used during processing. By default the property is set to TRUE, which means that only physical CPU cores are used.

Related objects



See also

Properties of the Engine Object
Working with Properties

PageProcessingParams Object (IPageProcessingParams Interface)

This object is used for tuning different parameters of layout analysis and recognition. It comprises child objects of **PageAnalysisParams**, **RecognizerParams**, **BarcodeParams**, **OrientationDetectionParams** and **ObjectsExtractionParams** types that are available through the corresponding properties. A pointer to this object is passed to all layout analysis and analysis-recognition functions along with other parameters.

The **PageProcessingParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

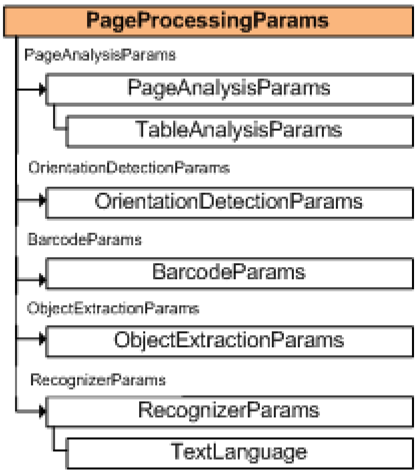
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BarcodeParams	**BarcodeParams**	Provides access to the subset of page processing parameters, affecting the process of barcode blocks recognition.
DetectBarcodes	**Boolean**	Specifies if barcodes are detected, and accordingly barcode blocks created, during page processing. If this property is FALSE, barcodes may be detected as blocks of some other type (e.g. pictures). This property is FALSE by default.

DetectInvertedImage	**Boolean**	The property is obsolete. We recommend that you keep its default value. This property set to TRUE tells ABBYY FineReader Engine to detect whether the image is inverted (white text against black background). The text color is detected during page processing, and if it differs from normal, ABBYY FineReader Engine automatically inverts the image. This property is FALSE by default.
DetectOrientation	**Boolean**	If this property is TRUE, the page orientation is detected during page processing, and if it differs from normal, ABBYY FineReader Engine automatically rotates the image. This property is FALSE by default.
ObjectsExtractionParams	**ObjectsExtractionParams**	Provides access to the subset of page processing parameters that affect extraction of objects.
OrientationDetectionParams	**OrientationDetectionParams**	Provides access to the parameters of orientation detection.
PageAnalysisParams	**PageAnalysisParams**	Provides access to the subset of page processing parameters that affect the process of page analysis. These parameters are ignored, if the value of the **PerformPageAnalysis** property is FALSE.
PerformPageAnalysis	**Boolean**	Specifies if page analysis is to be performed. If this property is FALSE, the **PageAnalysisParams** property is ignored. This property is TRUE by default.
RecognizerParams	**RecognizerParams**	Provides access to the subset of page processing parameters that affect the process of page recognition.
RemoveGeometricalDistortions	**Boolean**	Specifies if geometrical distortions (perspective on photos, curved lines from scanned books, etc.) should be removed during layout analysis. This property is FALSE by default.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Sample

Visual C++ (COM) code

```
// Global ABBYY FineReader Engine object.
FREngine::IEnginePtr Engine;
...
// Global ABBYY FineReader Engine object.
 FREngine::IEnginePtr Engine;
 ...
 // Open the image file
 FREngine::IImageDocumentPtr pImageDoc =
  Engine->PrepareAndOpenImage( L"D:\\Demo.tif", 0, 0, 0 );

 // Create the Layout object
 FREngine::ILayoutPtr pLayout = Engine->CreateLayout();
 // Create page processing parameters
 FREngine::IPageProcessingParamsPtr pPageProcessingParams =
  Engine->CreatePageProcessingParams();

 // Now tune it
 pPageProcessingParams->DetectBarcodes = VARIANT_TRUE;

 // Analyze and recognize the image
 Engine->AnalyzeAndRecognizePage( pImageDoc, pPageProcessingParams, 0, pLayout, 0 );
```

Visual Basic code

```
' Global ABBYY FineReader Engine object.
 Public Engine As FREngine.Engine
 ...
 ' Open the image file
 Dim ImageDoc As FREngine.ImageDocument
 Set ImageDoc = Engine.PrepareAndOpenImage("D:\Demo.tif")

 ' Create the Layout object
 Dim Layout As FREngine.Layout
 Set Layout = Engine.CreateLayout()
 ' Create page processing parameters
 Dim PageProcessingParams As FREngine.PageProcessingParams
 Set PageProcessingParams = Engine.CreatePageProcessingParams
 ' Now tune it
 PageProcessingParams.DetectBarcodes = True

 ' Perform page analysis
 Engine.AnalyzeAndRecognizePage ImageDoc, PageProcessingParams, Nothing, Layout
```

Output parameter

This object is the output parameter of the **CreatePageProcessingParams** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **AnalyzePage**, **AnalyzePages**, **AnalyzeAndRecognizePage**, **AnalyzeAndRecognizePages**, **RecognizeImageDocumentAsPlainText**, **RecognizeImageAsPlainText**, **RecognizeImageFile** of the **Engine** object.

- **AnalyzePage**, **AnalyzePages**, **AnalyzeRegion**, **AnalyzeTable**, **AnalyzeAndRecognizePage**, **AnalyzeAndRecognizePages**, **RecognizeImageDocumentAsPlainText** of the **DocumentAnalyzer** object.

- **Analyze**, **AnalyzePages**, **AnalyzeAndRecognize**, **AnalyzeAndRecognizePages**, **Process** of the **FRDocument** object.

- **Analyze**, **AnalyzeAndRecognize**, **AnalyzeTable**, **AnalyzeRegion** of the **FRPage** object.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
Working with Properties

See sample: CustomLanguage

PageAnalysisParams Object (IPageAnalysisParams Interface)

This object provides access to parameters used for tuning the layout analysis process. It is passed as a member of the **PageProcessingParams** object into layout analysis and layout analysis-recognition functions.

The **PageAnalysisParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

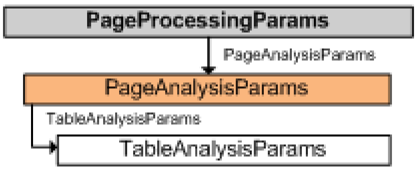
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DetectPictures	**Boolean**	If this property is TRUE, the pictures are detected during layout analysis. This property is TRUE by default.
DetectSeparators	**Boolean**	If this property is TRUE, the separators are detected during layout analysis. This property is TRUE by default.
DetectVectorGraphics	**Boolean**	If this property is TRUE, the vector pictures are detected during layout analysis. Vector picture blocks may appear in the layout only if this property has been set to TRUE during layout analysis. This property is TRUE by default.
DetectTables	**Boolean**	If this property is TRUE, the tables are detected during layout analysis. This property is TRUE by default.
NoShadowsMode	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to presume that an image has no shadows from scanning. This property is FALSE by default.
ProhibitDoublePageMode	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to presume that an image is not a book double page. This property is FALSE by default.
ProhibitModelAnalysis	**Boolean**	If this property is FALSE, typical variants of page layout will be gone through during page analysis and the best variant will be selected, which can improve recognition quality. If the best variant of page layout cannot be selected, standard page layout analysis will be performed. This property is FALSE by default.
SingleColumnMode	**Boolean**	If this property is set to TRUE, the analysis procedure presumes that there is only one column of text on a page. This property is FALSE by default. The value of this property is ignored, if the **ProhibitModelAnalysis** property is set to FALSE.
TableAnalysisParams	**TableAnalysisParams**	Provides access to the subset of page processing parameters that affect the process of table analysis.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreatePageAnalysisParams** method of the **Engine** object.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
PageProcessingParams
Working with Properties

TableAnalysisParams Object (ITableAnalysisParams Interface)

This object provides access to parameters affecting table block analysis process. All properties of a newly created object of this type are set to reasonable defaults. To know about the default value of this or that property, see its description.

The **TableAnalysisParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

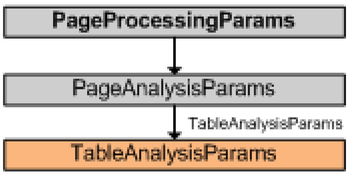
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
DetectCellsInversion	**Boolean**	If this property is TRUE, the cells inversion is detected during table block analysis. This property is TRUE by default.
DetectCellsOrientation	**Boolean**	If this property is TRUE, the cells orientation is detected during table block analysis. This property is TRUE by default.
SingleLinePerCell	**Boolean**	Set this property to TRUE if you only recognize tables with one line of text per each cell. The table layout will be analyzed more readily. This property is FALSE by default.
SplitOnlyBySeparators	**Boolean**	Set this property to TRUE if you only recognize tables with no hidden separators. The table layout will be analyzed more readily. This property is FALSE by default.

Methods

Name	**Description**
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.
SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreateTableAnalysisParams** method of the **Engine** object.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
PageAnalysisParams
Working with Properties

BarcodeParams Object (IBarcodeParams Interface)

This object allows you to tune the parameters of barcode block recognition. Each barcode block in layout has its own child object of **BarcodeParams** type. Besides, this object is passed as a sub-object of the **PageProcessingParams** object into ABBYY FineReader Engine layout analysis-recognition functions. Recognition functions use the barcode recognition parameters specified by barcode blocks' child objects of the **BarcodeParams** type, rather than those specified by the sub-object of the **PageProcessingParams** object passed to these functions.

Whenever a barcode block is created during layout analysis, the properties of its child object of the **BarcodeParams** type are initialized with the values of the **BarcodeParams** object properties passed to the layout analysis function. Properties of a barcode block which is created with the help of the **AddBlock** or **InsertBlock** methods of the **Layout** object are set to reasonable defaults. See the description of a particular property for the information on its default value.

The **BarcodeParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

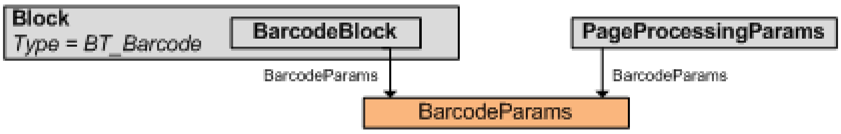
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
HasChecksum	**Boolean**	Specifies whether the barcode being recognized must be interpreted as the barcode of the same type but with a check sum. This property is only available for barcodes of types Code 39, Interleaved 2 of 5, Codabar, and Matrix 2 of 5. By default, this property is set to FALSE. **Note:** While Codabar has no check digit, ABBYY FineReader Engine uses an algorithm for computing check digits according to Modulo 16. The check digit is computed as follows. Each Codabar character has a value assigned to it. The sum of all character values is taken, including the Start and the Stop characters. The data character whose value, when added to this sum, equals a multiple of 16 is the check digit.
IsCode39WithoutAsterisk	**Boolean**	Specifies that the Code 39 barcode being recognized has no start and stop symbol, the asterisk "*". By default, this property is set to FALSE.
Orientation	**Long**	The value of this property is an OR superposition of the **BarcodeOrientationEnum** enumeration constants which denote the types of barcode orientation. For example, if it is set to BO_Left_To_Right | BO_Down_To_Top, ABBYY FineReader Engine will presume that barcode blocks may be oriented either from left to right or from down to top, ignoring all other variants. By default, this property is set to BO_Autodetect, i.e. ABBYY FineReader Engine will detect the barcode orientation automatically.
PDF417CodePage	**CodePageEnum**	This property is used to recognize barcodes which do not conform to the barcode specifications. Do not use this property for barcodes created in conformity with the barcode specifications. Some barcode printers use code pages other than US-MSDOS required by the specifications. In this case, use this property to specify the code page which was used by the barcode printer to create the barcode. In most cases this will be the code page of the operating system under which the barcode printer was running. By default, this property is set to CP_Null.
SupplementType	**Long**	The value of this property is an OR superposition of the **BarcodeSupplementTypeEnum** enumeration constants. This property is only available for barcodes of the EAN 8, 13, UPC-A, and UPC-E types. For example, if it is set to BS_Void | BS_2Digits, ABBYY FineReader Engine will try to recognize barcode blocks either without supplementary barcode or with 2-digit

		supplementary barcode. By default, this property is set to BS_Autodetect, i.e. ABBYY FineReader Engine will detect the supplementary barcode type automatically.
Type	**Long**	The value of this property is an OR superposition of the **BarcodeTypeEnum** enumeration constants which denote the types of barcodes. For example, if it is set to BT_EAN13 | BT_EAN8, ABBYY FineReader Engine will try to recognize barcode blocks in either EAN 13 or EAN 8 standard, ignoring all other variants. By default, this property is set to BT_Autodetect**,** i.e. ABBYY FineReader Engine will detect the barcode type automatically.

Methods

Name	**Description**
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents to a file on disk.
SaveToMemory	Saves the object contents to the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreateBarcodeParams** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **ExtractBarcodes** method of the **DocumentAnalyzer** object
- **ExtractBarcodes** method of the **FRPage** object

See also

Barcode Types
Tuning Analysis, Recognition, and Synthesis Parameters
Working with Properties

RecognizerParams Object (IRecognizerParams Interface)

This object allows you to tune the recognition parameters. Each text block and table cell in layout has its own child object of the **RecognizerParams** type. Besides, this object is passed as a sub–object of the **PageProcessingParams** object into ABBYY FineReader Engine layout analysis–recognition functions. Recognition functions use parameters of recognition defined by text blocks' and table cells' child objects of the type **RecognizerParams**. Whenever a text block or table cell is created during layout analysis, properties of its child object of the **RecognizerParams** type are initialized with values of properties of the **RecognizedParams** object, passed to analysis function. Properties of a subobject of the block which is created with the help of the **AddBlock** or **InsertBlock** method of the **Layout** object are set to reasonable defaults. To know about the default value of this or that property see its description.

The **RecognizerParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re–created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	**Type**	**Description**

Application	**Engine**, read–only	Returns the **Engine** object.
BalancedMode	**Boolean**	If this property is TRUE, the recognition will run in balanced mode. The balanced mode is an intermediate mode between full and fast modes. The fast mode can be activated with the help of the **FastMode** property. This property is available for machine–printed texts only, for hand–printed texts the recognition will be run in full mode. By default, this property is FALSE.
CaseRecognitionMode	**CaseRecognitionModeEnum**	This property specifies the mode of letter case recognition. By default the value of this property is CRM_AutoCase, which corresponds to automatic case recognition.
CellsCount	**Long**	Specifies the number of character cells for a recognized block. This property is valid only for the handprint recognition. It has a sense only for the field marking types (the **FieldMarkingType** property) that imply splitting the text in cells. Default value for this property is 1, but you should set the appropriate value to recognize the text correctly.
CJKTextDirection	**CJKTextDirectionEnum**	Sets the direction of the text to be recognized. This property is valid only for the hieroglyphic languages. By default, this property is CJKTD_Autodetect.
ErrorHiliteLevel	**ErrorHiliteLevelEnum**	Specifies the level at which the **ICharParams::IsSuspicious** property is set to TRUE for a recognized character. The name of the property reflects the fact that the uncertain characters are highlighted with color in ABBYY FineReader. By default the value of this property is EHL_Standard.
ExactConfidenceCalculation	**Boolean**	If this property is TRUE, character and word confidence will be defined more accurately, but recognition speed may get slower. The value of character confidence is stored in the **CharConfidence** property of the **CharacterRecognitionVariant** and **PlainText** objects. The value of word confidence is stored in the **WordConfidence** property of the **WordRecognitionVariant** object. This property is automatically set to TRUE if the **SaveCharacterRecognitionVariants** or **SaveWordRecognitionVariants** property is TRUE. By default, this property is FALSE.
FastMode	**Boolean**	This property set to TRUE provides 2–2.5 times faster recognition speed at the cost of a moderately increased error rate (1.5–2 times more errors). This property is available both for machine– and hand–printed texts. In the case of a hand–printed text (text type TT_Handprinted), a special recognition mode is used. On good print quality texts, ABBYY FineReader Engine makes an average of 1–2 errors per page, and such moderate increase in error rate can be easily tolerated in many cases, such as full text indexing with "fuzzy" searches, preliminary recognition, etc. By default, this property is FALSE. **Note:** We do not recommend using this mode to recognize small image fragments (for example, fragments which consist of only one line or word) because the time advantage will be insignificant.
FieldMarkingType	**FieldMarkingTypeEnum**	This property specifies the type of marking around letters (for example, underline, frame, box, etc.). This property is valid only for the handprint recognition. By default the value of this property is FMT_SimpleText,

		which means the plain text. **Note:** For correct handprint recognition use **CellsCount** property that allows you to set the number of character cells for a recognized block.
LowResolutionMode	**Boolean**	Specifies whether a text on an image with low resolution is recognized. By default, the value of this property is FALSE.
OneLinePerBlock	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to presume that the text in block to which the current **RecognizerParams** object belongs contains no more than one string. By default this property is FALSE.
OneWordPerLine	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to presume that no text line may contain more than one word, so the lines of text will be recognized as a single word. By default this property is FALSE.
PossibleTextTypes	**LongsCollection**	The property is obsolete. Use the **TextTypes** property instead. This property contains a collection of **TextTypeEnum** values. The property tells ABBYY FineReader Engine to presume that the text to recognize is of one of the types the collection contains. If the value of the **TextType** property is not TT_ToBeDetected, the value of this property will be ignored. The property returns a copy of the collection but not a reference to it. In order to modify the value of the property it is necessary to create a new collection, add required values to it, and then assign the collection to the property. The collection should contain at least one element and cannot contain TT_ToBeDetected. When this property is changed, the **TextType** property is automatically set to TT_ToBeDetected. By default it contains TT_Normal.
ProhibitHyphenation	**Boolean**	This property set to TRUE prohibits recognition of hyphenation from line to line. It is useful when a text with presumably no hyphenations is recognized, in which case it may speed up the recognition. If there exist any hyphenations in the recognized block, and this property is TRUE, the hyphenated words will be recognized incorrectly. By default this property is FALSE.
ProhibitInterblockHyphenation	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to presume that text from one block cannot be carried over to the next block. By default this property is FALSE.
ProhibitItalic	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine not to recognize letters printed with *italic*–styled font. It is useful when a text with presumably no italic letters is recognized, in which case it may speed up the recognition. If there exist any italic letters on the image, and this property is TRUE, these letters will be recognized incorrectly. By default this property is FALSE.
ProhibitSubscript	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine not to recognize subscript letters. It is useful when a text with presumably no subscripts is recognized, in which case it may speed up the recognition. If there exist any subscript letters on the image, and this property is TRUE, these letters will be recognized incorrectly. By default this property is FALSE.
ProhibitSuperscript	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine not to recognize superscript letters. It is useful when a text with presumably no superscripts is recognized, in which case it may speed up the

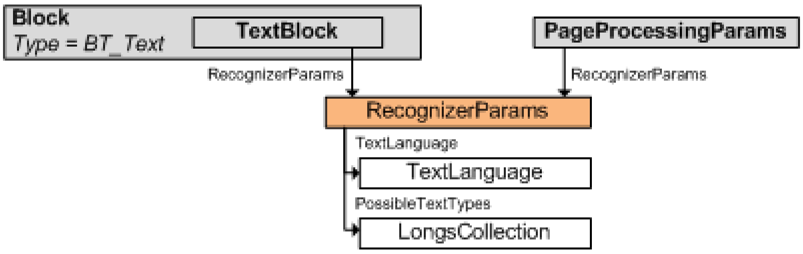
		recognition. If there exist any superscript letters on the image, and this property is TRUE, these letters will be recognized incorrectly. By default this property is FALSE.
SaveCharacterRecognitionVariants	**Boolean**	Specifies whether the variants of characters recognition are saved. The **ICharParams::CharacterRecognitionVariants** property returns a collection of recognition variants for a character. The default value is FALSE. See also **Using Voting API**.
SaveCharacterRegions	**Boolean**	Specifies whether the exact characters regions (**ICharParams::CharacterRegion**) are saved. The default value is FALSE.
SaveWordRecognitionVariants	**Boolean**	Specifies whether the variants of recognition of a word are saved. The **IParagraph::GetWordRecognitionVariants** method and **ICharParams::WordRecognitionVariants** property return a collection of recognition variants for a word. The default value is FALSE. See also **Using Voting API**.
TextLanguage	**TextLanguage**	This property refers to the **TextLanguage** object used for image recognition. By default this parameter is initialized with English language. This property may be easily set via the **SetPredefinedTextLanguage** method.
TextType	**TextTypeEnum**	The property is obsolete. Use the **TextTypes** property instead. This property tells ABBYY FineReader Engine to presume that the text to recognize is of that type. By default the value of this property is TT_Normal. **Note:** If this property is set to TT_ToBeDetected, TT_Handprinted, or TT_Index, the **TrainUserPatterns** property cannot be set to TRUE.
TextTypes	**Long**	The value of this property is an OR superposition of the **TextTypeEnum** enumeration constants which denote possible text types used for recognition. For example, if it is set to TT_Normal | TT_Index, ABBYY FineReader Engine will presume that the text contains only common typographic text and digits written in ZIP–code style, ignoring all other variants. By default, this property is set to TT_Normal. The property cannot be set to TT_ToBeDetected. See also **Using Text Type Autodetection**. **Notes:** • If this property is set to TT_Handprinted, or TT_Index, the **TrainUserPatterns** property cannot be set to TRUE. • If this property is equal to any combination of TT_Matrix, TT_Typewriter, TT_OCR_A, and TT_OCR_B, italic fonts and superscript/subscript will not be recognized, regardless of the values of the **ProhibitItalic**, **ProhibitSubscript** and **ProhibitSuperscript** properties.
TrainUserPatterns	**Boolean**	This property specifies whether user patterns should be trained during the recognition. If this property is TRUE, some user pattern file should be specified in the **UserPatternsFile** property. The **Pattern Training** dialog box will display during recognition. For correct

		operation of pattern training process it is necessary to set the value of the parent window HWND handle (**IEngine::ParentWindow** property). See also **Recognizing with Training**. By default this property is FALSE. If this property is set to TRUE, the **TextType** and **TextTypes** properties cannot be set to TT_ToBeDetected, TT_Handprinted, or TT_Index. **Notes:** Pattern training is not supported for hieroglyphic languages.
UseBuiltInPatterns	**Boolean**	This property set to TRUE means that ABBYY FineReader Engine will use its own built–in patterns for recognition. Patterns are files establishing relationship between character image and character itself. By default this property is TRUE. You may want to set this property to FALSE when you do not want to use standard ABBYY FineReader Engine patterns for character recognition, but user patterns only. This may be useful for recognition of text typed with decorative or non–standard fonts. In this case it is better not to use ABBYY FineReader Engine built–in patterns, but use your own user–defined patterns trained for these fonts. A path to user–defined pattern file is stored in the **UserPatternsFile** property. If the **UserPatternsFile** property is empty the **UseBuiltInPatterns** property is ignored. See also **Recognizing with Training**.
UserPatternsFile	**String**	Contains the full path to a file of the user pattern used for recognition. By default this property stores an empty string. If the value of this property is not empty, information from the user pattern file will be used during recognition. If the **UseBuiltInPatterns** property is FALSE, which means that standard ABBYY FineReader Engine patterns are not used during recognition, this property should contain a path to user–defined pattern file, as only information stored in it will be used. See also **Recognizing with Training**.
WritingStyle	**WritingStyleEnum**	Provides additional information about handprinted letters writing style. By default the value of this property is WS_Default, which means that the writing style is selected depending on the current language of the operating system.

Methods

Name	**Description**
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.
SetPredefinedTextLanguage	Sets the language of recognition to be one of the predefined ABBYY FineReader Engine languages.

Related objects



Output parameter

This object is the output parameter of the **CreateRecognizerParams** method of the **Engine** object.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
Recognizing Handprinted Texts
PageProcessingParams
TextBlock
Working with Properties

See sample: CustomLanguage

SetPredefinedTextLanguage Method of the RecognizerParams Object

This method sets the language of recognition to be one of the predefined ABBYY FineReader Engine languages. It affects the value of the **IRecognizerParams::TextLanguage** property.

Visual Basic Syntax

```
Method SetPredefinedTextLanguage(
  internalName As String
)
```

C++ Syntax

```
HRESULT SetPredefinedTextLanguage(
  BSTR internalName
);
```

Parameters

internalName

[in] This variable is the internal name of one of the ABBYY FineReader Engine predefined languages. This name should be one from the list of ABBYY FineReader Engine predefined languages. This parameter may also contain several languages names divided by commas, for example "English,French,German".

Return Values

In case the predefined language you are trying to set is not available, or the language with the name passed is not supported, the E_INVALIDARG error code is returned. This method may also return standard return values of ABBYY FineReader Engine functions.

Remarks

Availability of this or that predefined language depends on the availability of the corresponding modules in the set of ABBYY FineReader Engine modules.

See also

IRecognizerParams::TextLanguage
Working with Languages

ObjectsExtractionParams Object (IObjectsExtractionParams Interface)

This object provides access to the parameters used for objects extraction. Objects extraction is a process which detects additional objects (e.g. garbage, texture, small text areas of low quality) on an image before recognition.

The **ObjectsExtractionParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

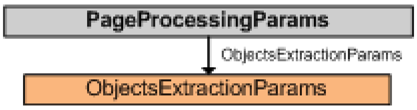
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DetectMatrixPrinter	**Boolean**	If this property is TRUE, the text printed on matrix printer is detected during objects extraction. This property is TRUE by default.
DetectPorousText	**Boolean**	If this property is TRUE, the regions with porous text are detected during objects extraction. This property is TRUE by default.
FastObjectsExtraction	**Boolean**	If this property is TRUE, objects extraction will speed up, but its quality may deteriorate. This property is FALSE by default.
FlexiFormsDA	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to locate all text on the page, including small text areas of low quality and text in diagrams and pictures. Tables are recognized as plain text. This property is FALSE by default.
FullTextIndexDA	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to detect all text on an image, including text embedded into the image. Reading order is not changed to provide ability for further full-text search. This property is FALSE by default.
ProhibitColorImage	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to use only black-and-white plane during objects extraction. In this case detection quality of colored tables and pictures can get worse. This property is FALSE by default.
RemoveGarbage	**Boolean**	Specifies if garbage (excess dots that are smaller than a certain size) is to be removed from the image during objects extraction. This property is FALSE by default.
RemoveTexture	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to remove the background noise from a temporary image used for recognition. The source image remains unaffected. This property is TRUE by default.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreateObjectsExtractionParams** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **Recognize**, **RecognizePages** methods of the **FRDocument** object.
- **Recognize**, **RecognizeBlocks**, **RemoveGeometricalDistortions**, **ExtractBarcodes**, **DetectOrientation**, **FindPageSplitPosition** methods of the **FRPage** object.

- **RecognizePage**, **RecognizePages**, **RecognizeBlocks**, **ExtractBarcodes**, **RemoveGeometricalDistortions**, **DetectOrientation**, **FindPageSplitPosition** methods of the **DocumentAnalyzer** object.

- **RecognizePage**, **RecognizePages** methods of the **Engine** object.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
PageProcessingParams,
Working with Properties

OrientationDetectionParams Object (IOrientationDetectionParams Interface)

This object provides access to the parameters used for tuning the page orientation detection. It is passed as a parameter into the **DetectOrientation** methods of the **DocumentAnalyzer** and **FRPage** objects. Besides, this object is passed as a sub-object of the **PageProcessingParams** object into ABBYY FineReader Engine layout analysis-recognition functions.

The **OrientationDetectionParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

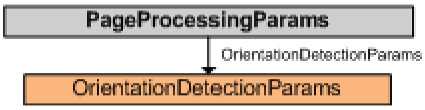
Properties

Name	**Type**	**Description**
Application	**Engine**, read-only	Returns the **Engine** object.
OrientationDetectionMode	**OrientationDetectionModeEnum**	Specifies the mode of page orientation detection. This property is ODM_Normal by default.
ProhibitClockwiseRotation	**Boolean**	Disables clockwise page rotation when selecting the page orientation. This property is FALSE by default. **Note:** This property must not have the TRUE value if the **ProhibitCounterclockwiseRotation** and **ProhibitUpsidedownRotation** properties are set to TRUE.
ProhibitCounterclockwiseRotation	**Boolean**	Disables counterclockwise page rotation when selecting the page orientation. This property is FALSE by default. **Note:** This property must not have the TRUE value if the **ProhibitClockwiseRotation** and **ProhibitUpsidedownRotation** properties are set to TRUE.
ProhibitUpsidedownRotation	**Boolean**	Disables upside-down page rotation when selecting the page orientation. This property is FALSE by default. **Note:** This property must not have the TRUE value if the **ProhibitClockwiseRotation** and **ProhibitCounterclockwiseRotation** properties are set to TRUE.

Methods

Name	**Description**
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreateOrientationDetectionParams** method of the **Engine** object

Input parameter

This object is the input parameter of the following methods and properties:

- **DetectOrientation** method of the **FRPage** object
- **DetectOrientation** method of the **DocumentAnalyzer** object

See also

Tuning Analysis, Recognition, and Synthesis Parameters
PageAnalysisParams
Working with Properties

SynthesisParamsForDocument Object (ISynthesisParamsForDocument Interface)

This object is used for setting up the parameters of the document synthesis. It allows you to specify the fonts that will be used for reproducing different font types in the recognized text.

The **SynthesisParamsForDocument** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

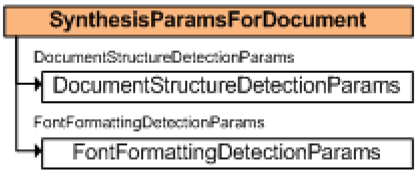
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DetectDocumentStructure	**Boolean**	Specifies whether document structure detection should be performed while document synthesis. This property is set to TRUE by default.
DetectFontFormatting	**Boolean**	Specifies whether font formatting detection should be performed while document synthesis. This property is set to TRUE by default.
DocumentStructureDetectionParams	**DocumentStructureDetectionParams**	Provides access to the parameters of document structure detection.
FontFormattingDetectionParams	**FontFormattingDetectionParams**	Provides access to the parameters of font formatting detection.
InsertEmptyParagraphsForBigInterlines	**Boolean**	If this property is set to TRUE, empty paragraphs are inserted to reproduce big line spacing of the original text. This property is set to FALSE by default.
PagePoolSize	**Long**	Specifies how many pages may be loaded by document synthesis simultaneously. This property allows you to decrease memory usage. We recommend to use the value in range from 32 to 64. The more the value, the more speed of processing. However, for processing big documents it is not recommended to use the highest values

		of this property, as this may lead to an "out of memory" error. The value less than 5 is ignored. By default the value of this property is 64.
RecognizedTextFontCount	**Long**, read-only	Stores the number of elements in the collection of fonts of the recognized text.
SaveRecognitionInfo	**Boolean**	If this property is set to TRUE, the information about words and character (quality, model, etc.) will be saved during export. If the **SaveCharacterRecognitionVariants** property or the **SaveWordRecognitionVariants** property of the **RecognizerParams** object is set to TRUE, the value of this property is ignored. This property is set to TRUE by default.

Methods

Name	Description
AddRecognizedTextFontName	Adds a font in the collection of fonts which are used in the recognized text.
CleanRecognizedTextFontNames	Cleans the collection of fonts which are used in the recognized text.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
GetRecognizedTextFontName	Returns the name of the font in the collection of fonts which are used in the recognized text.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects


SynthesisParamsForDocument
DocumentStructureDetectionParams
DocumentStructureDetectionParams
FontFormattingDetectionParams
FontFormattingDetectionParams

Output parameter

This object is the output parameter of the **CreateSynthesisParamsForDocument** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **Process**, **Synthesize**, **SynthesizePages** of the **FRDocument** object.
- **RecognizeImageFile**, **SynthesizePages**, **SynthesizePagesEx** of the **Engine** object.

See also

Tuning Analysis, Recognition, and Synthesis Parameters
SynthesisParamsForPage
Working with Properties

AddRecognizedTextFontName Method of the SynthesisParamsForDocument Object

This method adds a font to the collection of fonts of the recognized text.

Visual Basic Syntax

```
Method AddRecognizedTextFontName(
  Value As String
)
```

C++ Syntax

```
HRESULT AddRecognizedTextFontName(
  BSTR Value
);
```

Parameters

Value

[in] This parameter specifies the name of the font that should be added.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

SynthesisParamsForDocument

CleanRecognizedTextFontNames Method of the SynthesisParamsForDocument Object

This method cleans the collection of fonts which are used in the recognized text.

Visual Basic Syntax

```
Method CleanRecognizedTextFontNames()
```

C++ Syntax

```
HRESULT CleanRecognizedTextFontNames();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

SynthesisParamsForDocument

GetRecognizedTextFontName Method of the SynthesisParamsForDocument Object

This method returns the name of the font in the collection of fonts which are used in the recognized text.

Visual Basic Syntax

```
Method GetRecognizedTextFontName(
  FontNumber As Long
) As String
```

C++ Syntax

```
HRESULT GetRecognizedTextFontName(
  long  FontNumber
  BSTR* Result
);
```

Parameters

FontNumber

[in] This parameter specifies the index of the font in the internal collection of fonts used in the recognized text. Must be in a range from 0 to the value of the **ISynthesisParamsForDocucment::RecognizedTextFontCount** property -1.

Result

[out] A pointer to a string variable that receives the font name.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

SynthesisParamsForDocument

DocumentStructureDetectionParams Object (IDocumentStructureDetectionParams Interface)

This object is used for setting up the parameters of the document structure detection during document synthesis. This object is passed as a subobject of **SynthesisParamsForDocument** object to recognition and synthesis methods. By default, all the Boolean properties of this object are set to TRUE. You may turn off some of the properties, if your document does not contain any elements of this type (e.g. it does not have footnotes or table of contents) which may speed up processing.

The **DocumentStructureDetectionParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

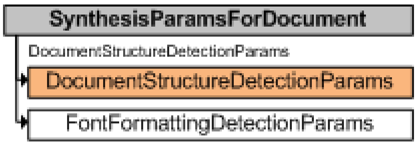
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
ClassifySeparators	**Boolean**	If this property is set to TRUE, additional properties of separators (such as their type, etc.) are detected during document synthesis. This property is set to TRUE by default.
DetectCaptions	**Boolean**	If this property is set to TRUE, the captions are detected during document synthesis. This property is set to TRUE by default.
DetectColumns	**Boolean**	If this property is set to TRUE, the columns are detected during document synthesis. This property is set to TRUE by default.
DetectFootnotes	**Boolean**	If this property is set to TRUE, the footnotes are detected during document synthesis. This property is set to TRUE by default.
DetectHeadlines	**Boolean**	If this property is set to TRUE, the headlines are detected during document synthesis. This property is set to TRUE by default.
DetectLists	**Boolean**	If this property is set to TRUE, the lists are detected during document document synthesis. This property is set to TRUE by default.
DetectOverflowingParagraphs	**Boolean**	If this property is set to TRUE, the overflowing paragraphs are detected during document synthesis. The overflowing paragraph is the one which starts from one page and ends on another page. If the property is set to FALSE, the program presumes that there are no overflowing paragraphs in the document. This property is set to TRUE by default.
DetectRunningTitles	**Boolean**	If this property is set to TRUE, the running titles are detected during document synthesis. This property is set to TRUE by default.
DetectTableOfContents	**Boolean**	If this property is set to TRUE, the table of contents is detected during document synthesis. This property is set to TRUE by default.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects


SynthesisParamsForDocument
DocumentStructureDetectionParams
DocumentStructureDetectionParams
FontFormattingDetectionParams

See also

Tuning Analysis, Recognition, and Synthesis Parameters
SynthesisParamsForDocument
Working with Properties

FontFormattingDetectionParams Object (IFontFormattingDetectionParams Interface)

This object is used for setting up the parameters of font formatting detection during document synthesis. This object is passed as a subobject of **SynthesisParamsForDocument** object to recognition and synthesis methods. By default, all the Boolean properties of this object are set to TRUE.

The **FontFormattingDetectionParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

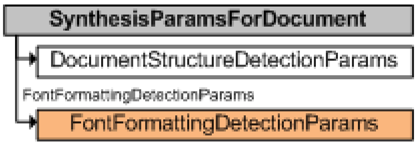
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DetectBold	**Boolean**	If this property is set to TRUE, the bold-face type is detected during synthesis. This property is set to TRUE by default.
DetectDropCaps	**Boolean**	If this property is set to TRUE, the drop caps are detected during synthesis. This property is set to TRUE by default.
DetectFontFamily	**Boolean**	If this property is set to TRUE, the font name is detected during synthesis. This property is set to TRUE by default.
DetectFontSize	**Boolean**	If this property is set to TRUE, the font size is detected during synthesis. This property is set to TRUE by default.
DetectItalic	**Boolean**	If this property is set to TRUE, the italic-face type is detected during synthesis. This property is set to TRUE by default.
DetectMonospace	**Boolean**	If this property is set to TRUE, the monospace typeface is detected during synthesis. This property is set to TRUE by default. If this property is FALSE, the **MonospaceDetectionMode** property is ignored.
DetectScaling	**Boolean**	If this property is set to TRUE, the scaling is detected during synthesis. This property is TRUE by default.
DetectSerifs	**Boolean**	If this property is set to TRUE, serif is detected during synthesis, i.e. if serif has been detected, serif typeface is selected to represent the recognized text. If this property is set to FALSE, serif is ignored. This means that the most suitable font (from both serif and sans serif typefaces) is selected to represent the recognized text, no matter whether the text is serif or sans serif. This property is set to TRUE by default.
DetectSmallCaps	**Boolean**	If this property is set to TRUE, the small capital letters are detected during synthesis. This property is set to TRUE by default.
DetectSpacing	**Boolean**	If this property is set to TRUE, the spacing is detected during synthesis. This property is TRUE by default.

DetectSubscriptsSuperscripts	**Boolean**	If this property is set to TRUE, the subscripts and superscripts are detected during synthesis. This property is TRUE by default.
DetectUnderlineStrikeout	**Boolean**	If this property is set to TRUE, the underline and strikeout are detected during synthesis. This property is set to TRUE by default.
MonospaceDetectionMode	**MonospaceDetectionModeEnum**	Specifies the mode of monospaced font detection. The property makes sense only if the **DetectMonospace** property is set to TRUE. The default mode is MDM_Auto.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects


SynthesisParamsForDocument
DocumentStructureDetectionParams
FontFormattingDetectionParams
FontFormattingDetectionParams

See also

Tuning Analysis, Recognition, and Synthesis Parameters
SynthesisParamsForDocument
Working with Properties

SynthesisParamsForPage Object (ISynthesisParamsForPage Interface)

This object is used for setting up the parameters of the page synthesis. Particularly, it allows you to specify the parameters of text and background color detection.

The **SynthesisParamsForPage** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
AllowGrayBackgroundColor	**Boolean**	If this property is set to TRUE, the gray color is detected for background. Otherwise, background will be detected as black or white. The value of this property is taken into account only if the **DetectBackgroundColor** property is set to TRUE. The default value of this property is TRUE.
AllowGrayTextColor	**Boolean**	If this property is set to TRUE, the gray color is detected for text. Otherwise, text will be detected as black or white. The value of this property is taken into account only if the **DetectTextColor** property is set to TRUE. The default value of this property is FALSE.
Application	**Engine**, read-only	Returns the **Engine** object.

CorrectDynamicRange	**Boolean**	If this property is TRUE, image colors will be corrected so that the background is white and the text is black, or vice versa, which improves image quality. Recognition, however, will slow down. We recommend using this property only if the **DetectBackgroundColor** and **DetectTextColor** properties are TRUE. This property is set to FALSE by default.
DetectBackgroundColor	**Boolean**	If this property is set to TRUE, the background color is detected during page synthesis. This property is set to FALSE by default.
DetectDocumentLinks	**Boolean**	If this property is set to TRUE, document references (e.g. cross-references) are detected during page synthesis. This property is set to TRUE by default.
DetectFontFormattingAtPageLevel	**Boolean**	If this property is set to TRUE, font parameters are detected at the stage of page synthesis. This property set to TRUE enables detection of subscripts, superscripts, italic-face type, small capital letters at the stage of page synthesis and allows you to set additional parameters using **FontFormattingDetectionParams** property. If this property is FALSE, the **FontFormattingDetectionParams** property is ignored. 📝**Note:** Normally, ABBYY FineReader Engine 10 detects font parameters at the stage of document synthesis. Therefore, if you set the value of this property to TRUE, you can then turn off the detection of font parameters during document synthesis. To do this, set the **ISynthesisParamsForDocument::DetectFontFormatting** property to FALSE. Detection of font parameters during page synthesis enables the program to speed up the subsequent document synthesis and decrease memory usage. However, the quality of font detection may deteriorate.
DetectHyperlinks	**Boolean**	If this property is set to TRUE, hyperlinks are detected during page synthesis. This property is set to TRUE by default.
DetectTextColor	**Boolean**	If this property is set to TRUE, the text color is detected during page synthesis. This property is set to FALSE by default.
FontFormattingDetectionParams	**FontFormattingDetectionParamsForPage**	Specifies additional parameters of font formatting detection at the stage of page synthesis: bold-face type detection, font name and font size detection. This property is used, only if the **DetectFontFormattingAtPageLevel** property is set to TRUE.
ParagraphExtractionMode	**ParagraphExtractionModeEnum**	Specifies the mode of paragraph extraction. The default mode is PEM_NormalExtraction.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.
SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Output parameter

This object is the output parameter of the **CreateSynthesisParamsForPage** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **AnalyzeAndRecognize**, **Recognize**, **RecognizeBlocks** of the **FRPage** object
- **AnalyzeAndRecognize**, **AnalyzeAndRecognizePages**, **Process**, **Recognize**, **RecognizePages** of the **FRDocument** object
- **AnalyzeAndRecognizePage**, **AnalyzeAndRecognizePages**, **RecognizeBlocks**, **RecognizeImageDocumentAsPlainText**, **RecognizePage**, **RecognizePages** of the **DocumentAnalyzer** object
- **RecognizePage**, **AnalyzeAndRecognizePage**, **RecognizeImageFile**, **RecognizeImageAsPlainText**, **RecognizeImageDocumentAsPlainText**, **RecognizePages**, **AnalyzeAndRecognizePages** of the **Engine** object

See also

Tuning Analysis, Recognition, and Synthesis Parameters
SynthesisParamsForDocument
Working with Properties

FontFormattingDetectionParamsForPage Object (IFontFormattingDetectionParamsForPage Interface)

This object specifies the parameters of font formatting detection at the stage of page synthesis. This object is a subobject of the **SynthesisParamsForPage** object and is used only if the **ISynthesisParamsForPage::DetectFontFormattingAtPageLevel** property is set to TRUE.

Normally, ABBYY FineReader Engine 10 detects font parameters at the stage of document synthesis. Detection of font parameters during page synthesis enables the program to speed up the subsequent document synthesis and decrease memory usage. However, the quality of font detection may deteriorate.

The **FontFormattingDetectionParamsForPage** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

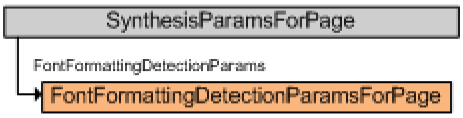
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DetectBold	**Boolean**	If this property is ser to TRUE, the program will detect bold-face type at the stage of page synthesis. This property is set to FALSE by default.
DetectFontFamily	**Boolean**	If this property is ser to TRUE, the program will detect font name at the stage of page synthesis. This property is set to FALSE by default.
DetectFontSize	**Boolean**	If this property is ser to TRUE, the program will detect font size at the stage of page synthesis. This property is set to FALSE by default.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.
SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Related objects


SynthesisParamsForPage
FontFormattingDetectionParams
FontFormattingDetectionParamsForPage

See also

Tuning Analysis, Recognition, and Synthesis Parameters
SynthesisParamsForPage
Working with Properties

HTMLExportParams Object (IHTMLExportParams Interface)

This object provides functionality for tuning parameters of recognized text export in HTML format by means of ABBYY FineReader Engine export functions. A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.

The **HTMLExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
CodePage	**CodePageEnum**	This property specifies the code page to which the recognized text is exported. The value of this property is taken into account only when the **EncodingType** property has value TET_Simple (exported text is not Unicode), and in this case the property must specify a valid code page (it cannot be CP_Null). **Note:** Firstly, you should set the correct code page, and then change the value of the **EncodingType** property to TET_Simple. By default this property is CP_Null.
EncodingType	**TextEncodingTypeEnum**	Specifies the encoding type of the output file in HTML format. This property is TET_Auto by default which means that encoding is selected automatically. **Note:** If you want to change the value of this property to TET_Simple, at first you should set the correct code page (the **CodePage** property).
HTMLFormatMode	**HTMLFormatModeEnum**	Specifies the version of HTML used for export. Export may be done in HTML 3.2 format for old browsers, HTML 4.0 format for newer browsers. The default value is HFM_Format40, which specifies the format supported only by newer browsers.

HTMLSynthesisMode	**HTMLSynthesisModeEnum**	Specifies a mode of synthesizing HTML code from the recognized text. There exist three modes of synthesis: retain paragraphs only, retain paragraphs and fonts, retain full logical structure of the document. The default value is HSM_FlexibleLayout, which means that the whole logical structure of the document is retained. If you set the value of this property to HSM_PlainText, the value of the **HTMLFormatMode** property is automatically set to HFM_Format32.
KeepLines	**Boolean**	Specifies if original lines in recognized text are retained during export. This property is FALSE by default.
KeepTextAndBackgroundColor	**Boolean**	Specifies if original colors of text and background are retained during export of the recognized text in HTML format. This property is TRUE by default.
PictureFormat	**ExportPictureFormatEnum**	Specifies the image format to be used during export to HTML; images are saved to separate files. This property can have one of the following values: EPF_Automatic, EPF_JpegColor, EPF_JpegGray, EPF_PngBlackWhite, EPF_PngColor, EPF_PngGray. The default value for this property is EPF_Automatic.
PictureJpegQuality	**Long**	Stores the value of the JPEG quality for color pictures saved in HTML format in percent. The default value for this property is 50%.
PictureResolution	**Long**	Stores the value of picture resolution in dpi, that is used for exporting pictures for HTML format. This property may be set to -1, which means that the original resolution must be preserved. The default value for it is 72 dpi.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting in HTML format. This property is RTM_WriteAsNative by default.
SplitDocumentToFiles	**HTMLDocumentSplittingModeEnum**	Specifies the mode of splitting output document into files. By default this property is HDSM_None.
WriteAuthor	**Boolean**	Specifies if the author of the document should be written into the output HTML file. The author of the document is defined in the **Author** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteKeywords	**Boolean**	Specifies if the keywords of the document should be written into the output HTML file. The keywords of the document is defined in the **Keywords** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WritePictures	**Boolean**	Specifies whether pictures must be saved along with the file in HTML format. If pictures are not written, references to them in HTML files are also omitted. The default value is TRUE.
WriteSubject	**Boolean**	Specifies if the subject of the document should be written into the output HTML file. The subject of the document is defined in the **Subject** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteTitle	**Boolean**	Specifies if the title of the document should be written into the output HTML file. The title of the

		document is defined in the **Title** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the **CreateHTMLExportParams** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object.
- **Export** method of the **FRPage** object.
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object.
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object.

See also

Tuning Export Parameters
Working with Properties

PPTExportParams Object (IPPTExportParams Interface)

This object provides functionality for tuning of parameters of recognized text export in PPTX format by means of ABBYY FineReader Engine export functions. A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.

The **PPTExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColorMode	**BackgroundColorModeEnum**	Specifies the mode of background color saving when exporting to PPTX format. Only background color of rectangular text and table blocks can be saved. This property is BCM_Color by default.
KeepLines	**Boolean**	Specifies whether original lines in recognized text are retained during export. The default value is FALSE.
KeepTextColor	**Boolean**	Specifies if original colors of text are retained during export of the recognized text to PPTX format. This property is TRUE by default.
PaperHeight	**Long**	Specifies paper height in twips (1/1440 of inch) for PPTX file. The value of this property should be in range from 1 to 56 inches. If the value of this property or **PaperWidth** property is outside the scope, the program will use the height of a standard slide which encloses

		the layout of exporting page. By default the value of this property is 0.
PaperWidth	**Long**	Specifies paper width in twips (1/1440 of inch) for PPTX file. The value of this property should be in range from 1 to 56 inches. If the value of this property or **PaperHeight** property is outside the scope, the program will use the width of a standard slide which encloses the layout of exporting page. By default the value of this property is 0.
PictureFormat	**ExportPictureFormatEnum**	Specifies the image format to be used during export to PPTX. This property can have one of the following values: EPF_Automatic, EPF_JpegColor, EPF_JpegGray, EPF_PngBlackWhite, EPF_PngColor, EPF_PngGray. The default value is EPF_Automatic.
PictureJpegQuality	**Long**	Stores the value in percentage points of the JPEG quality for color pictures saved in PPTX format. The default value is 50%.
PictureResolution	**Long**	Stores the value of picture resolution in dpi, which is used for exporting pictures to PPTX format. This property may be set to -1, which means that the original resolution must be preserved. The default value is 150 dpi.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting to PPTX format. This property is RTM_WriteAsNative by default.
WrapTextInBlock	**Boolean**	Specifies whether the text must fit into the original blocks when **KeepLines** is set to TRUE. The default value is FALSE. **Note:** Text in hieroglyphic languages which is arranged vertically is exported as if the **WrapTextInBlock** is set to FALSE, no matter what the value of the property is.
WriteAuthor	**Boolean**	Specifies if the author of the document should be written into the output PPTX file. This property is TRUE by default. The author of the document is defined in the **Author** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteKeywords	**Boolean**	Specifies if the keywords of the document should be written into the output PPTX file. This property is TRUE by default. The keywords of the document are defined in the **Keywords** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WritePictures	**Boolean**	Specifies whether pictures must be written in files in PPTX format. The default value is TRUE.
WriteSubject	**Boolean**	Specifies if the subject of the document should be written into the output PPTX file. This property is TRUE by default. The subject of the document is defined in the **Subject** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteTitle	**Boolean**	Specifies if the title of the document should be written into the output PPTX file. This property is TRUE by default. The title of the document is defined in the **Title** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the **CreatePPTExportParams** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object.
- **Export** method of the **FRPage** object.
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object.
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object.

See also

Tuning Export Parameters
Working with Properties

RTFExportParams Object (IRTFExportParams Interface)

This object provides functionality for tuning parameters of recognized text export in RTF/DOC/DOCX format by means of ABBYY FineReader Engine export methods. A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.

The **RTFExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColorMode	**BackgroundColorModeEnum**	Specifies the mode of background color saving when exporting to RTF/DOC/DOCX format. This property is BCM_Color by default.
ErrorBackgroundColor	**Long**	Stores the value of color used to highlight uncertainly recognized symbols' background in the text exported to RTF/DOC/DOCX format. This property is used only when the **HighlightErrorsWithBackgroundColor** property is TRUE. It stores the color with which the background of uncertainly recognized symbols is highlighted. By default this property is (0,255,0) in RGB format, which corresponds to green color. **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color green equals 65280.
ErrorTextColor	**Long**	Stores the value of color used to highlight uncertainly recognized symbols in the text exported to RTF/DOC/DOCX format. This property is used only when the **HighlightErrorsWithTextColor** property is TRUE. It stores the color with which the text of uncertainly recognized symbols is highlighted. By default this property is (0,255,0) in RGB format, which corresponds to green color. **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value)*, where *red*

		value is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color green equals 65280.
ForceFixedPageSize	**Boolean**	Specifies whether export result must fit to dimensions set by the **PaperWidth** and **PaperHeight** properties. This property is FALSE by default.
HighlightErrorsWithBackgroundColor	**Boolean**	Specifies if uncertainly recognized symbols are highlighted with background color when exported to RTF/DOC/DOCX format. The color with which to highlight the background of uncertainly recognized symbols is stored in the property **ErrorBackgroundColor**. This property is FALSE by default.
HighlightErrorsWithTextColor	**Boolean**	Specifies if uncertainly recognized symbols are highlighted with text color when exported to RTF/DOC/DOCX format. The color with which to highlight the text of uncertainly recognized symbols is stored in the property **ErrorTextColor.** This property is FALSE by default.
KeepLines	**Boolean**	Specifies if original lines in recognized text are retained during export in RTF/DOC/DOCX format. This property is FALSE by default.
KeepPages	**Boolean**	Specifies if original page arrangement in recognized text is retained during export to RTF/DOC/DOCX format. This property is TRUE by default.
KeepTextAndBackgroundColor	**Boolean**	This property is obsolete. Use the properties **KeepTextColor** and **BackgroundColorMode** instead.
KeepTextColor	**Boolean**	Specifies if original colors of text are retained during export of the recognized text to RTF/DOC/DOCX format. This property is TRUE by default.
PageOrientation	**RTFPageOrientationEnum**	Specifies page orientation during export in RTF/DOC/DOCX format. By default, the property is set to POM_Auto. The value of this property is ignored if the **PageSynthesisMode** property is set to PSM_RTFColumns. In this case, portrait orientation is used.
PageSynthesisMode	**RTFPageSynthesisModeEnum**	Specifies the mode of RTF/DOC/DOCX file synthesis from the recognized text when exporting to RTF/DOC/DOCX format. This property is PSM_RTFColumns by default.
PaperHeight	**Long**	Stores paper height in twips (1/1440 of inch). Default for this property is the height of A4 format page. See the table below.
PaperWidth	**Long**	Stores paper width in twips (1/1440 of inch). Default for this property is the width of A4 format page. See the table below.
PictureFormat	**ExportPictureFormatEnum**	Specifies the image format which will be used during export to an RTF/DOC/DOCX file with embedded pictures. This property can have one of the following values: EPF_JpegColor, EPF_JpegGray, EPF_PngBlackWhite, EPF_PngColor, EPF_PngGray, EPF_DontSave or EPF_Automatic. The default value for this property is

		EPF_Automatic.
PictureJpegQuality	**Long**	Stores the value of the JPEG quality for color pictures saved in RTF/DOC/DOCX format in percent. The default value for this property is 50%.
PictureResolution	**Long**	Stores the value of picture resolution in dpi. This property may be set to -1, which means that the original resolution must be preserved. The default value for it is 150 dpi.
RemoveSoftHyphens	**Boolean**	Tells ABBYY FineReader Engine to remove optional hyphens when exporting recognized text to RTF/DOC/DOCX format. If the **KeepLines** property is TRUE, optional hyphens are replaced with hyphens. By default this property is FALSE.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting to RTF/DOC/DOCX format. This property is RTM_WriteAsNative by default.
WriteAuthor	**Boolean**	Specifies if the author of the document should be written into the output RTF/DOC/DOCX file. The author of the document is defined in the **Author** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteKeywords	**Boolean**	Specifies if the keywords of the document should be written into the output RTF/DOC/DOCX file. The keywords of the document are defined in the **Keywords** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WritePictures	**Boolean**	Specifies if pictures are written in files in RTF/DOC/DOCX format. By default this property is TRUE.
WriteSubject	**Boolean**	Specifies if the subject of the document should be written into the output RTF/DOC/DOCX file. The subject of the document is defined in the **Subject** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteTitle	**Boolean**	Specifies if the title of the document should be written into the output RTF/DOC/DOCX file. The title of the document is defined in the **Title** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Paper size in different units of measurement

Paper size	in inch	in mm	in twips (1/1440 of inch)
A3	11,69 x 16,54	297 x 420	16838 x 23811
A4	8,27 x 11,69	210 x 297	11909 x 16834

A5	5,83 x 8,27	148 x 210	8391 x 11909
Legal	8,5 x 14	216 x 356	12240 x 20160
Letter	8,5 x 11	216 x 279	12240 x 15840
Executive	7,25 x 10,5	184 x 266	10440 x 15120

Output parameter

This object is the output parameter of the **CreateRTFExportParams** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object.
- **Export** method of the **FRPage** object.
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object.
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object.

See also

Tuning Export Parameters
Working with Properties

TextExportParams Object (ITextExportParams Interface)

This object provides functionality for tuning parameters of recognized text export in TXT or CSV format by means of ABBYY FineReader Engine export functions. To select the format of export, use the **ExportFormat** property. In CSV the following formatting applies:

- Original lines are retained
- Lines containing separator symbols are quoted(" ")
- Quotes inside other quotes are duplicated

A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.

The **TextExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
AppendEOF	**Boolean**	Specifies if the EOF symbol is inserted at the end of file. This property is FALSE by default.
AppendToEnd	**Boolean**	Specifies if exported text is appended at the end of file if it already exists. This property is FALSE by default.
Application	**Engine**, read-only	Returns the **Engine** object.
CodePage	**CodePageEnum**	This property specifies the code page to which the recognized text is exported. The value of this property is taken into account only when the **EncodingType** property has value TET_Simple (exported text is not Unicode). If this property does not specify any code page (CP_Null), the code page is selected automatically. By default this property is CP_Null.
EncodingType	**TextEncodingTypeEnum**	Specifies the encoding type of the output file in TXT or

		CSV format. This property is TET_Auto by default which means that encoding is selected automatically.
ExportFormat	**TXTExportFormatEnum**	Specifies the format of export: TXT, CSV with full layout retained, or CSV with text from tables only. By default, the value of the property is TEF_TXT, which means that export to TXT format is performed.
ExportParagraphsAsOneLine	**Boolean**	Specifies if each paragraph in the recognized text is exported as one line. This property is FALSE by default.
InsertEmptyLineBetweenParagraphs	**Boolean**	Specifies if an empty line should be inserted between paragraphs. This property is FALSE by default.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting in TXT format. This property is RTM_WriteAsNative by default.
TabSeparator	**String**	Stores the string with which the table separators are replaced in the exported text. By default the value of the table separator is "\t". This property is taken into account during export to CSV and TXT formats.
UsePageBreaks	**Boolean**	Specifies if page break symbols (0x12) will be inserted between pages in case multiple pages are exported into TXT or CSV format. This property is FALSE by default.

Methods

Name	**Description**
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the **CreateTextExportParams** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object
- **Export** method of the **FRPage** object
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object

See also

Tuning Export Parameters
Working with Properties

XLExportParams Object (IXLExportParams Interface)

This object provides functionality for tuning parameters of recognized text export in XLS/XLSX format. A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.

The **XLExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
BackgroundColorMode	**BackgroundColorModeEnum**	Specifies the mode of background color saving when exporting to XLSX format. Only table cells background color can be saved. The background color of the text outside tables is not saved regardless of the value of this property. This property is BCM_DontSave by default.
ConvertStringsToNumbers	**Boolean**	Specifies if numerical values in recognized text are exported to XLS/XLSX format as numbers rather than as strings. This property is TRUE by default.
KeepTextColor	**Boolean**	Specifies if original colors of text are retained during export of the recognized text to XLSX format. This property is FALSE by default.
PageOrientation	**RTFPageOrientationEnum**	Specifies page orientation during export in XLSX format. By default, the property is set to POM_Auto.
PaperSize	**XLSXPaperSizeEnum**	Specifies one of the standard paper sizes for XLSX file. By default the value of this property is XLPS_NotSpecified.
RemoveFormatting	**Boolean**	This property set to TRUE tells ABBYY FineReader Engine to remove formatting for the text exported in XLS format. This property is FALSE by default.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting to XLS format. This property is RTM_WriteAsNative by default.
TablesOnly	**Boolean**	In case this property is TRUE, recognized text from table blocks only is exported into XLS/XLSX format. The default for it is FALSE.
WriteAuthor	**Boolean**	Specifies if the author of the document should be written into the output XLS/XLSX file. This property is TRUE by default. The author of the document is defined in the **Author** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteKeywords	**Boolean**	Specifies if the keywords of the document should be written into the output XLS/XLSX file. This property is TRUE by default. The keywords of the document are defined in the **Keywords** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteSubject	**Boolean**	Specifies if the subject of the document should be written into the output XLS/XLSX file. This property is TRUE by default. The subject of the document is defined in the **Subject** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteTitle	**Boolean**	Specifies if the title of the document should be written into the output XLS/XLSX file. This property is TRUE by default. The title of the document is defined in the **Title** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
XLFileFormat	**XLFileFormatEnum**	This property determines how recognized text will be exported to XLS format. It may be set to MS Excel 5, MS Excel 8, or both. This property is XLFF_DoubleStream by default. The value of this property is ignored when exporting to XLSX format.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.
SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Output parameter

This object is the output parameter of the **CreateXLExportParams** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object
- **Export** method of the **FRPage** object
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object

See also

Tuning Export Parameters
Working with Properties

XMLExportParams Object (IXMLExportParams Interface)

This object provides functionality for tuning parameters of recognized text export in XML format. A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.

Note: If your license supports the "ASCII License Basic Modules" module, the default values of the **WriteCharAttributes** and **WriteNondeskewedCoordinates** differ from the values specified in the description of the properties (set to XCA_Ascii and TRUE, respectively).

You can find the XML scheme of an XML document in the FineReader10-schema-v1.xsd file, which can be found in the Inc folder (Start > Programs > ABBYY FineReader Engine 10 > Installation Folders > Include Files Folder).

The **XMLExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read–only	Returns the **Engine** object.
WriteCharacterRecognitionVariants	**Boolean**	Specifies if collections of variants of characters recognition are to be written in files in XML format. This property is not taken into account if the **WriteCharAttributes** property is XCA_None. This property is FALSE by default. Note that the collections may contain more than one element only if the **IRecognizerParams::SaveCharacterRecognitionVariants** property was set to TRUE during recognition. See also Using Voting API section.
WriteCharAttributes	**XMLCharAttributesEnum**	Specifies which character attributes are to be written in files in XML format. This property is XCA_None by default.
WriteCharFormatting	**Boolean**	Specifies if character formatting is to be written in files in XML

		format. This property is FALSE by default.
WriteNondeskewedCoordinates	**Boolean**	Specifies if character coordinates written in files in XML format are on a non-deskewed image plane. This property is FALSE by default.
WriteWordRecognitionVariants	**Boolean**	Specifies if collections of variants of words recognition are to be written in files in XML format. This property is FALSE by default. Note that the collections may contain more than one element only if the **IRecognizerParams::SaveWordRecognitionVariants** property was set to TRUE during recognition. See also Using Voting API section.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the **CreateXMLExportParams** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object
- **Export** method of the **FRPage** object
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object

See also

Tuning Export Parameters
Working with Properties

PDFExportParams Object (IPDFExportParams Interface)

This object provides functionality for tuning the parameters of export of recognized text into PDF (PDF/A) format by means of the ABBYY FineReader Engine export functions.

A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults.

The **PDFExportParams** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Colority	**PDFColorityModeEnum**	Specifies color settings of the resulting PDF (PDF/A) file. The default value for this property is PCM_KeepColority.
MRCMode	**PDFMRCModeEnum**	Specifies the mode of using Mixed Raster Content for output PDF (PDF/A) file. By default, the value of this property is MRC_Auto.

PDFAComplianceMode	**PDFAComplianceModeEnum**	Specifies the format of export: PDF, PDF/A-1a, or PDF/A-1b. By default, the value is PCM_None, which means that export to PDF should be performed.
Resolution	**Long**	Specifies the picture resolution in dpi. The default value for the property is 150 dpi.
ResolutionType	**PDFResolutionTypeEnum**	Defines how to use the value of the picture resolution specified in the **Resolution** property. It may be used: • as the absolute resolution (used for all pictures), • as the desired resolution (may be used only if the original resolution is above the desired), • or the value is ignored (and the original resolution is used). By default, the value of this property is PRT_Desired.
Scenario	**PDFExportScenarioEnum**	Specifies the scenario of export to PDF (PDF/A) format, which optimizes export for some parameters: quality, size of the file, or/and speed of export. The default value is PES_Balanced.
TextExportMode	**PDFExportModeEnum**	Specifies the mode of export of recognized text into PDF format. It may be: text and pictures only, text over the page image, text under the page image, page image only. This property is PEM_ImageOnText by default.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Output parameter

This object is the output parameter of the **CreatePDFExportParams** method of the **Engine** object.

See also

Tuning Export Parameters
Working with Properties

PDFExportParamsOld Object (IPDFExportParamsOld Interface)

This object is obsolete. We recommend you to use the **PDFExportParams** object to tune export to PDF format.

This object provides functionality for tuning the parameters of export of recognized text into PDF format by means of the ABBYY FineReader Engine export functions.

Note: The recognized text is exported into linearized PDF that are optimized for Web publishing.

A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults.

When you save texts that use a non-Latin codepage (say, Cyrillic, Greek, Czech, etc.), ABBYY FineReader uses the fonts provided by ParaType company (www.paratype.com/shop).

ABBYY FineReader Engine has some peculiarities of exporting hieroglyphic languages to PDF. See the Recognizing Hieroglyphic Languages section for details.

The **PDFExportParamsOld** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Note: The earliest version of the PDF file which matches the specified properties of the **PDFEncryptionInfo** object and the **IPDFExportParamsOld::WriteTaggedPDF** property is selected as the version of the PDF file.

- The earliest file version available is the version **1.3**.
- If at least one of the **PermissionFillFormFields**, **PermissionExtractTextAndGraphicsExt**, **PermissionAssembleDoc**, **PermissionPrintExt** properties of the **PDFEncryptionInfo** object or the **WriteTaggedPDF** property is TRUE, or the encryption key length exceeds 40 bits, the PDF file version will be **1.4**.
- If the **IPDFEncryptionInfo::UseAES** property is TRUE, the version will be **1.6**.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
EmbedFonts	**Boolean**	Specifies whether fonts should be embedded during export to PDF. The **FontMode** property specifies the fonts to be embedded. If the value of the FontMode property is FM_UseStandardFonts and Latin Code page is used, fonts are not embedded (the value of the EmbedFonts property is ignored). The default value for this property is TRUE.
EnableMRC	**Boolean**	Specifies whether tuning Mixed Raster Content parameters is enabled. By default, the value of this property is FALSE.
EncryptionInfo	**PDFEncryptionInfo**	Specifies encryption parameters of the PDF file. **Note:** The property returns a constant object. To change the encryption parameters, you must first receive an intermediate **PDFEncryptionInfo** object with the help of the **IEngine::CreatePDFEncryptionInfo** method, change the necessary parameters, and then assign this object to the property.
ExportMode	**PDFExportModeEnum**	Specifies the mode of export of recognized text into PDF format. It may be: text and pictures only, text over the page image, text under the page image, page image only. This property is PEM_ImageOnText by default.
FontMode	**FontModeEnum**	Specifies the mode of font usage for export of recognized text into PDF format. The standard fonts can be used or the fonts can be taken from the **Text** object, which represents the recognized text. This property is FM_UseFontsFromIText by default.
KeepTextAndBackgroundColor	**Boolean**	Specifies if the original colors of the text and background are retained during export of the recognized text into PDF format. This property is TRUE by default. It is only usable when the **ExportMode** property is PEM_TextWithPictures, otherwise the value of this property is ignored.
MRCParams	**PDFMRCParams**, read-only	Returns a reference to the **PDFMRCParams** object which specifies Mixed Raster Content parameters of the PDF file. The property is only usable when the **ExportMode** property is PEM_ImageOnText or PEM_ImageOnly and **EnableMRC** property is set to TRUE, otherwise the value of this property is ignored.
PaperHeight	**Long**	Stores paper height in twips (1/1440 of inch). Default for this property is the height of A4 format page. See the table "Paper size in different units of measurement" in the **RTFExportParams** object.
PaperWidth	**Long**	Stores paper width in twips (1/1440 of inch). Default for this property is the width of A4 format page. See the table "Paper size in different units of measurement" in the **RTFExportParams** object.
PDFVersion	**PDFVersionEnum**	Specifies the version of the PDF file. The version should not conflict with the specified export parameters (see the

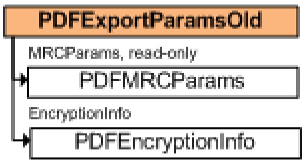
		note above for details). The default value for this property is PVN_Auto which specifies that the version is detected automatically.
PictureFormat	**ExportPictureFormatEnum**	Specifies the image format to be used during export to a PDF file with embedded pictures. This property can have one of the following values: EPF_Automatic, EPF_JpegColor, EPF_JpegGray, EPF_LZWColor, EPF_LZWGray, EPF_ZipColor, EPF_ZipGray, EPF_CCITT4, EPF_JBIG2. The default value for this property is EPF_Automatic.
PictureResolution	**Long**	Stores the value of picture resolution in dpi, which is used for exporting pictures into PDF format. This property may be set to -1, which means that the original resolution must be preserved. The default value for it is 150 dpi.
Quality	**Long**	Stores the value of the JPEG quality for color pictures saved in PDF format in percent. This value is ignored for black-and-white pictures. The default value for this property is 50%.
ReplaceUncertainWordsWithImage	**Boolean**	Specifies if uncertainly recognized words will be replaced with their images during export into PDF format. You may use this property when the **ExportMode** property is set to PEM_TextWithPictures or PEM_TextOnImage, otherwise its value is ignored. This property is FALSE by default.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting in PDF format. This property is RTM_WriteAsNative by default.
SetPageSizeByLayoutSize	**Boolean**	Specifies that the page size must be equal to the layout size during export of the recognized text into PDF format. If this property is FALSE the **PaperHeight** and **PaperWidth** properties define the page size. This property is TRUE by default.
WriteAuthor	**Boolean**	Specifies if the author of the document should be written into the output PDF file. This property is TRUE by default. The author of the document is defined in the **Author** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteCreator	**Boolean**	Specifies if the creator of the document should be written into the output PDF file. This property is TRUE by default. The author of the document is defined in the **Creator** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteKeywords	**Boolean**	Specifies if the keywords of the document should be written into the output PDF file. This property is TRUE by default. The keywords of the document are defined in the **Keywords** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteLinks	**Boolean**	Specifies that the hyperlinks must be retained during export of the recognized text into PDF format. This property is TRUE by default. If this property is FALSE the hyperlinks are exported as text.
WriteProducer	**Boolean**	Specifies if the producer of the document should be written into the output PDF file. This property is FALSE by default. The subject of the document is defined in the **Producer** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteSubject	**Boolean**	Specifies if the subject of the document should be written into the output PDF file. This property is TRUE by default. The subject of the document is defined in the **Subject** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

WriteTaggedPDF	**Boolean**	Specifies if the recognized text should be exported to tagged PDF. *Tagged PDF* is a particular use of structured PDF that allows page content to be extracted and used for various purposes such as reflow of text and graphics, conversion to file formats such as HTML and XML, and accessibility to the visually impaired. This property is FALSE by default.
WriteTitle	**Boolean**	Specifies if the title of the document should be written into the output PDF file. This property is TRUE by default. The title of the document is defined in the **Title** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreatePDFExportParamsOld** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object
- **Export** method of the **FRPage** object
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object
- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object

See also

Tuning Export Parameters
Working with Properties

PDFAExportParamsOld Object (IPDFAExportParamsOld Interface)

This object is obsolete. We recommend you to use the **PDFExportParams** object to tune export to PDF/A format.

This object provides functionality for tuning the parameters of export of recognized text into PDF/A format by means of the ABBYY FineReader Engine export functions. PDF/A is a constrained form of PDF version 1.4 intended to be suitable for long-term preservation of page-oriented documents.

A pointer to this object is passed into the export methods as an input parameter, and thus affects the results of export. All properties of a newly created object of this type are set to reasonable defaults.

When you save texts that use a non-Latin codepage (say, Cyrillic, Greek, Czech, etc.), ABBYY FineReader uses the fonts provided by ParaType company (www.paratype.com/shop).

The ABBYY FineReader Engine has some peculiarities of exporting hieroglyphic languages to PDF/A. See the Recognizing Hieroglyphic Languages section for details.

The **PDFAExportParamsOld** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

Note: ABBYY uses the Adobe Preflight utility (version 9.0) to examine the implementation of export to PDF/A for compliance with standard.

Properties

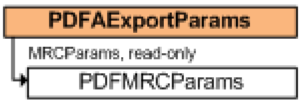
Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
EnableMRC	**Boolean**	Specifies whether tuning Mixed Raster Content parameters is enabled. By default, the value of this property is FALSE.
ExportMode	**PDFExportModeEnum**	Specifies the mode of export of recognized text to PDF/A format. It may be: text and pictures only, text over the page image, text under the page image, page image only. We recommend using the text under the page image mode as it is the most suitable for the PDF/A standard. This property is PEM_ImageOnText by default.
MRCParams	**PDFMRCParams**, read-only	Returns a reference to the **PDFMRCParams** object which specifies Mixed Raster Content parameters of the PDF/A file. The MRC property is only usable when the **ExportMode** property is PEM_ImageOnText or PEM_ImageOnly and **EnableMRC** property is set to TRUE, otherwise the value of this property is ignored.
PaperHeight	**Long**	Stores paper height in twips (1/1440 of inch). Default for this property is the height of A4 format page. See the table "Paper size in different units of measurement" in the **RTFExportParams** object.
PaperWidth	**Long**	Stores paper width in twips (1/1440 of inch). Default for this property is the width of A4 format page. See the table "Paper size in different units of measurement" in the **RTFExportParams** object.
PDFVersion	**PDFVersionEnum**	Specifies the version of the PDF/A file. The minimal version of the PDF/A file is 1.4. The default value for this property is PVN_Auto which specifies that the version is detected automatically.
PictureFormat	**ExportPictureFormatEnum**	Specifies the image format to be used during export to a PDF/A file with embedded pictures. This property can have one of the following values: EPF_Automatic, EPF_JpegColor, EPF_JpegGray, EPF_ZipColor, EPF_ZipGray, EPF_CCITT4, EPF_JBIG2. The default value for this property is EPF_Automatic.
PictureResolution	**Long**	Stores the value of picture resolution in dpi, which is used for exporting pictures to PDF/A format. This property may be set to -1, which means that the original resolution must be preserved. The default value for it is 150 dpi.
Quality	**Long**	Stores the value of the JPEG quality for color pictures saved in PDF/A format in percent. This value is ignored for black-and-white pictures. The default value for this property is 50%.
RunningTitleMode	**RunningTitleModeEnum**	Specifies the mode of running titles saving when exporting to PDF/A format. This property is RTM_WriteAsNative by default.
SetPageSizeByLayoutSize	**Boolean**	Specifies that the page size must be equal to the layout size during export of the recognized text to PDF/A format. If this property is FALSE the **PaperHeight** and **PaperWidth** properties define the page size. This property is TRUE by default.
WriteAuthor	**Boolean**	Specifies if the author of the document should be written into the output PDF/A file. This property is TRUE by default. The author of the document is defined in the **Author** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

WriteCreator	**Boolean**	Specifies if the creator of the document should be written into the output PDF/A file. This property is TRUE by default. The author of the document is defined in the **Creator** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteKeywords	**Boolean**	Specifies if the keywords of the document should be written into the output PDF/A file. This property is TRUE by default. The keywords of the document are defined in the **Keywords** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteLinks	**Boolean**	Specifies that the hyperlinks must be retained during export of the recognized text to PDF/A format. This property is TRUE by default. If this property is FALSE the hyperlinks are exported as text.
WritePDFA1A	**Boolean**	Specifies if the recognized text should be exported to PDF/A-1a format. The values of this property and the **WriteTaggedPDF** property depend on each other, if one property is set to TRUE, the other is automatically set to TRUE. This property is FALSE by default.
WriteProducer	**Boolean**	Specifies if the producer of the document should be written into the output PDF/A file. This property is FALSE by default. The subject of the document is defined in the **Producer** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteSubject	**Boolean**	Specifies if the subject of the document should be written into the output PDF/A file. This property is TRUE by default. The subject of the document is defined in the **Subject** property of the **DocumentContentInfo** subobject of the **FRDocument** object.
WriteTaggedPDF	**Boolean**	Specifies if the recognized text should be exported to tagged PDF. The values of this property and the **WritePDFA1A** property depend on each other, if one property is set to TRUE, the other is automatically set to TRUE. This property is FALSE by default.
WriteTitle	**Boolean**	Specifies if the title of the document should be written into the output PDF/A file. This property is TRUE by default. The title of the document is defined in the **Title** property of the **DocumentContentInfo** subobject of the **FRDocument** object.

Methods

Name	Description
CopyFrom	Initializes the properties of the current object with the values of similar properties of another object.
LoadFromFile	Restores the object contents from a file on disk.
LoadFromMemory	Restores the object contents from the global memory.
SaveToFile	Saves the object contents into a file on disk.
SaveToMemory	Saves the object contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreatePDFAExportParamsOld** method of the **Engine** object.

Input parameter

This object is passed as the input parameter to the following methods:

- **Export**, **ExportPages** methods of the **FRDocument** object
- **Export** method of the **FRPage** object
- **ExportPage**, **ExportPages**, **RecognizeImageFile** methods of the **Engine** object

- **ExportPages**, **ExportPagesEx** methods of the **Exporter** object

See also

Tuning Export Parameters
Working with Properties

PDFEncryptionInfo Object (IPDFEncryptionInfo Interface)

This object provides access to encryption parameters of the PDF file during export. These parameters are set in the **EncryptionInfo** property of **PDFExportParamsOld**. The **PDFEncryptionInfo** object allows you to do the following:

- set owner and user passwords;
- set the level of encryption;
- enable or disable the following:
  - adding or modifying text annotations and interactive form fields;
  - assembling the document: inserting, rotating, or deleting pages and creating navigation elements such as bookmarks or thumbnail images;
  - copying or otherwise extracting text and graphics from the document;
  - filling out forms (that is, filling out existing interactive form fields) and signing the document (which amounts to filling out existing signature fields, a type of interactive form field);
  - modifying the contents of the document;
  - printing the document.

Properties

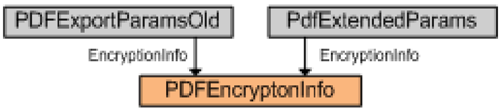
Name	**Type**	**Description**
Application	**Engine**, read–only	Returns the **Engine** object.
IsEncryptionRequested	**Boolean**	Specifies whether the PDF file must be encrypted. If this property is set to FALSE, the other properties will be ignored. This property is FALSE by default.
KeyLength	**PDFKeyLengthEnum**	Sets the length of the encryption key. This property is automatically set to PDFKL_128Bit if the **UseAES** property is TRUE. This property is PDFKL_40Bit by default.
OwnerPassword	**String**	Stores owner password. Opening the document with the correct owner password (assuming it is not the same as the user password) allows full (owner) access to the document. This unlimited access includes the ability to change the document's passwords and access permissions.
PermissionAddAnnotations	**Boolean**	Enables/disables adding or modifying text annotations and interactive form fields. The default value is FALSE.
PermissionAssembleDoc	**Boolean**	Enables/disables assembling the document: inserting, rotating, or deleting pages and creating navigation elements such as bookmarks or thumbnail images. The default value is FALSE.
PermissionExtractTextAndGraphics	**Boolean**	Enables/disables copying or otherwise extracting text and graphics from the document. The default value is FALSE.
PermissionExtractTextAndGraphicsExt	**Boolean**	Enables/disables extracting text and graphics (to make the accessible to users with disabilities or for other purposes). The default value is FALSE.

PermissionFillFormFields	**Boolean**	Enables/disables filling out forms (that is, filling out existing interactive form fields) and signing the document (which amounts to filling out existing signature fields, a type of interactive form field). The default value is FALSE.
PermissionModifyContent	**Boolean**	Enables/disables modifying the contents of the document. The default value is FALSE.
PermissionPrint	**Boolean**	Enables/disables printing the document. The default value is FALSE.
PermissionPrintExt	**Boolean**	Enables/disables printing to a representation from which a faithful digital copy of the PDF content could be generated. Disallowing such printing may result in degradation of output quality (a feature implemented as "Print As Image" in Acrobat). The default value is FALSE. The value of this property is ignored if the **PermissionPrint** property is set to FALSE.
UseAES	**Boolean**	Enables/disables a high (128–bit AES) encryption level, but Acrobat 6.0 (or earlier) users cannot open PDF documents with this encryption level. If the value of this property is TRUE, the value of the **KeyLength** property is automatically set to PDFKL_128Bit. This property is FALSE by default.
UserPassword	**String**	Stores the user password. Opening the document with the correct user password (or opening a document that does not have a user password) allows additional operations to be performed according to the user access permissions specified in the document's encryption dictionary. The default value is an empty string.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.

Related objects



Output parameter

This object is the output parameter of the **CreatePDFEncryptionInfo** method of the **Engine** object.

See also

Tuning Export Parameters
PDFExportParams
PdfExtendedParams
Working with Properties

PDFMRCParams Object (IPDFMRCParams Interface)

This object allows you to tune Mixed Raster Content (MRC) parameters for PDF (PDF/A) files. These parameters are set in the **MRCParams** property of the **PDFExportParamsOld** (**PDFAExportParamsOld**) object.

The MRC imaging model represents a document as three different layers: a foreground plane, a mask plane, and a background plane. Each layer is compressed separately using the best type of compression for that data type. The MRC technology for PDF (PDF/A) allows you to achieve significantly better file compression without visible degradation of document representation.

The **PDFMRCParams** object allows you to do the following:

- set MRC compression level;

- set the parameters of compression for background, color mask, and text mask;
- change background and text color.

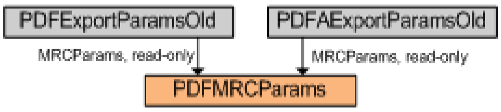
All the properties of the **PDFMRCParams** object are set to reasonable defaults. For more information about the default value of this or that property, see the description of the corresponding property.
Note: The value of the **CompressionLevel** property is set to PMRC_Custom automatically if you change the default value of any other property of the **PDFMRCParams** object.

Properties

Name	Type	Description
Application	**Engine**, read–only	Returns the **Engine** object.
BackgroundColor	**Long**	Specifies the background color. You can set the value of this property to –1. In this case, the background color will be detected automatically based on the original background. By default the background color is white or RGB(255,255,255). **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value*), where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color white equals 16777215.
BackgroundDownSampling	**Long**	Specifies the down sampling rate of the background. Only a positive rate makes sense. The default value is 2.
BackgroundFormat	**ExportPictureFormatEnum**	Specifies the background format. Only the EPF_JpegColor, EPF_JpegGray, EPF_J2KColor, and EPF_J2KGray values make sense during export to PDF, and only the EPF_JpegColor and EPF_JpegGray values make sense during export to PDF/A. The default value is EPF_J2KColor.
BackgroundQuality	**Long**	Stores the value of JPEG compression for background color in percentage points. By default, this property is set to 50%.
ColorMaskDownSampling	**Long**	Specifies the down sampling rate of the color mask. Only a positive rate makes sense. The default value is 2.
ColorMaskFormat	**ExportPictureFormatEnum**	Specifies the color mask format. Only the EPF_JpegColor, EPF_JpegGray, EPF_ZipColor, EPF_LZWColor, EPF_J2KColor, EPF_J2KGray values make sense during export to PDF, and only the EPF_JpegColor, EPF_JpegGray, EPF_ZipColor values make sense during export to PDF/A. The default value is EPF_ZipColor.
ColorMaskQuality	**Long**	Specifies the color mask quality in percentage points. The bigger the value, the better the quality. By default, this property is set to 30%.
CompressionLevel	**PDFMRCCompressionLevelEnum**	Stores the MRC compression level. The default value of this property is PMRC_AvgCompression.
KeepBackground	**Boolean**	Specifies if the original background is retained during export to a PDF (PDF/A) file with Mixed Raster Content. This property is TRUE by default.
MonochromeText	**Boolean**	Specifies if the recognized text is monochrome. If you set the value of this property to TRUE, you can specify the text color in the **TextColor** property. The default value of the **MonochromeText** property is FALSE.
PicturesInBackground	**Boolean**	Specifies whether pictures should be considered as parts of the background. If this property is set to TRUE, the background compression options are used for the pictures. The default value of this property is FALSE.

TextColor	**Long**	Specifies the text color in monochrome mode. This property is used only when the **MonochromeText** property is set to TRUE. By default, the text color is black or RGB(0,0,0). **Note:** The Long value is calculated from the RGB triplet using the formula: (*red value*) + (256 x *green value*) + (65536 x *blue value)*, where *red value* is the first triplet component, *green value* is the second triplet component, *blue value* is the third triplet component. Hence the Long value of the color black equals 0.
TextMaskDownSampling	**Long**	Specifies the down sampling rate of the text mask. Only a positive rate makes sense. The default value is 1.
TextMaskFormat	**ExportPictureFormatEnum**	Specifies the text mask compression algorithm. Only the EPF_CCITT4 and EPF_JBIG2 values make sense. The default value is EPF_JBIG2.
TextMaskQuality	**Long**	Specifies the text mask quality in percentage points. The bigger the value, the better the quality. By default, this property is set to 50%.

Related objects



See also

Tuning Export Parameters
PDFExportParamsOld
PDFAExportParamsOld
Working with Properties

License-Related Objects

This section contains descriptions of the following license-related objects:

- **License**
- **LicenseCollection**

The license-related objects hierarchy

For more information about the hierarchy of the ABBYY FineReader Engine objects, please see the **Object Diagram**.

License Object (ILicense Interface)

This object stores information about the current license.

Properties

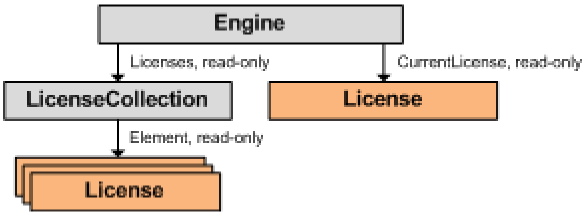
Name	Type	Description
AllowedCoresCount	**Long**, read-only	Returns the number of CPU cores that can be used simultaneously. If the value of this property is 0, the number of CPU cores is unlimited.
Application	**Engine**, read-only	Returns the **Engine** object.
AvailableEngineModules	**Long**, read-only	Describes the set of the ABBYY FineReader Engine modules available in the license as a bitwise OR combination of the **AEM_** prefixed flags.
AvailableExportFormats	**Long**, read-only	Describes the set of the export formats available in the license as a bitwise OR combination of the **AEF_** prefixed flags.

AvailableLanguageSets	**Long**, read-only	Describes the set of the language sets available in the license as a bitwise OR combination of the **ALS_** prefixed flags.
AvailableTextTypes	**Long**, read-only	Describes the set of the text types available in the license as a bitwise OR combination of the **ATT_** prefixed flags.
AvailableVisualComponents	**Long**, read-only	Describes the set of the ABBYY FineReader Engine Visual Components modules available in the license as a bitwise OR combination of the **AVC_** prefixed flags.
MinimumCoresCountPerInstance	**Long**, read-only	Returns the minimum number of CPU cores, which is allocated by ABBYY FineReader Engine at initialization.
SerialNumber	**String**, read-only	Returns the serial number of the license.
Volume	**Long**, read-only	Returns the total number of pages/characters which can be processed during a period if the license has such a limitation. See also **VolumeRefreshingPeriod** property.
VolumeRefreshingPeriod	**VolumeRefreshingPeriodEnum**, read-only	Returns information about the limitation period if the license limits the number of processed pages/characters during this period. See also **VolumeRemaining**, **Volume** properties.
VolumeRemaining	**Long**, read-only	Returns the remaining number of pages/characters which can be processed till the end of the current period if the license has such a limitation. When this property value reaches 0, analysis, recognition and export operations will not be possible. See also **VolumeRefreshingPeriod** property.

Methods

Name	Description
ExpirationDate	Returns the flag indicating whether the license has an absolute or relative time limitation as well as the date at which the license will stop working.

Related objects



Output parameter

This object is the output parameter of the **Item** and **FindLicense** methods of the **LicenseCollection** object.

Input parameter

This object is the input parameter of the **SetCurrentLicense** method of the **Engine** object.

See also

LicenseCollection
Working with Properties

Volume Property of the License Object

This property provides access to the total number of pages/characters which can be processed during a period if the license has such a limitation. The property uses as an input parameter the type of units (pages, characters) used by the ABBYY FineReader Engine license to limit the number of operations. The period is specified by the **ILicense::VolumeRefreshingPeriod** property.

Visual Basic Syntax

```
Property Volume(
   counterType As LicenseCounterTypeEnum
 ) As Long
   read-only
```

C++ Syntax

```
HRESULT get_Volume(
    LicenseCounterTypeEnum counterType,
    long*                  result
 );
```

Parameters

counterType

[in] This variable specifies the type of units used by the ABBYY FineReader Engine license to limit the number of operations during the period. See the description of the **LicenseCounterTypeEnum** constants.

result

[out, retval] Returns the total number of the specified limitation units which can be processed during the period.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

License
Working with Properties

VolumeRefreshingPeriod Property of the License Object

This property provides access to the limitation period if the license limits the number of processed pages/characters during this period. The property uses as an input parameter the type of units (pages, characters) used by the ABBYY FineReader Engine license to limit the number of operations.

Visual Basic Syntax

```
Property VolumeRefreshingPeriod(
   counterType As LicenseCounterTypeEnum
 ) As VolumeRefreshingPeriodEnum
   read-only
```

C++ Syntax

```
HRESULT get_VolumeRefreshingPeriod(
    LicenseCounterTypeEnum      counterType,
    VolumeRefreshingPeriodEnum* result
 );
```

Parameters

counterType

[in] This variable specifies the type of units used by the ABBYY FineReader Engine license to limit the number of operations during the period. See the description of the **LicenseCounterTypeEnum** constants.

result

[out, retval] Returns the license limitation period for the specified limitation units as the **VolumeRefreshingPeriodEnum** constant.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

License
Working with Properties

VolumeRemaining Property of the License Object

This property provides access to the remaining number of pages/characters which can be processed till the end of the current period if the license has such a limitation. When this property value reaches 0, analysis, recognition and export operations will not be possible. The property uses as an input parameter the type of units (pages, characters) used by the ABBYY FineReader Engine license to limit the number of operations. The period is specified by the **ILicense::VolumeRefreshingPeriod** property.

Visual Basic Syntax

```
Property VolumeRemaining(
   counterType As LicenseCounterTypeEnum
 ) As Long
   read-only
```

C++ Syntax

```
HRESULT get_VolumeRemaining(
    LicenseCounterTypeEnum counterType,
    long*                  result
 );
```

Parameters

counterType

[in] This variable specifies the type of units used by the ABBYY FineReader Engine license to limit the number of operations during the period. See the description of the **LicenseCounterTypeEnum** constants.

result

[out, retval] Returns the remaining number of the specified limitation units which can be processed till the end of the current period.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

License
Working with Properties

ExpirationDate Method of the License Object

This method returns the date at which the license will stop working if the license has an absolute or relative time limitation.

Visual Basic Syntax

```
Method ExpirationDate(
  year     As Long,
  month    As Long,
  day      As Long
) As Boolean
```

C++ Syntax

```
HRESULT ExpirationDate(
  long*         year,
  long*         month,
  long*         day,
  VARIANT_BOOL* hasTimeLimitation
);
```

Parameters

year

[out] A pointer to the **long** variable that receives the year of the expiration or 0 if no time limitation is used.

month

[out] A pointer to the **long** variable that receives the month of the expiration or 0 if no time limitation is used.

day

[out] A pointer to the **long** variable that receives the day of the expiration or 0 if no time limitation is used.

hasTimeLimitation

[out] A pointer to the **bool** variable that receives the flag indicating whether a time limitation is used.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

License
IEngine::SetCurrentLicense

LicenseCollection Object (ILicenseCollection Interface)

This object is a collection of available (activated) licenses. The collection is accessible via the **Engine** object.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

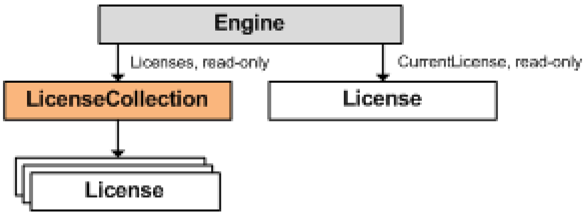
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**License**, read-only	Provides access to a single element of the collection.

Methods

Name	Description
FindLicense	Provides access to the license by its serial number.
Item	Provides access to a single element of the collection.

Related objects



See also

License
IEngine::SetCurrentLicense
Working with Properties

FindLicense Method of the LicenseCollection Object

This method provides access to the license by its serial number.

Visual Basic Syntax

```
Method FindLicense(
  serialNumber As String
) As License
```

C++ Syntax

```
HRESULT FindLicense(
  BSTR       serialNumber,
  ILicense** result
);
```

Parameters

serialNumber

[in] This parameter specifies the serial number of the license.

result

[out, retval] A pointer to the **ILicense*** pointer variable that receives the interface pointer of the **License** object. *result* is guaranteed to be non-NULL after successful method call.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

LicenseCollection

Supplementary Objects

This group of auxiliary objects. These objects are collections of different types and other objects which are used as input parameters and return values in ABBYY FineReader Engine methods.

This section contains descriptions of the following supplementary objects and interfaces:

- **StringsCollection**
- **LongsCollection**
- **DocumentInfo**
- **Region**
- **FRRectangle**
- **IRecognizedPages**

StringsCollection Objects (IStringsCollection Interface)

This object represents a collection of strings. It serves as a storage to pass various sets of parameters into those ABBYY FineReader Engine functions that require them. It may also be return value of ABBYY FineReader Engine methods.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**String**	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Output parameter

This object is the output parameter of the following methods:

- **CreateStringsCollection** and **PrepareImage** method of the **Engine** object
- **ExportPages**, **ExportPagesEx** of the **Exporter** object
- **Scan** method and **ScanSources** property of the **ScanManager** object

Input parameter

This object is the input parameter of the following methods:

- **AddWordsToCacheDictionary** method of the **DocumentAnalyzer** object
- **AddWords**, **DeleteWords** methods of the **Dictionary** object
- **CreateCompoundTextLanguage** method of the **LanguageDatabase** object
- **GetAllFootnoteTargets** method of the **DocumentStructure** object
- **MergePatterns** method of the **Engine** object

See also

Working with Properties

LongsCollection Object (ILongsCollection Interface)

This object represents a collection of **Long** type variables. It serves as a storage to pass various sets of parameters into those ABBYY FineReader Engine functions that require them. It may also be return value of ABBYY FineReader Engine methods.

⚠**Important**! The indexing of ABBYY FineReader Engine collections starts with 0.

Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Count	**Long**, read-only	Stores the number of elements in the collection.
Element	**Long**	Provides access to a single element of the collection.

Methods

Name	Description
Add	Adds a new element at the end of the collection.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
Insert	Inserts a new element into the specified position in the collection.
Item	Provides access to a single element of the collection.
Remove	Removes an element from the collection.
RemoveAll	Removes all the elements from the collection.

Output parameter

This collection is the output parameter of the following methods and properties:

- **CreateLongsCollection** method of the **Engine** object
- **PossibleTextTypes** property of the **RecognizerParams** object
- **PageIds** property of the **IRecognizedPages** interface

Input parameter

This collection is the input parameter of the following methods and properties:

- **AddImageFile**, **AddImageFileWithPassword**, **AddImageFileWithPasswordCallback**, **AnalyzeAndRecognizePages**, **AnalyzePages**, **ExportPages**, **RecognizePages**, **SynthesizePages** methods of the **FRDocument** object
- **Renumber** method of the **FRPages** object
- **RecognizeBlocks** of the **FRPage** object

- **AddWords** method of the **Dictionary** object
- **InitializeGrid** method of the **TableBlock** object
- **PossibleTextTypes** property of the **RecognizerParams** object

See also

Working with Properties

Element Property

This property provides access to a single element of ABBYY FineReader Engine collection. Each ABBYY FineReader Engine collection uses this property.

Visual Basic Syntax

```
Property Element(
  index As Long,
) As ObjectType
```

C++ Syntax

```
HRESULT Element(
  long           index,
  InterfaceType** pVal
);
```

Parameters

index

[in] This variable contains the index of the element that is accessed via this method. It must be in the range from 0 to the *Number of elements - 1*, where the number of elements may be received from the **Count** property of the same collection.

ObjectType

[out] The type of objects in collection. For example, for the **LayoutsCollection** collection this type is **Layout**.

pVal

[out] A variable of type **InterfaceType*** that receives a pointer to the interface of the collection element. *pVal* must not be NULL. **pVal* is guaranteed to be non-NULL after a successful method call. **InterfaceType** is the type of the interface of the objects forming the collection.

Remark

The following objects provide this property:

- Image-related objects
  - **ImageDocumentsCollection**
  - **TrainingImagesCollection**
- Layout and blocks:
  - **LayoutBlocks**
  - **LayoutsCollection**
  - **CheckmarkGroup**
  - **SeparatorGroup**
  - **TableCells**
  - **TableSeparators**
  - **BarcodeText**

- Language-related objects
  - **BaseLanguages**
  - **PredefinedLanguages**
  - **FuzzyStringsCollection**
  - **DictionaryDescriptions**
- Text-related objects
  - **Paragraphs**
  - **ParagraphLines**
  - **CharacterRecognitionVariants**
  - **WordRecognitionVariants**
  - **Words**
  - **TabPositions**
- Document-related objects
  - **FRPages**
  - **Captions**
  - **FootnoteSeriesArray**
  - **List**
  - **PageElements**
  - **PageSections**
  - **PageStreams**
  - **RunningTitleSeriesArray**
- Supplementary objects
  - **StringsCollection**
  - **LongsCollection**
- **LicenseCollection**

See also

Item
Working with Properties

Add Method

This method adds a new element at the end of the collection.

Visual Basic Syntax

```
Method Add(
  item As <ElementType>
)
```

C++ Syntax

```
HRESULT Add(
  <ElementType> item
);
```

Parameters

item

[in] This parameter contains the newly added element. Its type depends on the type of collection and is described in the following table:

Collection type	Element type (Visual Basic/C++)
BarcodeText	BarcodeSymbol/IBarcodeSymbol*
BaseLanguages	BaseLanguage/IBaseLanguage*
DocumentInformationDictionary	DocumentInformationDictionaryItem/IDocumentInformationDictionaryItem*
FuzzyStringsCollection	FuzzyString/IFuzzyString*
ImageDocumentsCollection	ImageDocument/IImageDocument*
LayoutBlocks *	Block/IBlock*
LayoutsCollection	Layout/ILayout*
LongsCollection	Long/long
StringsCollection	String/BSTR
TabPositions	TabPosition/ITabPosition*
TrainingImagesCollection	TrainingImage/ITrainingImage*

* — The method cannot be used for the **LayoutBlocks** object received using the **ILayout::Blocks** or **ILayout::BlackSeparators** property. To add a block into the collection, use the **AddBlock** method of the corresponding **Layout** object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Insert Method of Collection Objects

This method inserts a new element at the specified position in the collection.

Visual Basic Syntax

```
Method Insert(
  item  As <CollectionType>,
  index As Long
)
```

C++ Syntax

```
HRESULT Insert(
  <CollectionType> item,
  long             index
);
```

Parameters

item

[in] This parameter contains the newly inserted element. Its type depends on the type of collection and is described in the following table:

Collection type	Element type (Visual basic/C++)
DocumentInformationDictionary	DocumentInformationDictionaryItem/IDocumentInformationDictionaryItem*
FuzzyStringsCollection	FuzzyString/IFuzzyString*
ImageDocumentsCollection	ImageDocument/IImageDocument*
LayoutBlocks *	Block/IBlock*

LayoutsCollection	Layout/ILayout*
LongsCollection	Long/long
StringsCollection	String/BSTR
TrainingImagesCollection	TrainingImage/ITrainingImage*

* — The method cannot be used for the **LayoutBlocks** object received using the **ILayout::Blocks** property. To insert a block into the collection, use the **InsertBlock** method of the corresponding **Layout** object.

index

[in] This parameter specifies the index of the newly inserted element. If the element is inserted in place of the existing element, the elements of the collection are shifted to the right. The element may also be inserted at the end of collection, in which case the value of this parameter must be equal to the value of the **Count** property.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Add

Item Method

This method provides access to a single element of ABBYY FineReader Engine collection. Each ABBYY FineReader Engine collection uses this method.

Visual Basic Syntax

```
Method Item(
  index As Long,
) As ObjectType
```

C++ Syntax

```
HRESULT Item(
  long index,
  InterfaceType** pVal
);
```

Parameters

index

[in] This variable contains the index of the element that is accessed via this method. It must be in the range from 0 to the *Number of elements - 1*, where the number of elements may be received from the **Count** property of the same collection.

ObjectType

[out, retval] The type of objects in collection. For example, for the **LayoutsCollection** collection this type is **Layout**.

pVal

[out, retval] A variable of type **InterfaceType*** that receives a pointer to the interface of the collection element. *pVal* must not be NULL. **pVal* is guaranteed to be non-NULL after a successful method call. **InterfaceType** is the type of the interface of the objects forming the collection.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

The following objects provide this method:

- Layout-related objects
    - **LayoutBlocks**
    - **LayoutsCollection**

  - **BarcodeText**
  - **CheckmarkGroup**
  - **SeparatorGroup**
- Image-related objects
  - **ImageDocumentsCollection**
  - **TrainingImagesCollection**
- Language-related objects
  - **BaseLanguages**
  - **PredefinedLanguages**
  - **FuzzyStringsCollection**
- Text-related objects
  - **Paragraphs**
  - **ParagraphLines**
  - **Words**
  - **WordRecognitionVariants**
  - **CharacterRecognitionVariants**
  - **TabPositions**
- Document-related objects
  - **FRPages**
  - **DocumentInformationDictionary**
  - **PageSections**
  - **PageStreams**
  - **PageElements**
  - **Captions**
  - **FootnoteSeriesArray**
  - **RunningTitleSeriesArray**
  - **List**
- Supplementary objects
  - **LongsCollection**
  - **StringsCollection**
- **LicenseCollection**

See also

Element

See samples: RecognizedTextProcessing, CustomLanguage

Remove Method

This method is specific to collection objects. It removes an element from collection by its index.

Visual Basic Syntax

```
Method Remove(
  index As Long
)
```

C++ Syntax

```
HRESULT Remove(
  long index
);
```

Parameters

index

[in] This variable contains index of the collection element. It should be in a range from 0 to the value of the **Count** property of this collection minus 1.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions

Remark

The following objects have this method:

- **BaseLanguages**
- **CheckmarkGroup**
- **DocumentInformationDictionary**
- **FRPages**
- **FuzzyStringsCollection**
- **ImageDocumentsCollection**
- **LayoutBlocks**
- **LayoutsCollection**
- **LongsCollection**
- **StringsCollection**
- **TabPositions**
- **TrainingImagesCollection**

See also

RemoveAll

RemoveAll Method

This method is specific to ABBYY FineReader Engine collection objects. It removes all the elements from collection and empties it.

Visual Basic Syntax

```
Method RemoveAll()
```

C++ Syntax

```
HRESULT RemoveAll();
```

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remark

The following objects provide this method:

- **BarcodeText**
- **BaseLanguages**
- **CheckmarkGroup**
- **DocumentInformationDictionary**
- **FuzzyStringsCollection**
- **ImageDocumentsCollection**
- **LayoutBlocks**
- **LayoutsCollection**
- **List**
- **LongsCollection**
- **PageSections**
- **StringsCollection**
- **TabPositions**
- **TrainingImagesCollection**

See also

Remove

See sample: CustomLanguage

DocumentInfo Object (IDocumentInfo Interface)

This object stores service information about document. The object may be used in two different scenarios.

The first one uses the **DocumentInfo** object for OCR. You should save it when preparing the image and then pass it to the corresponding functions for use during analysis and recognition. If service information need not be used or files other than PDF are being opened, pass 0 to the corresponding function parameter.

The second scenario use document information during document synthesis and export. In this case, the **DocumentInfo** object is passed as a parameter to the **SynthesizePages**, **SynthesizePagesEx** methods of the **Engine** object and then is used during export. This allows to use during export all the information about document which was received during synthesis.

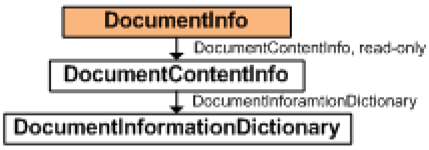
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
DocumentContentInfo	**DocumentContentInfo**, read-only	Returns a reference to the **DocumentContentInfo** object, which contains information about the author, keywords, subject, and title of the document and stores the document information dictionary.

Methods

Name	Description
Close	Releases all the resources that were used by the **DocumentInfo** object.

Related objects


DocumentInfo
DocumentContentInfo, read-only
DocumentContentInfo
DocumentInforamtionDictionary
DocumentInformationDictionary

Output parameter

This object is the output parameter of the **CreateDocumentInfo** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **PrepareAndOpenImage**, **PrepareImage**, **AnalyzeAndRecognizePage**, **AnalyzeAndRecognizePages**, **AnalyzePage**, **AnalyzePages**, **ExportPages**, **RecognizeImageDocumentAsPlainText**, **RecognizePage**, **RecognizePages**, **SynthesizePages**, **SynthesizePagesEx** of the **Engine** object.

- **AnalyzeAndRecognizePage**, **AnalyzeAndRecognizePages**, **AnalyzePage**, **AnalyzePages**, **AnalyzeRegion**, **AnalyzeTable**, **ExtractBarcodes**, **RecognizeBlocks**, **RecognizeImageDocumentAsPlainText**, **RecognizePage**, **RecognizePages** of the **DocumentAnalyzer** object.

- **ExportPages**, **ExportPagesEx** of the **Exporter** object.

See also

Engine
Working with Properties

Region Object (IRegion Interface)

This is a supplementary object. It is designed to store the information about the region of an ABBYY FineReader Engine block.

A *region* is a set of rectangles positioned one under another in such a way that the top line of the lower rectangle is the bottom line of the upper one (so that the rectangles do not overlap). Some examples of correct and incorrect ABBYY FineReader Engine regions are shown on the following figure:

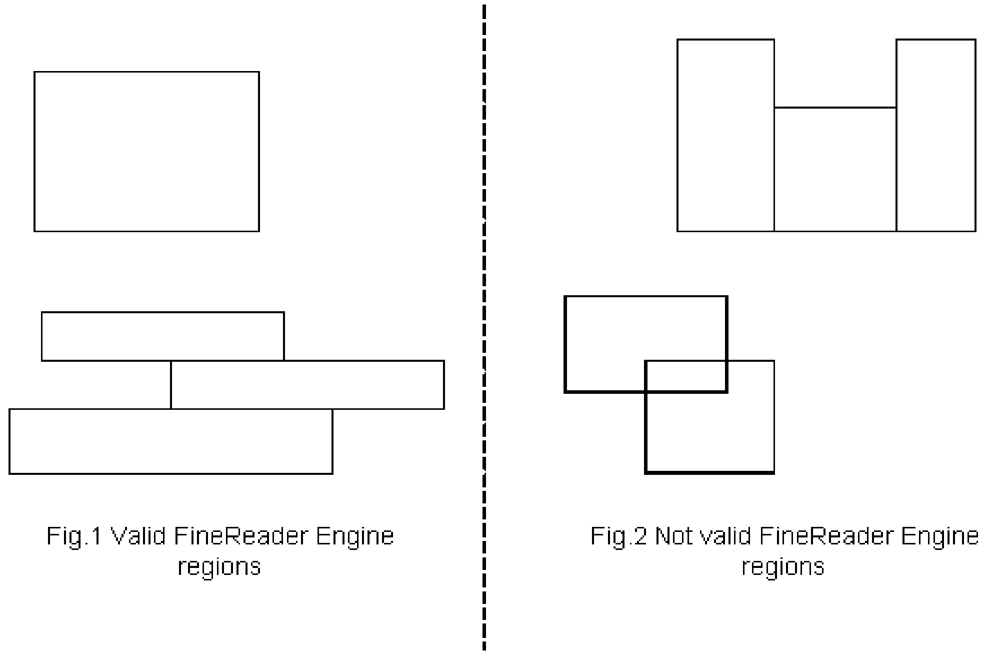


Fig.1 Valid FineReader Engine regions

Fig.2 Not valid FineReader Engine regions

An empty **Region** object may be created by calling the **IEngine::CreateRegion** method, and then rectangles may be added to it one-by-one by calling the **IRegion::AddRect** method. We recommend you to add rectangles in top to bottom order, because the **Region** object is optimized for it, and this is the fastest way to add rectangles to it.

The **Region** object is a persistent object. This means that it is able to write its current state, indicated by the values of its properties, to persistent storage: an area in the global memory or a disk file. Later, the object can be re-created by reading the object's state from persistent storage. The following methods provide persistence of the object: **SaveToFile**, **LoadFromFile**, **SaveToMemory**, and **LoadFromMemory**.

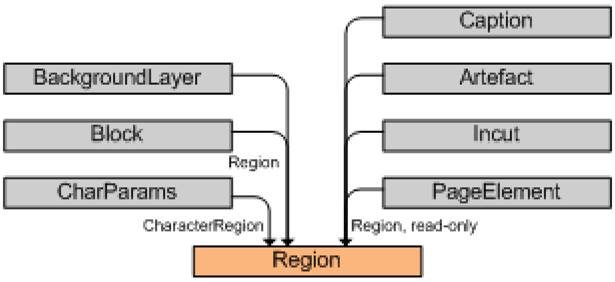
Properties

Name	Type	Description
Application	**Engine**, read-only	Returns the **Engine** object.
Bottom	**Long**, read-only	Returns the coordinate of the bottom border of the specified rectangle.
Count	**Long**, read-only	Stores the number of rectangles in the region.
Left	**Long**, read-only	Returns the coordinate of the left border of the specified rectangle.
Right	**Long**, read-only	Returns the coordinate of the right border of the specified rectangle.
Top	**Long**, read-only	Returns the coordinate of the top border of the specified rectangle.

Methods

Name	Description
AddRect	Adds a new rectangle into the region.
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
LoadFromFile	Restores the object's contents from a file on disk.
LoadFromMemory	Restores the object's contents from the global memory.
MakeEmpty	Removes all the rectangles from the region.
SaveToFile	Saves the object's contents into a file on disk.
SaveToMemory	Saves the object's contents into the global memory.

Related objects



Output parameter

This object is the output parameter of the **CreateRegion** method of the **Engine** object.

Input parameter

This object is the input parameter of the following methods:

- **AnalyzeRegion** method of the **FRPage** object
- **AnalyzeRegion** method of the **DocumentAnalyzer** object
- **RemoveColorObjects**, **RemoveCameraBlur**, **RemoveCameraNoise**, **RemoveGarbage**, **SmoothImage**, **SaveImageRegionTo** methods of the **ImageDocument** object
- **AddCheckmark**, **InsertCheckmark** methods of the **CheckmarkGroup** object
- **AddSeparator**, **InsertSeparator** methods of the **SeparatorGroup** object

See also

Working with Properties

Bottom Property of the Region Object

This property returns the coordinate of the right border of the specified rectangle.

Visual Basic Syntax

```
Property Bottom(index As Long) As Long
  read-only
```

C++ Syntax

```
HRESULT get_Bottom(
   long  index,
   long* pVal
);
```

Parameters

index

[in] This parameter specifies the rectangle inside the region. It should be in the range from 0 to the value of the **IRegion::Count** property - 1.

pVal

[in] A pointer to **long** variable that receives the value of this property. Must not be NULL.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Region
Working with Properties

Left Property of the Region Object

This property returns the coordinate of the left border of the specified rectangle.

Visual Basic Syntax

```
Property Left(index As Long) As Long
  read-only
```

C++ Syntax

```
HRESULT get_Left(
   long  index,
   long* pVal
);
```

Parameters

index

[in] This parameter specifies the rectangle inside the region. It should be in the range from 0 to the value of the **IRegion::Count** property - 1.

pVal

[in] A pointer to **long** variable that receives the value of this property. Must not be NULL.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Region
Working with Properties

Right Property of the Region Object

This property returns the coordinate of the right border of the specified rectangle.

Visual Basic Syntax

```
Property Right(index As Long) As Long
  read-only
```

C++ Syntax

```
HRESULT get_Right(
   long  index,
   long* pVal
);
```

Parameters

index

[in] This parameter specifies the rectangle inside the region. It should be in the range from 0 to the value of the **IRegion::Count** property - 1.

pVal

[in] A pointer to **long** variable that receives the value of this property. Must not be NULL.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Region
Working with Properties

Top Property of the Region Object

This property returns the coordinate of the top border of the specified rectangle.

Visual Basic Syntax

```
Property Top(index As Long) As Long
  read-only
```

C++ Syntax

```
HRESULT get_Top(
   long  index,
   long* pVal
);
```

Parameters

index

[in] This parameter specifies the rectangle inside the region. It should be in the range from 0 to the value of the **IRegion::Count** property - 1.

pVal

[in] A pointer to **long** variable that receives the value of this property. Must not be NULL.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

Region
Working with Properties

AddRect Method of the Region Object

This method adds a new rectangle to the region.

Visual Basic Syntax

```
Method AddRect(
  left   As Long,
  top    As Long,
  right  As Long,
  bottom As Long
)
```

C++ Syntax

```
HRESULT AddRect(
  long left,
  long top,
  long right,
  long bottom
);
```

Parameters

left

[in] This parameter specifies coordinate of the left border of the rectangle.

top

[in] This parameter specifies coordinate of the top border of the rectangle.

right

[in] This parameter specifies coordinate of the right border of the rectangle.

bottom

[in] This parameter specifies coordinate of the bottom border of the rectangle.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

We recommend you to add rectangles in top to bottom order, because if a rectangle is inserted between the existing ones, it may change the region structure unpredictably.

See also

Region

FRRectangle Object (IFRRectangle Interface)

This object represents the location and size of a rectangle. It is used in a number of ABBYY FineReader Engine methods and properties as input or output parameter.

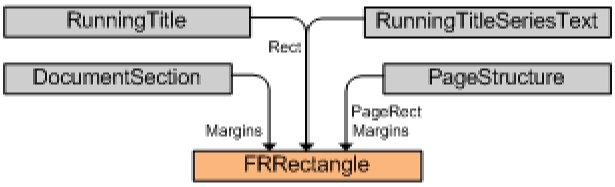
Properties

Name	Type	Description
Height	**Long**	Specifies the height of the rectangle.
Left	**Long**	Specifies the coordinate of the left border of the rectangle.
Top	**Long**	Specifies the coordinate of the top border of the rectangle.
Width	**Long**	Specifies the width of the rectangle.

Methods

Name	Description
CopyFrom	Initializes properties of the current object with values of similar properties of another object.
SetRectangle	Sets the location and size of the rectangle.

Related objects



Output parameter

This object is the output parameter of the **CreateRectangle** method of the **Engine** object.

Input parameter

This object is the input parameter of the **CreateCell** method of the **TextTable** object.

See also

Working with Properties

SetRectangle Method of the FRRectangle Object

This method allows setting the location and size of the rectangle.

Visual Basic Syntax

```
Method SetRectangle(
  Left   As Long,
  Top   As Long,
  Width   As Long,
  Height   As Long
)
```

C++ Syntax

```
HRESULT SetRectangle(
  long Left,
  long Top,
  long Width,
  long Height
);
```

Parameters

Left

[in] This parameter contains the coordinate of the left border of the rectangle.

Top

[in] This parameter contains the coordinate of the top border of the rectangle.

Width

[in] This parameter contains the width of the rectangle.

Height

[in] This parameter contains the height of the rectangle.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

See also

FRRectangle

IRecognizedPages Interface

This interface is to be implemented on the client side. It contains properties and methods necessary for passing recognized texts and images of the pages to be exported, one-by-one. The sequence of usage for this interface is as follows:

1. The user of ABBYY FineReader Engine implements an object with the **IRecognizedPages** interface. For C++, this object should be derived from this interface and implement its **get_PageIds**, **get_Layout**, **put_Layout**, **get_ImageDocument** and **raw_ReleasePage** methods. This object should also implement the methods of the **IUnknown** interface.

2. The user then passes a pointer to this object's interface into the **IEngine::SynthesizePagesEx** and **IExporter::ExportPagesEx** methods as one of their input parameters. ABBYY FineReader Engine will call the properties and methods of this object to get the pointers to the next page layout and image document.

Why use this interface to export a large number of recognized pages into one file?

When exporting a large number of recognized pages into a single file, you have the alternative of using the **IExporter::ExportPages** method. This method requires objects of the **IImageDocumentsCollection** and **ILayoutsCollection** types to be passed to it as input parameters. This means that you have to store all your recognized texts and corresponding image objects in memory. For a large number of pages this may become critical from the point of view of memory consumption. In practice, only several hundred of recognized pages may fit into computer memory.

The **IExporter::ExportPagesEx** method allows you to export an unlimited number of recognized pages, because it requires image document and layout objects corresponding only to one recognized page to be loaded into memory at a time. So you may:

1. Open the image with the help of the **IEngine::PrepareImage** method and specify the temporary folder where the image will be saved. Then load the image into memory with the help of the **IEngine::OpenImage** method and recognize it.

2. Save the image and recognized text as files with the help of the **IImageDocument::SaveModified** and **ILayout::SaveToFile** methods.

3. Use the **IRecognizedPages** interface to implement your own object. This object must obtain the **ImageDocument** object with the help of the **IEngine::OpenImage** method using the saved images and return it through the **get_ImageDocument** method. This object must also obtain the **Layout** object: it must return it through the **get_Layout** method and set it through the **put_Layout** method if the layout has been modified. It must release all unreleased objects when the **raw_ReleasePage** method is called.

When exporting image files into PDF (PDF/A) format using the **PEM_ImageOnly** mode, you may skip the recognition stage altogether as the layout information is not used here. In this case ABBYY FineReader Engine does not call the **get_Layout** or **put_Layout** methods, and your implementation of these methods will not result in any action. However, the methods must be present as their presence is required by the definition of the COM interface.

Properties

Name	Type	Description
PageIds	**LongsCollection**, read-only	Returns the collection of identifiers of pages to be exported.
ImageDocument	**ImageDocument**, read-only	Returns the **ImageDocument** object for the specified page.
Layout	**Layout**	Returns or sets the **Layout** object for the specified page.

Methods

Name	Description
ReleasePage	This method is called after the page has been processed to release the cached objects. Any clean-up should be done here.

Sample

Visual C++ (COM) code

```
class CRecognizePagesImpl : public FREngine::IRecognizedPages {
 public:
  CRecognizePagesImpl()
  {
  pageIds = Engine->CreateLongsCollection();
  fileNames = Engine->CreateStringsCollection();
  }

  void AddPage( FREngine::IImageDocument* imageDoc, FREngine::ILayout* layout )
  {
  long pageId = imageDoc->Id;
  _bstr_t imagePath = imageDoc->Path;

  imageDoc->SaveModified();
  layout->SaveToFile( imagePath + L".layout" );

  fileNames->Add( imagePath );
  pageIds->Add( pageId );
  }


//////////////////////////////////////////////////////////////////////////////////////
///
  // IRecognizedPages implementation

  // Since we completely control callback object lifetime
  // there is no need to implement reference counter
  ULONG STDMETHODCALLTYPE AddRef() { return 1; }
  ULONG STDMETHODCALLTYPE Release() { return 1; }

  HRESULT STDMETHODCALLTYPE QueryInterface( REFIID riid, void** ppObject )
  {
  *ppObject = 0;
  if( riid == __uuidof( IUnknown ) || riid == __uuidof( FREngine::IRecognizedPages ) )
  {
  *ppObject = this;
  AddRef();
  return S_OK;
  } else {
  return E_NOINTERFACE;
  }
  }

  HRESULT STDMETHODCALLTYPE get_PageIds( FREngine::ILongsCollection **result )
  {
  pageIds.AddRef();
  *result = pageIds;
  return S_OK;
  }

  HRESULT STDMETHODCALLTYPE get_ImageDocument( long pageId, FREngine::IImageDocument**
result )
  {
  int index = findPageId( pageId );
  FREngine::IImageDocumentPtr image = Engine->OpenImage( fileNames->Item( index ) );
  image.AddRef();
  *result = image;
  return S_OK;
  }

  HRESULT STDMETHODCALLTYPE get_Layout( long pageId, FREngine::ILayout** result )
  {
  int index = findPageId( pageId );
  FREngine::ILayoutPtr layout = Engine->CreateLayout();
  layout->LoadFromFile( fileNames->Item( index ) + L".layout" );
  layout.AddRef();
  *result = layout;
  return S_OK;
  }

  HRESULT STDMETHODCALLTYPE put_Layout( long pageId, FREngine::ILayout* layout )
  {
  int index = findPageId( pageId );
  layout->SaveToFile( fileNames->Item( index ) + L".layout" );
  return S_OK;
  }

  HRESULT STDMETHODCALLTYPE raw_ReleasePage( long pageId )
  {
  return S_OK;
  }

 private:
  FREngine::ILongsCollectionPtr pageIds;
  FREngine::IStringsCollectionPtr fileNames;

  int findPageId( long pageId )
  {
  int pageIdsCount = pageIds->Count;
  for( int i = 0; i < pageIdsCount; i++ ) {
  if( pageIds->Item( i ) == pageId ) {
  return i;
  }
  }
  return -1;
  }
 };

 void processMultiPageOldAPI()
 {
  displayMessage( L"Loading image..." );
  _bstr_t imagePath = ::GetSamplesFolder();
  imagePath += L"\\SampleImages\\Demo.tif";

  FREngine::IDocumentInfoPtr docInfo = Engine->CreateDocumentInfo();
  FREngine::IStringsCollectionPtr imageNames = Engine->PrepareImage( imagePath,
L"d:\\temp", 0, -1, 0, docInfo );

  CRecognizePagesImpl recognizedPages;
  int pagesCount = imageNames->Count;

  for( int i = 0; i < pagesCount; i++ ) {
  _bstr_t imageName = imageNames->Item( i );
  FREngine::IImageDocumentPtr imageDoc = Engine->OpenImage( imageName );
  FREngine::ILayoutPtr layout = Engine->CreateLayout();

  Engine->AnalyzeAndRecognizePage( imageDoc, 0, 0, layout, docInfo );
  recognizedPages.AddPage( imageDoc, layout );
  }

  Engine->SynthesizePagesEx( &recognizedPages, 0, docInfo );

  _bstr_t exportPath = ::GetSamplesFolder();
  exportPath += L"\\SampleImages\\Demo-oldmp.rtf";

  FREngine::IExporterPtr exporter = Engine->CreateExporter();
  exporter->ExportPagesEx( FREngine::FEF_RTF, exportPath, &recognizedPages, 0, docInfo,
0, 0 );
 }
```

See also

IExporter::ExportPagesEx
IEngine::SynthesizePagesEx

UserProperty Property

This property allows you to associate any user-defined information with an object. This information is passed as **VARIANT**, which may contain only simple types (String, integer types), but no **SAFEARRAY** or **VARIANT** types may be contained inside this **VARIANT.** More precisely, only the following variant types are allowed: VT_EMPTY, VT_UI1, VT_I2, VT_I4, VT_R4, VT_R8, VT_CY, VT_BSTR, VT_NULL, VT_ERROR, VT_BOOL, VT_DATE.

Visual Basic Syntax

```
Property UserProperty(name As String) As Variant
```

C++ Syntax

```
HRESULT get_UserProperty(
   BSTR     name,
   VARIANT* pVal
);
HRESULT put_UserProperty(
   BSTR    name,
   VARIANT newVal
);
```

Parameters

name

[in] This variable contains any string value you want to identify the property among others, for example, "MyProperty".

pVal

[out] A pointer to **VARIANT** variable that receives the value of the user-defined property.

newVal

[in] A **VARIANT** variable that contains the new value for the property.

Return Values

This function has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Properties are identified by their names. Thus only one property with the given name is allowed for a given object. If an object does not have a user property with the given name, it is created when a value for this property is first assigned. When trying to get a value of the property that does not exist in an object, an empty VARIANT is returned. When copying an object via the **CopyFrom** method, user-defined properties are also copied. If an object may persist, user-defined properties are also persistent.

The following objects provide this property:

- **BaseLanguage**
- **Block**
- **Layout**
- **TextLanguage**

See also

Working with Properties

CopyFrom Method

This method initializes properties of the current object with the values of similar properties of another object.

Visual Basic Syntax

```
Method CopyFrom(
  otherObject As <ObjectType>
)
```

C++ Syntax

```
HRESULT CopyFrom(
  I<ObjectType>* otherObject
);
```

Parameters

otherObject

[in] This variable refers to the object of the same type as the current one. This object serves as a source data to be copied into the new object.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Sample

Visual C++ (COM) code

```
// Global FineReader Engine object.
```

```
FREngine::IEnginePtr Engine;
// Create new TextLanguage object
FREngine::ITextLanguagePtr pTextLanguage = Engine->CreateTextLanguage();
// Copy all attributes from predefined English language
FREngine::ITextLanguagePtr pEnglishLanguage =
            Engine->PredefinedLanguages->FindLanguage( "English" )->TextLanguage;
pTextLanguage->CopyFrom( pEnglishLanguage );
pTextLanguage->InternalName = "SampleTL";
```

Visual Basic code

```
Public Engine As FREngine.Engine
' Create new TextLanguage object
Dim TextLanguage As FREngine.TextLanguage
Set TextLanguage = Engine.CreateTextLanguage

' Copy all attributes from predefined English language
TextLanguage.CopyFrom _
Engine.PredefinedLanguages.FindLanguage("English").TextLanguage
TextLanguage.InternalName = "SampleTL"
TextLanguage.BaseLanguages(0).InternalName = "SampleBL"
```

Remark

The following objects provide this method:

- Image-related objects
  - **ImageDocumentsCollection**
  - **ImageProcessingParams**
  - **PrepareImageMode**
  - **ImageModification**
  - **JpegExtendedParams**
  - **TrainingImagesCollection**
- Layout and blocks:
  - **Layout**
  - **LayoutBlocks**
  - **LayoutsCollection**
  - **BarcodeBlock**
  - **CheckmarkBlock**
  - **CheckmarkGroup**
  - **SeparatorBlock**
  - **SeparatorGroup**
  - **TextBlock**
  - **RasterPictureBlock**
  - **BarcodeSymbol**
- Language-related objects

  - **TextLanguage**
  - **BaseLanguage**
  - **FuzzyStringsCollection**
- Text-related objects
  - **ParagraphParams**
  - **CharParams**
  - **TabPositions**
  - **TabPosition**
  - **TextOrientation**
- Analysis, recognition, and export parameters
  - **PageProcessingParams**
  - **PageAnalysisParams**
  - **TableAnalysisParams**
  - **BarcodeParams**
  - **ObjectsExtractionParams**
  - **OrientationDetectionParams**
  - **RecognizerParams**
  - **SynthesisParamsForDocument**
  - **DocumentStructureDetectionParams**
  - **FontFormattingDetectionParams**
  - **SynthesisParamsForPage**
- Export parameters
  - **HTMLExportParams**
  - **PPTExportParams**
  - **RTFExportParams**
  - **TextExportParams**
  - **XLExportParams**
  - **XMLExportParams**
  - **PDFExportParams**
  - **PDFEncryptionInfo**
  - **PDFAExportParamsOld**
  - **PDFExportParamsOld**
- Supplementary objects
  - **StringsCollection**

  - **LongsCollection**
  - **Region**
  - **FRRectangle**
- Document synthesis objects
  - **FootnoteSeries**
  - **ParagraphStyle**
- **DocumentInformationDictionary**

See also

See sample: CustomLanguage

LoadFromFile Method

This method restores the contents of the object from a file on disk, where it should have previously been saved by the **SaveToFile** method.

Visual Basic Syntax

```
Method LoadFromFile(
  path As String
)
```

C++ Syntax

```
HRESULT LoadFromFile(
  BSTR path
);
```

Parameters

path

[in] A path to the file on disk where the contents of the object is stored. If a file specified by this path was not obtained as a result of a call to the **SaveToFile** method of an object of the same type as the current one, some specific error code is returned.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Note that in the case when an object data is read by a functionally limited ABBYY FineReader Engine version, while it was saved by a version with a wider functional set, object properties that cannot be changed in the limited version, are restored to default values.

The following objects provide this method:

- Image-related objects
  - **ImageProcessingParams**
  - **PrepareImageMode**
  - **ImageModification**
  - **JpegExtendedParams**
- Language-related objects
  - **TextLanguage**
  - **BaseLanguage**
- Analysis, recognition, and export parameters

  - **PageProcessingParams**
  - **PageAnalysisParams**
  - **TableAnalysisParams**
  - **BarcodeParams**
  - **ObjectsExtractionParams**
  - **OrientationDetectionParams**
  - **RecognizerParams**
  - **SynthesisParamsForDocument**
  - **DocumentStructureDetectionParams**
  - **FontFormattingDetectionParams**
  - **SynthesisParamsForPage**
- Export parameters
  - **HTMLExportParams**
  - **PPTExportParams**
  - **RTFExportParams**
  - **TextExportParams**
  - **XLExportParams**
  - **XMLExportParams**
  - **PDFExportParams**
  - **PDFEncryptionInfo**
  - **PDFAExportParamsOld**
  - **PDFExportParamsOld**
- **Layout**
- **ParagraphParams**
- **Region**

See also

SaveToFile
SaveToMemory
LoadFromMemory

LoadFromMemory Method

This method restores the object contents from the global memory.

Visual Basic Syntax

```
Method LoadFromMemory( hGlobal As Long )
```

C++ Syntax

```
HRESULT LoadFromMemory(
  long hGlobal
);
```

Parameters

hGlobal

[in] This parameter specifies the HGLOBAL handle of the memory from where the object contents should be loaded. The parameter is statically casted to the **Long** type. This handle should be the one obtained from the **SaveToMemory** method of an object of the same type as the current one, and should be valid (not freed by the **GlobalFree** function).

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

The size of the memory area that the object allocates, can be obtained by calling the **GlobalSize** function.

Note that in the case when an object data is read by a functionally limited ABBYY FineReader Engine version, while it was saved by a version with a wider functional set, object properties that cannot be changed in the limited version, are restored to default values.

The following objects provide this method:

- Image-related objects
  - **ImageProcessingParams**
  - **PrepareImageMode**
  - **ImageModification**
  - **JpegExtendedParams**
- Language-related objects
  - **TextLanguage**
  - **BaseLanguage**
- Analysis, recognition, and export parameters
  - **PageProcessingParams**
  - **PageAnalysisParams**
  - **TableAnalysisParams**
  - **BarcodeParams**
  - **ObjectsExtractionParams**
  - **OrientationDetectionParams**
  - **RecognizerParams**
  - **SynthesisParamsForDocument**
  - **DocumentStructureDetectionParams**
  - **FontFormattingDetectionParams**
  - **SynthesisParamsForPage**
- Export parameters
  - **HTMLExportParams**
  - **PPTExportParams**
  - **RTFExportParams**
  - **TextExportParams**

    - **XLExportParams**
    - **XMLExportParams**
    - **PDFExportParams**
    - **PDFEncryptionInfo**
    - **PDFAExportParamsOld**
    - **PDFExportParamsOld**
- **Layout**
- **ParagraphParams**
- **Region**

See also

SaveToMemory
SaveToFile
LoadFromFile

SaveToFile Method

This method saves the object contents into a file on disk.

Visual Basic Syntax

```
Method SaveToFile(
  path As String
)
```

C++ Syntax

```
HRESULT SaveToFile(
  BSTR path
);
```

Parameters

path

[in] This parameter specifies the path to the file where the object contents should be saved. If a file with this name already exists, it is overwritten without prompt.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Note that in the case when an object data is read by a functionally limited ABBYY FineReader Engine version, while it was saved by a version with a wider functional set, object properties that cannot be changed in the limited version, are restored to default values.

The following objects provide this method:

- Image-related objects
    - **ImageProcessingParams**
    - **PrepareImageMode**
    - **ImageModification**
    - **JpegExtendedParams**
- Language-related objects
    - **TextLanguage**

  - **BaseLanguage**
- Analysis, recognition, and export parameters
  - **PageProcessingParams**
  - **PageAnalysisParams**
  - **TableAnalysisParams**
  - **BarcodeParams**
  - **ObjectsExtractionParams**
  - **OrientationDetectionParams**
  - **RecognizerParams**
  - **SynthesisParamsForDocument**
  - **DocumentStructureDetectionParams**
  - **FontFormattingDetectionParams**
  - **SynthesisParamsForPage**
- Export parameters
  - **HTMLExportParams**
  - **PPTExportParams**
  - **RTFExportParams**
  - **TextExportParams**
  - **XLExportParams**
  - **XMLExportParams**
  - **PDFExportParams**
  - **PDFEncryptionInfo**
  - **PDFAExportParamsOld**
  - **PDFExportParamsOld**
- **Layout**
- **ParagraphParams**
- **Region**

See also

LoadFromFile
SaveToMemory
LoadFromMemory

SaveToMemory Method

This method saves the contents of the object into the global memory and returns an HGLOBAL handle — casted to the **Long** type, of the memory area allocated for the object. It is user's responsibility to deallocate this memory when it is no longer needed. As the memory is allocated by the **GlobalAlloc** API function, it should be deallocated by the **GlobalFree** function. The size of the memory area that the object allocates, can be obtained by calling the **GlobalSize** function.

Visual Basic Syntax

```
Method SaveToMemory() As Long
```

C++ Syntax

```
HRESULT SaveToMemory(
  long* hGlobal
);
```

Parameters

hGlobal

[out, retval] A pointer to a **long** variable that receives the HGLOBAL handle — casted to **long** — of the memory area allocated for the object. Should not be NULL.

Return Values

This method has no specific return values. It returns standard return values of ABBYY FineReader Engine functions.

Remarks

Note that in the case when an object data is read by a functionally limited ABBYY FineReader Engine version, while it was saved by a version with a wider functional set, object properties that cannot be changed in the limited version, are restored to default values.

The following objects provide this method:

- Image-related objects
  - **ImageProcessingParams**
  - **PrepareImageMode**
  - **ImageModification**
  - **JpegExtendedParams**
- Language-related objects
  - **TextLanguage**
  - **BaseLanguage**
- Analysis, recognition, and export parameters
  - **PageProcessingParams**
  - **PageAnalysisParams**
  - **TableAnalysisParams**
  - **BarcodeParams**
  - **ObjectsExtractionParams**
  - **OrientationDetectionParams**
  - **RecognizerParams**
  - **SynthesisParamsForDocument**
  - **DocumentStructureDetectionParams**
  - **FontFormattingDetectionParams**
  - **SynthesisParamsForPage**
- Export parameters
  - **HTMLExportParams**

  - **PPTExportParams**
  - **RTFExportParams**
  - **TextExportParams**
  - **XLExportParams**
  - **XMLExportParams**
  - **PDFExportParams**
  - **PDFEncryptionInfo**
  - **PDFAExportParamsOld**
  - **PDFExportParamsOld**
- **Layout**
- **ParagraphParams**
- **Region**

See also

LoadFromMemory
SaveToFile
LoadFromFile

Enumerations

AEF_ prefixed flags

The **AEF_** prefixed flags are used to denote the possible ABBYY FineReader Engine export formats whose availability depends on the license. The **ILicense::AvailableExportFormats** property returns a bitwise OR combination of zero or more of these flags values, where each set bit indicates that the corresponding export format is available in the license.

```
module AvailableExportFormatesFlags
 {
   const long AEF_RTF           = 0x00000001;
   const long AEF_HTML          = 0x00000002;
   const long AEF_XLS           = 0x00000004;
   const long AEF_PDF           = 0x00000008;
   const long AEF_Text          = 0x00000020;
   const long AEF_PDFImageOnly  = 0x00000040;
   const long AEF_XML           = 0x00000080;
   const long AEF_PPT           = 0x00000100;
   const long AEF_PDFA          = 0x00000200;
   const long AEF_PDFMRC        = 0x00000800;
 };
```

Elements

Flag name	Description
AEF_RTF	RTF, DOC, DOCX export format.
AEF_HTML	HTML export format.
AEF_XLS	XLS, XLSX export format.
AEF_PDF	PDF export format.
AEF_Text	Text export format.
AEF_PDFImageOnly	PDF Image Only export format.
AEF_XML	ABBYY XML export format.
AEF_PPT	PPTX export format.

AEF_PDFA	PDF/A export format.
AEF_PDFMRC	PDF MRC export format.

See also

License
ABBYY FineReader Engine 10 Modules

AEM_ prefixed flags

The **AEM_** prefixed flags are used to denote the possible ABBYY FineReader Engine modules whose availability depends on the license. The **ILicense::AvailableEngineModules** property returns a bitwise OR combination of zero or more of these flags values, where each set bit indicates that the corresponding ABBYY FineReader Engine module is available in the license.

```
const long AEM_1DBarcode             = 0x00000001;
const long AEM_PDF417                = 0x00000002;
const long AEM_Aztec                 = 0x00000004;
const long AEM_QRCode                = 0x00000008;
const long AEM_DataMatrix            = 0x00000010;
const long AEM_BarcodeAutolocation   = 0x00000020;
const long AEM_Analyze               = 0x00000040;
const long AEM_FullTextIndexDA       = 0x00000080;
const long AEM_FlexiFormsDA          = 0x00000100;
const long AEM_CyrillicHandprint     = 0x00000200;
const long AEM_OMR                   = 0x00000400;
const long AEM_ExtendedCharacterInfo = 0x00000800;
const long AEM_ASCII                 = 0x00001000;
const long AEM_OpenPDF               = 0x00002000;
const long AEM_UserPatterns          = 0x00004000;
const long AEM_BalancedMode          = 0x00008000;
const long AEM_FastMode              = 0x00010000;
const long AEM_CameraOCR             = 0x00020000;
const long AEM_ColorFiltering        = 0x00040000;
```

Elements

Name	Description
AEM_1DBarcode	1D Barcodes module.
AEM_PDF417	PDF417 module.
AEM_Aztec	Aztec module.
AEM_QRCode	QR Code module.
AEM_DataMatrix	DataMatrix module.
AEM_BarcodeAutolocation	Barcode Autolocation module.
AEM_Analyze	Document Analysis module.
AEM_FullTextIndexDA	DA for Full-Text Indexing module.
AEM_FlexiFormsDA	DA for Invoices module.
AEM_CyrillicHandprint	Cyrillic ICR module.
AEM_OMR	OMR module.
AEM_ExtendedCharacterInfo	Extended Character Info module.
AEM_ASCII	ASCII License Basic Modules module.
AEM_OpenPDF	PDF Opening module.
AEM_UserPatterns	User Patterns module.
AEM_BalancedMode	Balanced Mode module.
AEM_FastMode	Fast Mode module.
AEM_CameraOCR	Camera OCR module.
AEM_ColorFiltering	Color Filtering module.

See also

License
ABBYY FineReader Engine 10 Modules

ALS_ prefixed flags

The **ALS_** prefixed flags are used to denote the possible ABBYY FineReader Engine language sets whose availability depends on the license. The **ILicense::AvailableLanguageSets** property returns a bitwise OR combination of zero or more of these flags values, where each set bit indicates that the corresponding ABBYY FineReader Engine language set is available in the license.

```
module AvailableLanguageSetsFlags
 {
   const long ALS_Standard      = 0x00000001;
   const long ALS_DataCapture   = 0x00000002;
   const long ALS_Artificial    = 0x00000004;
   const long ALS_FineReaderXIX = 0x00000008;
   const long ALS_Programming   = 0x00000010;
   const long ALS_User          = 0x00000020;
   const long ALS_CJK           = 0x00000040;
   const long ALS_Hebrew        = 0x00000080;
   const long ALS_Thai          = 0x00000100;
   const long ALS_Vietnamese    = 0x00000200;
   const long ALS_Arabic        = 0x00000400;
 };
```

Elements

Name	Description
ALS_Standard	Natural languages module.
ALS_DataCapture	Natural for Data Capture languages module.
ALS_Artificial	Artificial languages module.
ALS_FineReaderXIX	FineReader XIX languages module.
ALS_Programming	Programming languages module.
ALS_User	User (Custom) OCR Languages module.
ALS_CJK	Chinese, Japanese, and Korean languages modules.
ALS_Hebrew	Hebrew and Yiddish languages modules.
ALS_Thai	Thai languages module.
ALS_Vietnamese	Vietnamese languages module.

See also

License
ABBYY FineReader Engine 10 Modules

ATT_ prefixed flags

The **ATT_** prefixed flags are used to denote the possible ABBYY FineReader Engine text types whose availability depends on the license. The **ILicense::AvailableTextTypes** property returns a bitwise OR combination of zero or more of these flags values, where each set bit indicates that the corresponding text type is available in the license.

```
module AvailableTextTypesFlags
 {
   const long ATT_Normal      = 0x00000001;
   const long ATT_Typewriter  = 0x00000002;
   const long ATT_Matrix      = 0x00000004;
   const long ATT_Index       = 0x00000008;
   const long ATT_Handprinted = 0x00000010;
   const long ATT_OCR_A       = 0x00000020;
   const long ATT_OCR_B       = 0x00000040;
   const long ATT_MICR_E13B   = 0x00000080;
   const long ATT_Gothic      = 0x00000100;
   const long ATT_MICR_CMC7   = 0x00000200;
   const long ATT_Fax         = 0x00000400;
 };
```

Elements

Flag name	Description
ATT_Normal	Normal text type (TextTypeEnum::TT_Normal)
ATT_Typewriter	Typewriter text type (TextTypeEnum::TT_Typewriter)
ATT_Matrix	Matrix text type (TextTypeEnum::TT_Matrix)
ATT_Index	Index text type (TextTypeEnum::TT_Index)
ATT_Handprinted	Handprinted text type (TextTypeEnum::TT_Handprinted)
ATT_OCR_A	OCR-A text type (TextTypeEnum::TT_OCR_A)
ATT_OCR_B	OCR-B text type (TextTypeEnum::TT_OCR_B)
ATT_MICR_E13B	E13B language and MICR text type (TextTypeEnum::TT_MICR_E13B).
ATT_Gothic	Gothic text type (TextTypeEnum::TT_Gothic). Available if and only if the *ABBYY FineReader XIX* module is available.
ATT_MICR_CMC7	CMC7 language and MICR text type (TextTypeEnum::TT_MICR_CMC7).
ATT_Fax	Normal text type with low resolution (TextTypeEnum::TT_Normal and **IRecognizerParams::LowResolutionMode** property set to TRUE)

See also

License
ABBYY FineReader Engine 10 Modules

AVC_ prefixed flags

The **AVC_** prefixed flags are used to denote the possible ABBYY FineReader Engine modules for Visual Components whose availability depends on the license. The **ILicense::AvailableVisualComponents** property returns a bitwise OR combination of zero or more of these flags values, where each set bit indicates that the corresponding Visual Component module is available in the license.

```
const long AVC_ImageViewer    = 0x00000001;
const long AVC_DocumentViewer = 0x00000002;
const long AVC_TextEditor     = 0x00000004;
const long AVC_TextValidator  = 0x00000008;
const long AVC_Scanning       = 0x00000010;
const long AVC_Training       = 0x00000020;
```

Elements

Flag name	Description
AVC_ImageViewer	*Image Viewing and Blocks Drawing* module. This module is currently not supported.
AVC_DocumentViewer	*Document Batch Managing* module. This module is currently not supported.
AVC_TextEditor	*Text Viewing and Editing* module. This module is currently not supported.
AVC_TextValidator	*Full-Text Verification* module. This module is currently not supported.
AVC_Scanning	*Scanning* module.
AVC_Training	User Patterns Training module.

See also

License
ABBYY FineReader Engine 10 Modules

BF_ prefixed flags

The **BF_** prefixed flags are used to denote borders of an incut frame. The **IIncut::Borders** property returns a bitwise OR combination of zero or more of these flags values, where each set bit indicates that the corresponding border is visible.

```
module BorderFlags
```

```
{
  const long BF_Top    = 0x00000001;
  const long BF_Bottom = 0x00000002;
  const long BF_Left   = 0x00000004;
  const long BF_Right  = 0x00000008;
}
```

Elements

Flag name	Description
BF_Top	Top separator.
BF_Bottom	Bottom separator.
BF_Left	Left separator.
BF_Right	Right separator.

See also

IIncut::Borders

BackgroundColorModeEnum

BackgroundColorModeEnum enumeration constants are used to denote modes of background color saving during export.

```
typedef enum {
      BCM_DontSave,
      BCM_BlackWhite,
      BCM_ColorForInverted,
      BCM_Color
} BackgroundColorModeEnum;
```

Elements

Name	Description
BCM_DontSave	The background color is not saved.
BCM_BlackWhite	The background is saved in black-and-white.
BCM_ColorForInverted	The background color is saved only for inverted blocks.
BCM_Color	The background color is saved.

See also

IRTFExportParams::BackgroundColorMode
IXLExportParams::BackgroundColorMode
IPPTExportParams::BackgroundColorMode

BaseLanguageLetterSetEnum

BaseLanguageLetterSetEnum enumeration constants are used to describe different letter sets of a base language. Letter sets are the sets of characters that are allowed or disallowed in certain places inside the words in a specified language.

```
typedef enum {
      BLLS_Alphabet,
      BLLS_Prefixes,
      BLLS_Suffixes,
      BLLS_IgnorableLetters,
      BLLS_SubscriptAlphabet,
      BLLS_SuperscriptAlphabet
} BaseLanguageLetterSetEnum;
```

Elements

Name	Description

BLLS_Alphabet	This value denotes a letter set that includes the full alphabet of the base language.
BLLS_Prefixes	This constant denotes a letter set that covers the punctuation marks that may be found immediately before words. Among these characters there may be ", (, { and so on.
BLLS_Suffixes	This constant denotes a letter set that covers the punctuation marks that may be found immediately after words. Among these characters there may be !, ",), } and so on.
BLLS_IgnorableLetters	This value denotes a letter set that includes the characters that may be found inside a word, but are ignored during the internal spelling check.
BLLS_SubscriptAlphabet	This value denotes a letter set that includes the characters allowed inside the words of the language as subscripts.
BLLS_SuperscriptAlphabet	This value denotes a letter set that includes the characters allowed inside the words of the language as superscripts.

See also

IBaseLanguage::LetterSet

BarcodeOrientationEnum

BarcodeOrientationEnum enumeration constants are used to denote the types of barcode orientation that can be detected by ABBYY FineReader Engine. It is used by the **BarcodeParams** object.

```
typedef enum {
      BO_Unknown       = 0x00000000,
      BO_Left_To_Right = 0x00000001,
      BO_Down_To_Top   = 0x00000002,
      BO_Right_To_Left = 0x00000004,
      BO_Top_To_Down   = 0x00000008,
      BO_Autodetect    = BO_Left_To_Right | BO_Down_To_Top | BO_Right_To_Left |
BO_Top_To_Down
} BarcodeOrientationEnum;
```

Elements

Name	**Description**
BO_Unknown	Denotes unknown type of barcode orientation. It may be used as the return value if ABBYY FineReader Engine has failed to detect barcode orientation.
BO_Left_To_Right	Barcode is oriented from left to right.
BO_Down_To_Top	Barcode is oriented from down to top.
BO_Right_To_Left	Barcode is oriented from right to left.
BO_Top_To_Down	Barcode is oriented from top to down.
BO_Autodetect	Detect the barcode orientation automatically.

See also

IBarcodeParams::Orientation

BarcodeSupplementTypeEnum

BarcodeSupplementTypeEnum enumeration constants are used to denote the types of supplementary barcodes that can be recognized by ABBYY FineReader Engine. The barcodes of the EAN 8, 13, UPC-A, and UPC-E types may include supplementary barcodes which may contain 2 or 5 digits.

```
typedef enum {
      BS_Unknown    = 0x00000000,
      BS_Void       = 0x00000001,
      BS_2Digits    = 0x00000002,
      BS_5Digits    = 0x00000004,
      BS_Autodetect = BS_Void | BS_2Digits | BS_5Digits
} BarcodeSupplementTypeEnum;
```

Elements

Name	Description
BS_Unknown	Denotes unknown type of supplementary barcode. It may be used as the return value if ABBYY FineReader Engine has failed to detect the type of supplementary barcode.
BS_Void	No supplementary barcode.
BS_2Digits	2-digit supplementary barcode.
BS_5Digits	5-digit supplementary barcode.
BS_Autodetect	Forces ABBYY FineReader Engine to automatically detect the supplementary barcode type during recognition.

See also

Barcode Types
IBarcodeBlock::SupplementType
IBarcodeParams::SupplementType

BarcodeTypeEnum

BarcodeTypeEnum enumeration constants are used to denote the types of barcodes that can be recognized by ABBYY FineReader Engine. These constants can be used during recognition to specify the types of barcodes to be recognized, or after recognition to define the types of recognized barcodes.

```
typedef enum {
      BT_Unknown       = 0x00000000,
      BT_Code39        = 0x00000001,
      BT_Interleaved25 = 0x00000002,
      BT_EAN13         = 0x00000004,
      BT_Code128       = 0x00000008,
      BT_EAN8          = 0x00000010,
      BT_PDF417        = 0x00000020,
      BT_Codabar       = 0x00000040,
      BT_UPCE          = 0x00000080,
      BT_Industrial25  = 0x00000100,
      BT_IATA25        = 0x00000200,
      BT_Matrix25      = 0x00000400,
      BT_Code93        = 0x00000800,
      BT_PostNet       = 0x00001000,
      BT_UCC128        = 0x00002000,
      BT_Patch         = 0x00004000,
      BT_Aztec         = 0x00008000,
      BT_DataMatrix    = 0x00010000,
      BT_QRCode        = 0x00020000,
      BT_UPCA          = 0x00040000,
      BT_Autodetect    = BT_Code39 | BT_Interleaved25 | BT_EAN13 | BT_Code128 |
BT_EAN8 | BT_PDF417 | BT_Codabar | BT_UPCE | BT_Industrial25 | BT_IATA25 | BT_Matrix25
| BT_Code93 | BT_PostNet | BT_UCC128 | BT_Patch | BT_Aztec | BT_DataMatrix | BT_QRCode
| BT_UPCA
} BarcodeTypeEnum;
```

Elements

Name	Description
BT_Unknown	Denotes unknown type of barcode. It may be used as the return value if ABBYY FineReader Engine has failed to detect the type of barcode.
BT_Code39	Barcode in Code 39 standard.
BT_Interleaved25	Barcode in Interleaved 2 of 5 standard.

BT_EAN13	Barcode in EAN 13 standard.
BT_Code128	Barcode in Code 128 standard.
BT_EAN8	Barcode in EAN 8 standard.
BT_PDF417	Barcode in PDF417 standard.
BT_Codabar	Barcode in Codabar standard.
BT_UPCE	Barcode in UPC-E standard.
BT_Industrial25	Barcode in Industrial 2 of 5 standard.
BT_IATA25	Barcode in IATA 2 of 5 standard.
BT_Matrix25	Barcode in Matrix 2 of 5 standard.
BT_Code93	Barcode in Code 93 standard.
BT_PostNet	Barcode in PostNet standard.
BT_UCC128	Barcode in UCC-128 standard.
BT_Patch	Barcode in Patch standard.
BT_Aztec	Barcode in Aztec standard.
BT_DataMatrix	Barcode in Data Matrix standard.
BT_QRCode	Barcode in QR Code standard.
BT_UPCA	Barcode in UPC-A standard.
BT_Autodetect	Forces ABBYY FineReader Engine to automatically detect the barcode type during recognition.

See also

Barcode Types
IBarcodeBlock::BarcodeType
IBarcodeParams::Type

BlockLayerTypeEnum

BlockLayerTypeEnum enumeration constants are used to designate the layers to which blocks belong. Blocks may be overlaid, for example, a text block may lay over a background picture block.

```
typedef enum {
      BLT_Unknown,
      BLT_Background,
      BLT_Foreground,
      BLT_Hidden
} BlockLayerTypeEnum;
```

Elements

Name	Description
BLT_Unknown	The layer is undefined.
BLT_Background	The block belongs to background of the page.
BLT_Foreground	The block is on foreground of the page.
BLT_Hidden	The block is hidden. Such blocks appear in the layout, if the **IObjectsExtractionParams::FullTextIndexDA** was set to TRUE during recognition and a text was found on a picture.

See also

IBlock::BlockLayerType

BlockRoleEnum

BlockRoleEnum enumeration constants are used to describe the role of the text block in the logic structure of the document.

```
typedef enum {
      BR_Unknown,
      BR_RunningTitle,
      BR_MainText,
      BR_IncutText,
      BR_Caption,
      BR_LineNumbers,
      BR_Artefact
} BlockRoleEnum;
```

Elements

Name	Description
BR_Unknown	The role is undefined.
BR_RunningTitle	The block contains a running title.
BR_MainText	The block contains the main text of the page.
BR_IncutText	The block contains an incut.
BR_Caption	The block contains a caption.
BR_LineNumbers	The block contains line numbers.
BR_Artefact	The block contains some garbage text.

See also

ITextBlock::BlockRole

BlockTypeEnum

BlockTypeEnum enumeration constants are used to designate the type of a block.

```
typedef enum {
      BT_Text,
      BT_RasterPicture,
      BT_Table,
      BT_Barcode,
      BT_Checkmark,
      BT_CheckmarkGroup
      BT_VectorPicture,
      BT_Separator,
      BT_SeparatorGroup
} BlockTypeEnum;
```

Elements

Name	Description
BT_Text	Designates a text block. It corresponds to an image zone recognized as formatted text. Additional properties of the blocks of this type are accessible via the **TextBlock** object.
BT_RasterPicture	Designates a raster picture. The part of the image that this block encloses is not recognized, and the block is exported "as is". Properties of this block type are represented by the **RasterPictureBlock** object.
BT_Table	Designates a table block. It corresponds to an image zone recognized as table. A table region may only be rectangular. Additional properties of the blocks of this type are accessible via the **TableBlock** object.
BT_Barcode	Designates a barcode block. It corresponds to an image zone recognized as barcode. Additional properties of the blocks of this type are accessible via the **BarcodeBlock** object.
BT_Checkmark	Designates a checkmark block. It corresponds to an image zone recognized as checkmark. Additional properties of the blocks of this type are accessible via the **CheckmarkBlock** object.
BT_CheckmarkGroup	Designates a checkmarks group block. It corresponds to an image zone recognized as checkmarks group. Additional properties of the blocks of this type are accessible via the **CheckmarkGroup** object.

BT_VectorPicture	Designates a vector picture block. Blocks of this type may appear in the layout only if a page has been analyzed with the **IPageAnalysisParams::DetectVectorGraphics** property set to TRUE. Usually background pictures are recognized as the blocks of this type. Additional properties of the blocks of this type are accessible via the **VectorPictureBlock** object.
BT_Separator	Designates a separator block. Separators are lines on an image. They may be parts of a table, lines that separate different text elements, etc. Additional properties of the blocks of this type are accessible via the **SeparatorBlock** object.
BT_SeparatorGroup	Designates a separators group block. It corresponds to an image zone recognized as a group of separators. A group of separators usually includes four separators, which form a rectangle. For example, four lines of a table border is recognized as a separators group. Additional properties of the blocks of this type are accessible via the **SeparatorGroup** object.

See also

IBlock::Type
ILayout::AddBlock
ILayout::InsertBlock

See sample: RecognizedTextProcessing

CaptionPositionEnum

CaptionPositionEnum enumeration constants are used to designate the position of the caption relative to the object which has this caption.

```
typedef enum {
      CP_Top,
      CP_Bottom,
      CP_Left,
      CP_Right,
      CP_Inside
} CaptionPositionEnum;
```

Elements

Name	Description
CP_Top	The caption is located above the object.
CP_Bottom	The caption is located below the object.
CP_Left	The caption is located to the left of the object.
CP_Right	The caption is located to the right of the object.
CP_Inside	The caption is crossed with the object.

See also

ICaption::Position
ICaptions::CreateCaption

CaseRecognitionModeEnum

CaseRecognitionModeEnum enumeration constants denote the modes of letter case recognition.

```
typedef enum {
      CRM_AutoCase,
      CRM_SmallCase,
      CRM_CapitalCase
} CaseRecognitionModeEnum;
```

Elements

Name	Description
CRM_AutoCase	This value tells ABBYY FineReader Engine to automatically detect the case of letters and to keep it in the output text.

CRM_SmallCase	The recognized text will be set in lowercase letters.
CRM_CapitalCase	The recognized text will be set in capitals.

See also

IRecognizerParams::CaseRecognitionMode

CheckmarkCheckStateEnum

CheckmarkCheckStateEnum enumeration constants are used to specify the state of a checkmark block.

```
typedef enum {
      CMCS_Checked,
      CMCS_NotChecked,
      CMCS_Corrected
} CheckmarkCheckStateEnum;
```

Elements

Name	Description
CMCS_Checked	Selected.
CMCS_NotChecked	Not selected.
CMCS_Corrected	Checkmark was selected but was corrected later.

See also

ICheckmarkBlock::CheckmarkState

CheckmarkTypeEnum

CheckmarkTypeEnum enumeration constants describe checkmark types.

```
typedef enum {
      CMT_Square,
      CMT_Empty,
      CMT_Custom
} CheckmarkTypeEnum;
```

Elements

Name	Description
CMT_Square	Checkmarks in squares.
CMT_Empty	Checkmarks against an empty background.
CMT_Custom	The checkmark has a non-standard form.

See also

ICheckmarkBlock::CheckmarkType

CJKTextDirectionEnum

Sets the direction of the text to be recognized. This parameter is valid only for the hieroglyphic languages.

```
typedef enum {
      CJKTD_Horizontal,
      CJKTD_Vertical,
      CJKTD_Autodetect
} CJKTextDirectionEnum;
```

Elements

Name	Description
CJKTD_Horizontal	The text to be recognized is arranged horizontally.

CJKTD_Vertical	The text to be recognized is arranged vertically. Characters are written one below the other, top to bottom.
CJKTD_Autodetect	The direction of the text is detected automatically.

See also

IRecognizerParams::CJKTextDirection

CodePageEnum

CodePageEnum enumeration represents Win32 standard code pages.

```
typedef enum {
       CP_Null                = 0,
       CP_Latin               = 1252,
       CP_Cyrillic            = 1251,
       CP_EasternEuropean     = 1250,
       CP_Baltic              = 1257,
       CP_Turkish             = 1254,
       CP_US_MSDOS            = 437,
       CP_LatinI_MSDOS        = 850,
       CP_Russian_MSDOS       = 866,
       CP_Baltic_MSDOS        = 775,
       CP_Turkish_IBM         = 857,
       CP_Slavic_MSDOS        = 852,
       CP_Greek               = 1253,
       CP_Greek_737           = 737,
       CP_Greek_869           = 869,
       CP_Latin_ISO           = 28591,
       CP_EasternEuropean_ISO = 28592,
       CP_Turkish_ISO         = 28593,
       CP_Baltic_ISO          = 28594,
       CP_Cyrillic_ISO        = 28595,
       CP_Greek_ISO           = 28597,
       CP_KOI8                = 20866,
       CP_Tatar               = 5000,
       CP_Tatar_MSDOS         = 5001,
       CP_Roman_Macintosh     = 10000,
       CP_Greek_Macintosh     = 10006,
       CP_Cyrillic_Macintosh  = 10007,
       CP_Ukrainian_Macintosh = 10017,
       CP_Latin2_Macintosh    = 10029,
       CP_Icelandic_Macintosh = 10079,
       CP_Turkish_Macintosh   = 10081,
       CP_Croatian_Macintosh  = 10082,
       CP_Armenian            = 5002,
       CP_Armenian_MSDOS      = 5003,
       CP_Armenian_Macintosh  = 5004,
       CP_Hebrew              = 1255,
       CP_Hebrew_MSDOS        = 862,
       CP_Hebrew_Macintosh    = 10005,
       CP_Hebrew_ISO          = 28598,
       CP_Latin5_ISO          = 28599,
       CP_Cyrillic_MSDOS      = 855,
       CP_Bashkir             = 5006,
       CP_Chinese_Simpl_GB    = 936,
       CP_Chinese_Simpl_Mac   = 10008,
       CP_Chinese_Trad_Big    = 950,
       CP_Chinese_Trad_Mac    = 10002,
       CP_Japan_Mac           = 10001,
       CP_Japan_SJIS          = 932,
       CP_Korean              = 949,
       CP_Korean_Johab        = 1361,
       CP_Korean_Mac          = 10003,
       CP_Mathematical        = 5007,
       CP_Digits              = 5008,
       CP_Thai                = 874,
       CP_Thai_Macintosh      = 10021,
       CP_Vietnamese          = 1258
} CodePageEnum;
```

Elements

Name	Description
CP_Null	Invalid code page.
CP_Latin	Windows Western Europe (1252)
CP_Cyrillic	Windows Cyrillic (1251)
CP_EasternEuropean	Windows Central Europe (1250)
CP_Baltic	Windows Baltic (1257)
CP_Turkish	Windows Turkish (1254)
CP_US_MSDOS	DOS United States (437)
CP_LatinI_MSDOS	DOS Multilingual Latin 1 (850)
CP_Russian_MSDOS	DOS Russian (866)
CP_Baltic_MSDOS	DOS Baltic (775)
CP_Turkish_IBM	DOS Turkish (857)
CP_Slavic_MSDOS	DOS Latin 2 (852)
CP_Greek	Windows Greek (1253)
CP_Greek_737	DOS Greek (737)
CP_Greek_869	DOS Modern Greek (869)
CP_Latin_ISO	ISO Latin 1 (8859-1)
CP_EasternEuropean_ISO	ISO Central Europe (8859-2)
CP_Turkish_ISO	ISO Latin 3 (8859-3)
CP_Baltic_ISO	ISO Baltic (8859-4)
CP_Cyrillic_ISO	ISO Cyrillic (8859-5)
CP_Greek_ISO	ISO Greek (8859-7)
CP_KOI8	Russian KOI8
CP_Tatar	Windows Tatar
CP_Tatar_MSDOS	DOS Tatar
CP_Roman_Macintosh	Macintosh Roman
CP_Greek_Macintosh	Macintosh Greek 1
CP_Cyrillic_Macintosh	Macintosh Cyrillic
CP_Ukrainian_Macintosh	Macintosh Ukrainian
CP_Latin2_Macintosh	Macintosh Latin 2

CP_Icelandic_Macintosh	Macintosh Icelandic
CP_Turkish_Macintosh	Macintosh Turkish
CP_Croatian_Macintosh	Macintosh Croatian
CP_Armenian	Windows Armenian
CP_Armenian_MSDOS	DOS Armenian
CP_Armenian_Macintosh	Macintosh Armenian
CP_Hebrew	Windows Hebrew (1255)
CP_Hebrew_MSDOS	DOS Hebrew (862)
CP_Hebrew_Macintosh	Macintosh Hebrew
CP_Hebrew_ISO	ISO Hebrew (8859-8)
CP_Latin5_ISO	ISO Turkish (8859-9)
CP_Cyrillic_MSDOS	DOS Cyrillic (855)
CP_Bashkir	Windows Bashkir
CP_Chinese_Simpl_GB	Chinese Simplified (GB2312)
CP_Chinese_Simpl_Mac	Chinese Simplified (Mac)
CP_Chinese_Trad_Big	Chinese Traditional (Big5)
CP_Chinese_Trad_Mac	Chinese Traditional (Mac)
CP_Japan_Mac	Japanese (Mac)
CP_Japan_SJIS	Japanese (Shift-JIS)
CP_Korean	Korean
CP_Korean_Johab	Korean (Johab)
CP_Korean_Mac	Korean (Mac)
CP_Mathematical	Mathematical symbols
CP_Digits	Digits
CP_Thai	Windows Thai (874)
CP_Thai_Macintosh	Macintosh Thai
CP_Vietnamese	Vietnamese

See also

ITextExportParams::CodePage
IHTMLExportParams::CodePage
IBarcodeParams::PDF417CodePage
IPlainText::SaveToTextFile

CorrectSkewModeEnum

CorrectSkewModeEnum enumeration constants are used to describe the type of the skew correction. These constants are bit flags.

```
typedef enum {
      CSM_CorrectSkewByBlackSquaresHorizontally = 1,
      CSM_CorrectSkewByBlackSquaresVertically = 2,
      CSM_CorrectSkewByHorizontalLines = 4,
      CSM_CorrectSkewByVerticalLines = 8,
      CSM_CorrectSkewByHorizontalText = 16,
      CSM_CorrectSkewByVerticalText = 32
} CorrectSkewModeEnum;
```

Elements

Name	Description
CSM_CorrectSkewByBlackSquaresHorizontally	The image skew angle is corrected based on so-called "black squares" (the skew angle is calculated based on the horizontal pairs of squares). Black squares are often placed on forms. It is recommended to use this constant only when working with images of forms, otherwise you may obtain incorrect results.
CSM_CorrectSkewByBlackSquaresVertically	The image skew angle is corrected based on so-called "black squares" (the skew angle is calculated based on the vertical pairs of squares). Black squares are often placed on forms. It is recommended to use this constant only when working with images of forms, otherwise you may obtain incorrect results.
CSM_CorrectSkewByHorizontalLines	The image skew angle is corrected based on horizontal lines.
CSM_CorrectSkewByVerticalLines	The image skew angle is corrected based on vertical lines.
CSM_CorrectSkewByHorizontalText	The image skew angle is corrected based on horizontal text lines.
CSM_CorrectSkewByVerticalText	The image skew angle is corrected based on vertical text lines.

See also

IImageDocument::CorrectSkew
IPrepareImageMode::CorrectSkewMode

DictionaryTypeEnum

DictionaryTypeEnum enumeration constants are used to denote different types of dictionaries.

```
typedef enum {
      DT_SystemDictionary,
      DT_UserDictionary,
      DT_RegularExpression,
      DT_ExternalDictionary
} DictionaryTypeEnum;
```

Elements

Name	Description
DT_SystemDictionary	The standard dictionary. The **StandardDictionaryDescription** provides access to the standard dictionary description.
DT_UserDictionary	The user-defined dictionary. The **UserDictionaryDescription** provides access to the user-defined dictionary description.
DT_RegularExpression	The regular expression-based dictionary. The **RegExpDictionaryDescription** provides access to the regular expression-based dictionary description.
DT_ExternalDictionary	The external dictionary. The **ExternalDictionaryDescription** provides access to the external dictionary description.

See also

IDictionaryDescription::Type

DocumentElementTypeEnum

DocumentElementTypeEnum enumeration constants are used to designate the types of document elements.

```
typedef enum {
      DET_Paragraph,
      DET_Table,
      DET_Picture,
      DET_Barcode
} DocumentElementTypeEnum;
```

Elements

Name	Description
DET_Paragraph	Paragraph.
DET_Table	Table.
DET_Picture	Picture.
DET_Barcode	Barcode.

See also

IDocumentElement::Type

EnhancedImageColorVarietyEnum

EnhancedImageColorVarietyEnum enumeration constants represent the variety of colors on the image.

```
typedef enum {
      EICV_DontKnow,
      EICV_FewColors,
      EICV_ManyColors
} EnhancedImageColorVarietyEnum;
```

Elements

Name	Description
EICV_DontKnow	Unknown.
EICV_FewColors	Relatively few colors. Saving in PNG or TIFF with LZW compression is recommended.
EICV_ManyColors	Quite many colors. Saving in JPEG is recommended.

Note: "Recommended" means that the recommended algorithm will provide the best compression.

See also

IRasterPictureBlock::ColorVariety

ErrorHiliteLevelEnum

ErrorHiliteLevelEnum enumeration constants are used to set the level at which the uncertainly recognized characters will be highlighted in the recognized text, that is the degree of their uncertainty.

```
typedef enum {
      EHL_None,
      EHL_Scanty,
      EHL_Standard,
      EHL_Thorough,
      EHL_AllText
} ErrorHiliteLevelEnum;
```

Elements

Name	Description
EHL_None	Do not highlight uncertain characters at all. This means that no character in the recognized text will have the property **ICharParams::IsSuspicious** set to TRUE.
EHL_Scanty	Highlight only very uncertain characters.
EHL_Standard	Sets standard highlight level. This value is used by default for the **IRecognizerParams::ErrorHiliteLevel** property.
EHL_Thorough	Highlight each character that is uncertain.
EHL_AllText	Highlight all characters in the text.

See also

IRecognizerParams::ErrorHiliteLevel
ICharParams::IsSuspicious

ExportPictureFormatEnum

ExportPictureFormatEnum enumeration constants specify the format or the image compression algorithm to be used during export to various formats. Some enumeration constants may be unavailable for certain formats. For more information, see the description of the **PictureFormat** property of the **RTFExportParams**, **HTMLExportParams**, **PPTExportParams**, **PDFExportParamsOld**, **PDFAExportParamsOld** objects and the **BackgroundFormat**, **ColorMaskFormat**, **TextMaskFormat** properties of the **PDFMRCParams** object.

```
typedef enum {
       EPF_DontSave,
       EPF_Automatic,
       EPF_JpegColor,
       EPF_JpegGray,
       EPF_PngColor,
       EPF_PngGray,
       EPF_PngBlackWhite,
       EPF_ZipColor,
       EPF_ZipGray
       EPF_LZWColor,
       EPF_LZWGray,
       EPF_CCITT4,
       EPF_BmpColor,
       EPF_BmpGray,
       EPF_BmpBlackWhite,
       EPF_J2KColor,
       EPF_J2KGray
       EPF_JBIG2
} ExportPictureFormatEnum;
```

Elements

Name	Description
EPF_DontSave	Images will not be exported.
EPF_Automatic	Format is defined automatically.
EPF_BmpBlackWhite	Black-and-white BMP format.
EPF_BmpColor	Color BMP format.
EPF_BmpGray	Gray BMP format.
EPF_CCITT4	CCITT4 compression algorithm.
EPF_J2KColor	Color JPEG 2000 format.
EPF_J2KGray	Gray JPEG 2000 format.
EPF_JBIG2	JBIG2 compression algorithm.
EPF_JpegColor	Color JPEG format.
EPF_JpegGray	Gray JPEG format.
EPF_LZWColor	LZW compression algorithm will be used during export in color.
EPF_LZWGray	LZW compression algorithm will be used during export in gray.
EPF_PngBlackWhite	Black-and-white PNG format.
EPF_PngColor	Color PNG format.
EPF_PngGray	Gray PNG format.

EPF_ZipColor	ZIP compression algorithm will be used during export in color.
EPF_ZipGray	ZIP compression algorithm will be used during export in gray.

See also

IRTFExportParams::PictureFormat
IHTMLExportParams::PictureFormat
IPPTExportParams::PictureFormat
IPDFMRCParams::BackgroundFormat
IPDFMRCParams::ColorMaskFormat
IPDFMRCParams::TextMaskFormat
IPDFExportParamsOld::PictureFormat
IPDFAExportParamsOld::PictureFormat

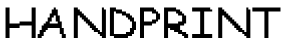
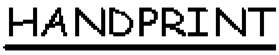
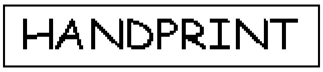
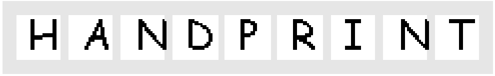
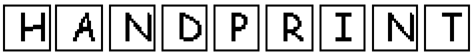
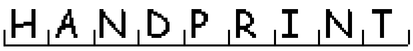
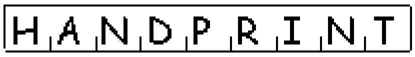
FieldMarkingTypeEnum

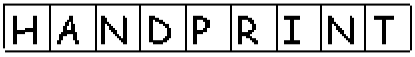
FieldMarkingTypeEnum enumeration constants describe available types of field marking for handprinted text.

Note: The number of character cells for a recognized block you can set with help of the IRecognizerParams::CellsCount property.

```
typedef enum {
      FMT_SimpleText,
      FMT_UnderlinedText,
      FMT_TextInFrame,
      FMT_GreyBoxes,
      FMT_CharBoxSeries,
      FMT_SimpleComb,
      FMT_CombInFrame,
      FMT_PartitionedFrame
} FieldMarkingTypeEnum;
```

Elements

Name	Description
FMT_SimpleText	This value denotes the plain text: 
FMT_UnderlinedText	This value specifies that the text is underlined: 
FMT_TextInFrame	This value specifies that the text is enclosed in a frame: 
FMT_GreyBoxes	This value specifies that the text is located in white fields on a gray background: 
FMT_CharBoxSeries	This value specifies that the field where the text is located is a set of separate boxes: 
FMT_SimpleComb	This value specifies that the field where the text is located is a comb: 
FMT_CombInFrame	This value specifies that the field where the text is located is a comb and that this comb is also the bottom line of a frame: 
FMT_PartitionedFrame	This value specifies that the field where the text is located is a frame and this frame is split by vertical lines:

See also

IRecognizerParams::FieldMarkingType
Recognizing Handprinted Texts

FileExportFormatEnum

FileExportFormatEnum enumeration constants define different file formats in which ABBYY FineReader Engine can save the recognized text.

```
typedef enum {
      FEF_RTF,
      FEF_HTML,
      FEF_XLS,
      FEF_PDF,
      FEF_Text,
      FEF_XML,
      FEF_PDFA,
      FEF_DOCX,
      FEF_XLSX,
      FEF_PPTX
} FileExportFormatEnum;
```

Elements

Name	Description
FEF_RTF	Microsoft RTF/DOC format. The parameters of the file in this format are tuned through the **RTFExportParams** object.
FEF_HTML	HTML/Unicode HTML format. The parameters of a file in this format are tuned through the **HTMLExportParams** object.
FEF_XLS	XLS (Microsoft Excel) format. The parameters of a file in this format are tuned through the **XLExportParams** object.
FEF_PDF	PDF or PDF/A format. The parameters of a file in this format are tuned through the **PDFExportParams** object.
FEF_Text	TXT/Unicode TXT or CSV/Unicode CSV format. The parameters of a file in this format are tuned through the **TextExportParams** object.
FEF_XML	XML format. The parameters of a file in this format are tuned through the **XMLExportParams** object.
FEF_PDFA	This constant is obsolete, use the FEF_PDF instead. The constant defines the PDF/A format. The parameters of a file in this format are tuned through the **PDFAExportParamsOld** object.
FEF_DOCX	DOCX (Microsoft Word 2007) format. The parameters of the file in this format are tuned through the **RTFExportParams** object.
FEF_XLSX	XLSX (Microsoft Excel 2007) format. The parameters of a file in this format are tuned through the **XLExportParams** object.
FEF_PPTX	PPTX (Microsoft PowerPoint 2007) format. The parameters of a file in this format are tuned through the **PPTExportParams** object.

See also

IFRDocument::Export
IFRDocument::ExportPages
IFRPage::Export
IEngine::ExportPage
IEngine::ExportPages
IEngine::RecognizeImageFile
IExporter::ExportPages
IExporter::ExportPagesEx

See sample: Hello

FontModeEnum

FontModeEnum enumeration constants set the mode of font usage during export of recognized text into PDF format.

```
typedef enum {
	FM_UseStandardFonts,
	FM_UseFontsFromIText
} FontModeEnum;
```

Elements

Name	Description
FM_UseStandardFonts	The PDF file refers to the standard system fonts Times, Helvetica and CourierNew. If the **IPDFExportParamsOld::EmbedFonts** property is set to TRUE, for a non-Latin code page (e.g., Cyrillic, Greek, Czech, etc.) ABBYY FineReader will embed the fonts provided by ParaType (www.paratype.com/shop), and for a Latin code page fonts will not be embedded and ABBYY FineReader will create references to the standard system fonts Times, Helvetica and CourierNew. If the **IPDFExportParamsOld::EmbedFonts** property is set to FALSE, ABBYY FineReader will create references to the standard system fonts Times, Helvetica and CourierNew for all code pages.
FM_UseFontsFromIText	During export, the names of the fonts are taken from the **Text** object, which represents the recognized text. System fonts are used in this case, therefore the fonts saved in the **Text** object must be installed on the system.

See also

IPDFExportParamsOld::FontMode

FontTypeEnum

FontTypeEnum enumeration constants are used to denote different types of fonts.

```
typedef enum {
	FT_Serif,
	FT_SansSerif,
	FT_MonoSpace,
	FT_Decorative,
	FT_Unknown
} FontTypeEnum;
```

Elements

Name	Description
FT_Serif	Serif font (e.g. Times New Roman).
FT_SansSerif	Sans Serif font (e.g. Arial).
FT_MonoSpace	Monospace font (e.g. Courier).
FT_Decorative	Decorative or handprinted font.
FT_Unknown	Font type is undefined.

See also

ICharParams::FontType
ICharParams::SetFont
IFontStyle::FontType

FootnotePositionOnPageTypeEnum

FootnotePositionOnPageTypeEnum enumeration constants are used to designate the position of a footnote relative to its anchor.

```
typedef enum {
  FPPT_LastColumn,
  FPPT_CurrentColumn,
  FPPT_SingleColumnSection
 } FootnotePositionOnPageTypeEnum;
```

Elements

Name	Description
FPPT_LastColumn	At the end of the last column on the page.
FPPT_CurrentColumn	In the same column with anchor.
FPPT_SingleColumnSection	In the single-column section.

See also

IFootnoteSeries::PositionOnPage
IFootnoteSeries::SetPosition

FootnoteNumberingTypeEnum

FootnoteNumberingTypeEnum enumeration constants are used to designates numbering types of footnotes.

```
typedef enum {
      FNT_1,
      FNT_I_capital,
      FNT_i_small,
      FNT_A_capital,
      FNT_a_small,
      FNT_Asterisk,
      FNT_AsteriskOnly
} FootnoteNumberingTypeEnum;
```

Elements

Name	Description
FNT_1	Decimal numbering. For example, 1, 2, 3, 4, 5, ...
FNT_I_capital	Uppercase Roman numerals. For example, I, II, III, IV, V, ...
FNT_i_small	Lowercase Roman numerals. For example, i, ii, iii, iv, v, ...
FNT_A_capital	Uppercase letters of the Latin alphabet. For example, A, B, C, D, E, ...
FNT_a_small	Lowercase letters of the Latin alphabet. For example, a, b, c, d, e, ...
FNT_Asterisk	Characters as defined in the Chicago Manual of Style. For example, *, †, ‡, §, **, ...
FNT_AsteriskOnly	Only asterisks. For example, *, **, ***, ****, ...

See also

IFootnoteSeries::NumberingType

FootnotePositionInDocumentTypeEnum

FootnotePositionInDocumentTypeEnum enumeration constants are used to designate the different types of footnote positions.

```
typedef enum {
      FPDT_TextEnd,
      FPDT_PageEnd,
      FPDT_SectionEnd,
      FPDT_DocumentEnd
} FootnotePositionInDocumentTypeEnum;
```

Elements

Name	Description
FPDT_TextEnd	At the end of text on the same page.
FPDT_PageEnd	At the end of page.
FPDT_SectionEnd	At the end of section (may be on another page).
FPDT_DocumentEnd	At the end of document.

See also

IFootnoteSeries::PositionInDocument
IFootnoteSeries::SetPosition

FrameHorizontalReferenceEnum

FrameHorizontalReferenceEnum enumeration constants designate different types of objects on the page to measure horizontal offset from. Horizontal offset is generally measured from the left border of the object for texts with left-to-right writing direction, and from the right border — for texts with right-to-left writing direction.

```
typedef enum {
      FHR_Margin,
      FHR_Page
} FrameHorizontalReferenceEnum;
```

Elements

Name	Description
FHR_Margin	The offset is measured from the left (or right — for right-to-left texts) margin of the page.
FHR_Page	The offset is measured from the left (or right — for right-to-left texts) border of the page.

See also

IIncut::HorizontalOffset

FrameVerticalReferenceEnum

FrameVerticalReferenceEnum enumeration constants designate different types of objects on the page to measure vertical offset from.

```
typedef enum {
      FVR_Page,
      FVR_Margin,
      FVR_Section,
      FVR_Paragraph
} FrameVerticalReferenceEnum;
```

Elements

Name	Description
FVR_Page	The offset is measured from the top of the page.
FVR_Margin	The offset is measured from the top margin of the page.
FVR_Section	The offset is measured from the top of the section on this page.
FVR_Paragraph	The offset is measured from the first line of the paragraph.

See also

IIncut::VerticalOffset

FREngineModuleEnum

FREngineModuleEnum enumeration constants are used to denote loaded modules.

```
typedef enum {
      FREM_ImageSupport,
      FREM_Export,
      FREM_DocumentAnalyzer,
      FREM_Recognizer,
      FREM_RecognizerHP,
      FREM_PDF,
      FREM_FREngineProcessor,
      FREM_ChineseTraditionalPatterns,
      FREM_ChineseSimplifiedPatterns,
      FREM_JapanesePatterns,
      FREM_KoreanPatterns,
      FREM_EuropeanPatterns
} FREngineModuleEnum;
```

Elements

Name	Description
FREM_ImageSupport	Specifies *Image Support* module. **Note:** The Image Support module is loaded during the creation of the **Engine** object. Attempts to load this module with the help of the **LoadModule** method will be ignored. This constant has been retained for backward compatibility.
FREM_Export	Specifies *Export* module.
FREM_DocumentAnalyzer	Specifies *Document Analyzer* module.
FREM_Recognizer	Specifies *Recognizer* module.
FREM_RecognizerHP	Specifies *RecognizerHP* module.
FREM_PDF	Specifies *PDF* module. **Note:** The PDF module is loaded during the creation of the **Engine** object. Attempts to load this module with the help of the **LoadModule** method will be ignored. This constant has been retained for backward compatibility.
FREM_FREngineProcessor	Specifies FineReader Engine Processor module.
FREM_ChineseTraditionalPatterns	Specifies Chinese Traditional Patterns module.
FREM_ChineseSimplifiedPatterns	Specifies Chinese Simplified Patterns module.
FREM_JapanesePatterns	Specifies *Japanese Patterns* module.
FREM_KoreanPatterns	Specifies *Korean Patterns* module.
FREM_EuropeanPatterns	Specifies *European Patterns* module.

See also

IEngine::LoadModule

HTMLFormatModeEnum

HTMLFormatModeEnum enumeration constants are used to specify the language version used for export to HTML format.

```
typedef enum {
      HFM_Format32,
      HFM_Format40
} HTMLFormatModeEnum;
```

Elements

Name	Description
HFM_Format32	Simple format using HTML 3.2 standard. Almost all browsers support this format (Netscape Navigator, Internet Explorer 3.0 and later). Not all document layout is retained: first-line indent and indents in tables are not retained.
HFM_Format40	Full format using HTML 4.0 standard. It supports all types of document layout retention. It requires Internet Explorer 4.0 or later. A built-in style sheet (CSS) is used.

See also

IHTMLExportParams::HTMLFormatMode

HTMLDocumentSplittingModeEnum

HTMLDocumentSplittingModeEnum enumeration constants are used to denote the mode of splitting HTML document into files.

```
typedef enum {
      HDSM_None,
      HDSM_Heading_1,
      HDSM_Heading_2,
      HDSM_Smart
} HTMLDocumentSplittingModeEnum;
```

Elements

Name	Description
HDSM_None	Do not split file. One output HTML file corresponds to one input file.
HDSM_Heading_1	Split into files by headings of one level.
HDSM_Heading_2	Split into files by headings of two levels.
HDSM_Smart	Smart mode. The program takes into account headings and controls the output files length in order the file length does not exceed some value.

See also

IHTMLExportParams::SplitDocumentToFiles

HTMLSynthesisModeEnum

HTMLSynthesisModeEnum enumeration constants are used to denote available modes of synthesizing HTML code from the recognized text.

```
typedef enum {
      HSM_PlainText,
      HSM_FormattedStream,
      HSM_FlexibleLayout
} HTMLSynthesisModeEnum;
```

Elements

Name	Description
HSM_PlainText	Only paragraphs are retained in the recognized text with the use of the **<p>** tag.
HSM_FormattedStream	Paragraphs and fonts of the recognized text are retained in the output HTML file. The **<p>** tag is used.
HSM_FlexibleLayout	Logical structure of the document is retained in the output HTML file.

See also

IHTMLExportParams::HTMLSynthesisMode

HyperlinkSchemeEnum

HyperlinkSchemeEnum enumeration constants are used to denote different types of hyperlinks.

```
typedef enum {
      HS_Unknown,
      HS_Local,
      HS_Ftp,
      HS_Gopher,
      HS_Http,
      HS_Https,
      HS_File,
      HS_News,
      HS_Mailto
} HyperlinkSchemeEnum;
```

Elements

Name	Description
HS_Unknown	The type of hyperlink is defined automatically.

HS_Local	A local hyperlink to a text fragment in the same document.
HS_Ftp	The FTP site address.
HS_Gopher	The Gopher server address.
HS_Http	The web site address.
HS_Https	The HTTPS web site address.
HS_File	The full path to the file.
HS_News	The full address to a news group.
HS_Mailto	The e-mail address.

See also

IHyperlink::ParseTarget
IHyperlink::Scheme

ImageColorTypeEnum

ImageColorTypeEnum enumeration constants are used to describe different color types of an image.

```
typedef enum {
      ICT_BlackWhite,
      ICT_Gray,
      ICT_Color
} ImageColorTypeEnum;
```

Elements

Name	Description
ICT_BlackWhite	Black-and-white image.
ICT_Gray	Gray image.
ICT_Color	Color image.

See also

IImageDocument::ImageColorType
IImage::ImageColorType
IRasterPictureBlock::ColorType

ImageCompressionEnum

ImageCompressionEnum enumeration constants are used to set the image compression type for temporary image files. Compression can be applied to color and gray images only. ZIP compression is used.

```
typedef enum {
      IC_NoCompression,
      IC_Compress,
      IC_Auto
} ImageCompressionEnum;
```

Elements

Name	Description
IC_NoCompression	Uncompressed. This compression type provides the largest image file size, the quality of the original image, and the least processing time.
IC_Compress	This compression type provides the least image file size and the worst image quality.
IC_Auto	Lossless compression. This compression type provides a medium image file size and the quality of the original image.

See also

IPrepareImageMode::ImageCompression

ImageFileFormatEnum

ImageFileFormatEnum enumeration constants are used to specify the format of the image file that can be read or written by means of ABBYY FineReader Engine. ABBYY FineReader Engine can open image files in all formats described by these enumeration constants, but not all formats are supported for writing.

```
typedef enum {
      IFF_UnknownFormat,
      IFF_BmpBwUncompressed,
      IFF_BmpGrayUncompressed,
      IFF_BmpColorUncompressed,
      IFF_DcxBwPackbits,
      IFF_DcxGrayPackbits,
      IFF_DcxColorPackbits,
      IFF_JpegGrayJfif,
      IFF_JpegColorJfif,
      IFF_PcxBwPackbits,
      IFF_PcxGrayPackbits,
      IFF_PcxColorPackbits,
      IFF_PngBwPng,
      IFF_PngGrayPng,
      IFF_PngColorPng,
      IFF_TiffBwUncompressed,
      IFF_TiffBwCcittGroup3,
      IFF_TiffBwCcittGroup3Fax,
      IFF_TiffBwCcittGroup4,
      IFF_TiffBwPackbits,
      IFF_TiffGrayUncompressed,
      IFF_TiffGrayPackbits,
      IFF_TiffGrayJpegJfif,
      IFF_TiffColorUncompressed,
      IFF_TiffColorPackbits,
      IFF_TiffColorJpegJfif,
      IFF_TiffGrayABBYYLossless,
      IFF_TiffColorABBYYLossless,
      IFF_Jpeg2kGray,
      IFF_Jpeg2kColor,
      IFF_PDF,
      IFF_TiffBwLZW,
      IFF_TiffGrayLZW,
      IFF_TiffColorLZW,
      IFF_TiffBwZip,
      IFF_TiffGrayZip,
      IFF_TiffColorZip,
      IFF_GifBwLZW,
      IFF_GifGrayLZW,
      IFF_GifColorLZW,
      IFF_DjVuBw,
      IFF_DjVuGray,
      IFF_DjVuColor,
      IFF_JBIG2,
      IFF_WdpBw,
       IFF_WdpGray,
      IFF_WdpColor,
      IFF_Wic
} ImageFileFormatEnum;
```

Elements

Name	Description	Supported for reading	Supported for writing
IFF_UnknownFormat	This value specifies unknown format. May only appear as the return value.		
IFF_BmpBwUncompressed	Black–and–white uncompressed BMP.	+	+
IFF_BmpGrayUncompressed	Gray uncompressed BMP.	+	+
IFF_BmpColorUncompressed	Color uncompressed BMP.	+	+
IFF_DcxBwPackbits	Black–and–white DCX.	+	+
IFF_DcxGrayPackbits	Gray DCX.	+	+
IFF_DcxColorPackbits	Color DCX.	+	+
IFF_JpegGrayJfif	Gray JPEG (JFIF fomat).	+	+
IFF_JpegColorJfif	Color JPEG (JFIF fomat).	+	+
IFF_PcxBwPackbits	Black–and–white PCX.	+	+
IFF_PcxGrayPackbits	Gray PCX.	+	+
IFF_PcxColorPackbits	Color PCX.	+	+
IFF_PngBwPng	Black–and–white PNG.	+	+
IFF_PngGrayPng	Gray PNG.	+	+
IFF_PngColorPng	Color PNG.	+	+
IFF_TiffBwUncompressed	Black–and–white uncompressed TIFF.	+	+
IFF_TiffBwCcittGroup3	Black–and–white TIFF, GROUP3 compressed.	+	+
IFF_TiffBwCcittGroup3Fax	Black–and–white TIFF, GROUP3FAX compressed.	+	+
IFF_TiffBwCcittGroup4	Black–and–white TIFF, GROUP4 compressed.	+	+
IFF_TiffBwPackbits	Black–and–white TIFF, PACKBITS compressed.	+	+
IFF_TiffGrayUncompressed	Gray uncompressed TIFF.	+	+
IFF_TiffGrayPackbits	Gray TIFF, PACKBITS compressed.	+	+
IFF_TiffGrayJpegJfif	Gray TIFF, JPEG(JFIF) compressed.	+	+
IFF_TiffColorUncompressed	Color uncompressed TIFF.	+	+
IFF_TiffColorPackbits	Color TIFF, PACKBITS compressed.	+	+
IFF_TiffColorJpegJfif	Color TIFF, JPEG(JFIF) compressed.	+	+
IFF_TiffGrayABBYYLossless	Gray TIFF, ABBYYLossless compressed.	+	
IFF_TiffColorABBYYLossless	Color TIFF, ABBYYLossless compressed.	+	
IFF_Jpeg2kGray	Gray JPEG 2000.	+	+
IFF_Jpeg2kColor	Color JPEG 2000.	+	+
IFF_PDF	PDF.	+	+
IFF_TiffBwLZW	Black–and–white TIFF, LZW–compressed.	+	+
IFF_TiffGrayLZW	Gray TIFF, LZW–compressed.	+	+
IFF_TiffColorLZW	Color TIFF, LZW–compressed.	+	+

IFF_TiffBwZip	Black–and–white TIFF, ZIP–compressed.	+	+
IFF_TiffGrayZip	Gray TIFF, ZIP–compressed.	+	+
IFF_TiffColorZip	Color TIFF, ZIP–compressed.	+	+
IFF_GifBwLZW	Black–and–white GIF, LZW–compressed.	+	
IFF_GifGrayLZW	Gray GIF, LZW–compressed.	+	
IFF_GifColorLZW	Color GIF, LZW–compressed.	+	
IFF_DjVuBw	Black–and–white DjVu.	+	
IFF_DjVuGray	Gray DjVu.	+	
IFF_DjVuColor	Color DjVu.	+	
IFF_JBIG2	JBIG2.	+	+
IFF_WdpBw	Black-and-white WDP.	+	
IFF_WdpGray	Gray WDP.	+	
IFF_WdpColor	Color WDP.	+	
IFF_Wic	WIC.	+	

See also

IEngine::CreateMultipageImageWriter
IImageDocument::SourceImageFileFormat
IImage::WriteToFile
Supported Image Formats

ImageTypeEnum

ImageTypeEnum enumeration constants are used to convert coordinates between different image planes of the **ImageDocument** object. The latter object represents an open image. The open image contains only one page for each color type (black-and-white or color). It is either deskewed or non-deskewed depending on the internal file preparation mode (see the description of the **PrepareImageMode** object).

```
typedef enum {
	IT_Base,
	IT_Deskewed,
	IT_Preview
} ImageTypeEnum;
```

Elements

Name	Description
IT_Base	Non-deskewed image.
IT_Deskewed	Fully deskewed image.
IT_Preview	Preview image. 📝 **Note:** An open image contains this image plane, only if **IPrepareImageMode::CreatePreview** property was set to TRUE during image preparation.

See also

ImageDocument
IImageDocument::ConvertCoordinates

LanguageCategoryEnum

LanguageCategoryEnum enumeration constants are used to describe the category of a predefined ABBYY FineReader Engine language.

```
typedef enum {
	LC_CoreLanguage,
	LC_AdditionalLanguage,
      LC_ConstructedLanguage,
      LC_FormalLanguage
} LanguageCategoryEnum;
```

Elements

Name	Description
LC_CoreLanguage	A widespread language, such as English or Russian.
LC_AdditionalLanguage	Additional languages, not so widely used as core languages. Examples are Afrikaans or Albanian.
LC_ConstructedLanguage	An artificial language such as Esperanto or Interlingua.
LC_FormalLanguage	Programming language or another formal language. For example, Basic and C/C++ languages belong to this category.

See also

IPredefinedLanguage::LanguageCategory

LanguageIdEnum

LanguageIdEnum enumeration represents Win32 standard language identifier (data type LANGID). It may be converted into the standard Win32 LCID by calling the **IEngine::ConvertLanguageIdToLCID** method.

```
typedef enum {
      LI_Null = 0,
      LI_EnglishUnitedStates = 1033,
      LI_EnglishUnitedKingdom = 2057,
      LI_EnglishAustralian = 3081,
      LI_EnglishCanadian = 4105,
      LI_EnglishNewZealand = 5129,
      LI_EnglishIreland = 6153,
      LI_EnglishSouthAfrica = 7177,
      LI_EnglishJamaica = 8201,
      LI_EnglishCaribbean = 9225,
      LI_EnglishBelize = 10249,
      LI_EnglishTrinidad = 11273,
      LI_Bulgarian = 1026,
      LI_Czech = 1029,
      LI_Danish = 1030,
      LI_GermanStandard = 1031,
      LI_GermanSwiss = 2055,
      LI_GermanAustrian = 3079,
      LI_GermanLuxembourg = 4103,
      LI_GermanLiechtenstein = 5127,
      LI_Greek = 1032,
      LI_SpanishTraditionalSort = 1034,
      LI_SpanishMexican = 2058,
      LI_SpanishModernSort = 3082,
      LI_SpanishGuatemala = 4106,
      LI_SpanishCostaRica = 5130,
      LI_SpanishPanama = 6154,
      LI_SpanishDominicanRepublic = 7178,
      LI_SpanishVenezuela = 8202,
      LI_SpanishColombia = 9226,
      LI_SpanishPeru = 10250,
      LI_SpanishArgentina = 11274,
      LI_SpanishEcuador = 12298,
      LI_SpanishChile = 13322,
      LI_SpanishUruguay = 14346,
      LI_SpanishParaguay = 15370,
      LI_SpanishBolivia = 16394,
      LI_SpanishElSalvador = 17418,
      LI_SpanishHonduras = 18442,
      LI_SpanishNicaragua = 19466,
      LI_SpanishPuertoRico = 20490,
      LI_Finnish = 1035,
      LI_FrenchStandard = 1036,
      LI_FrenchBelgian = 2060,
      LI_FrenchCanadian = 3084,
      LI_FrenchSwiss = 4108,
      LI_FrenchLuxembourg = 5132,
      LI_Hungarian = 1038,
      LI_Icelandic = 1039,
      LI_ItalianStandard = 1040,
      LI_ItalianSwiss = 2064,
      LI_DutchStandard = 1043,
      LI_DutchBelgian = 2067,
      LI_NorwegianBokmal = 1044,
      LI_NorwegianNynorsk = 2068,
      LI_Polish = 1045,
      LI_PortugueseBrazilian = 1046,
      LI_PortugueseStandard = 2070,
      LI_Romanian = 1048,
      LI_Russian = 1049,
      LI_Croatian = 1050,
      LI_SerbianLatin = 2074,
      LI_SerbianCyrillic = 3098,
      LI_Slovak = 1051,
      LI_Swedish = 1053,
      LI_SwedishFinland = 2077,
      LI_Turkish = 1055,
      LI_Slovenian = 1060,
      LI_Afrikaans = 1078,
      LI_Albanian = 1052,
      LI_Basque = 1069,
      LI_Belarusian = 1059,
      LI_Catalan = 1027,
      LI_Estonian = 1061,
      LI_Faeroese = 1080,
      LI_Indonesian = 1057,
      LI_Latvian = 1062,
      LI_Lithuanian = 1063,
      LI_Ukrainian = 1058,
      LI_Japanese = 1041,
      LI_Korean = 1042,
      LI_KoreanJohab = 2066,
      LI_ChinesePRC = 2052,
      LI_ChineseSingapore = 4100,
      LI_Thai = 1054,
      LI_ChineseTaiwan = 1028,
      LI_ChineseHongKong = 3076,
      LI_Vietnamese = 1066,
      LI_Hebrew = 1037,
      LI_Macedonian = 1071,
      LI_Swahili = 1089,
      LI_Tatar = 1092,
      LI_Irish = 1552,
      LI_Tagalog = 1553,
      LI_User = 1554,
      LI_MalayMalaysian = 1086,
      LI_MalayBruneiDarussalam = 2110,
      LI_Maori = 1064,
      LI_RomanianMoldavia = 2072,
      LI_RhaetoRomanic = 1047,
      LI_Breton = 1536,
      LI_Esperanto = 1537,
      LI_Fijian = 1538,
      LI_Hawaiian = 1539,
      LI_Latin = 1540,
      LI_Provencal = 1541,
      LI_Samoan = 1542,
      LI_Welsh = 1543,
      LI_Chechen = 1544,
      LI_CrimeanTatar = 1546,
      LI_Mongol = 1104,
      LI_Ossetic = 1547,
      LI_Kabardian = 1548,
      LI_Yiddish = 1077,
      LI_ArmenianEastern = 1067,
      LI_ArmenianWestern = 32811,
      LI_ArmenianGrabar = 33835,
      LI_GermanNewSpelling = 32775,
      LI_RussianOldSpelling = 32793,
      LI_AzeriCyrillic = 2092,
      LI_AzeriLatin = 1068,
      LI_ChineseMacau = 5124,
      LI_EnglishPhilippines = 13321,
      LI_EnglishZimbabwe = 12297,
      LI_FrenchMonaco = 6156,
      LI_GaelicScottish = 1084,
      LI_Kazakh = 1087,
      LI_Lappish = 1083,
      LI_LithuanianClassic = 2087,
      LI_Maltese = 1082,
      LI_RussianMoldavia = 2073,
      LI_Sorbian = 1070,
      LI_Tswana = 1074,
      LI_UzbekCyrillic = 2115,
      LI_UzbekLatin = 1091,
      LI_Xhosa = 1076,
      LI_Zulu = 1077,
      LI_Abkhaz = 1556,
      LI_Adyghe = 1557,
      LI_Awar = 1558,
      LI_Agul = 1559,
      LI_Altaic = 1545,
      LI_Aymara = 1560,
      LI_Bashkir = 1561,
      LI_Bemba = 1562,
      LI_Blackfoot = 1563,
      LI_Bugotu = 1564,
      LI_Buryat = 1565,
      LI_Chamorro = 1566,
      LI_Chukcha = 1567,
      LI_Chuvash = 1568,
      LI_Corsican = 1569,
      LI_Crow = 1570,
      LI_Dargwa = 1571,
      LI_Dungan = 1572,
      LI_EskimoCyrillic = 1573,
      LI_Even = 1574,
      LI_Evenki = 1575,
      LI_Frisian = 1576,
      LI_Friulian = 1577,
      LI_Gagauz = 1578,
      LI_Galician = 1579,
      LI_Ganda = 1580,
      LI_EskimoLatin = 1581,
      LI_Guarani = 1582,
      LI_Hani = 1583,
      LI_Ido = 1584,
      LI_Ingush = 1585,
      LI_Interlingua = 1586,
      LI_Kalmyk = 1587,
      LI_Karakalpak = 1588,
      LI_KarachayBalkar = 1589,
      LI_Kasub = 1590,
      LI_Kawa = 1591,
      LI_Khakas = 1592,
      LI_Khanty = 1593,
      LI_Kikuyu = 1594,
      LI_Kirgiz = 1595,
      LI_Kongo = 1598,
      LI_Koryak = 1599,
      LI_Kpelle = 1600,
      LI_Kumyk = 1601,
      LI_Kurdish = 1602,
      LI_Lak = 1604,
      LI_Lezgin = 1605,
      LI_Luba = 1606,
      LI_Malagasy = 1607,
      LI_Malinke = 1608,
      LI_Mansi = 1609,
      LI_Mari = 1610,
      LI_Maya = 1611,
      LI_Miao = 1612,
      LI_Minankabaw = 1613,
      LI_Mohawk = 1614,
      LI_Mordvin = 1615,
      LI_Nahuatl = 1616,
      LI_Nenets = 1618,
      LI_Nivkh = 1619,
      LI_Nogay = 1620,
      LI_Nyanja = 1621,
      LI_Occidental = 1622,
      LI_Ojibway = 1623,
      LI_Papiamento = 1624,
      LI_PidginEnglish = 1625,
      LI_Quechua = 1626,
      LI_Romany = 1627,
      LI_Ruanda = 1628,
      LI_Rundi = 1629,
      LI_Selkup = 1630,
      LI_Shona = 1631,
      LI_Sioux = 1632,
      LI_Somali = 1633,
      LI_Sotho = 1634,
      LI_Sunda = 1635,
      LI_Swazi = 1636,
      LI_Tabassaran = 1637,
      LI_Tajik = 1638,
      LI_Tahitian = 1639,
      LI_Tinpo = 1640,
      LI_Tongan = 1641,
      LI_Tun = 1642,
      LI_Turkmen = 1643,
      LI_Tuvin = 1644,
      LI_Udmurt = 1645,
      LI_UighurCyrillic = 1646,
      LI_Visayan = 1648,
      LI_Wolof = 1649,
      LI_Yakut = 1650,
      LI_Zapotec = 1651,
      LI_Hausa = 1652,
      LI_OldEnglish = 32777,
      LI_OldGerman = 33799,
      LI_OldFrench = 32780,
      LI_OldItalian = 32784,
      LI_OldSpanish = 32778,
      LI_EnglishLaw = 35849,
      LI_GermanLaw = 34823,
      LI_GermanNewSpellingLaw = 35847,
      LI_EnglishMedical = 33801,
      LI_GermanMedical = 36871,
      LI_GermanNewSpellingMedical = 37895,
      LI_UighurLatin = 1647,
      LI_LatvianGothic = 1655
} LanguageIdEnum;
```

See also

IEngine::CreateNewDictionary
IEngine::ConvertLanguageIdToLCID
IEngine::ConvertLCIDToLanguageId
IBaseLanguage::LanguageId
ICharParams::LanguageId
IRecognizerParams::WritingStyle

LicenseCounterTypeEnum

LicenseCounterTypeEnum enumeration constants are used to denote the units (pages, characters) used by the ABBYY FineReader Engine license to limit the number of the recognition and export operations during a period.

```
typedef enum {
      LCT_Pages,
      LCT_Characters,
      LCT_FineReaderXIXPages,
      LCT_FineReaderXIXCharacters
} LicenseCounterTypeEnum;
```

Elements

Name	Description
LCT_Pages	The limitation counts pages.
LCT_Characters	The limitation counts characters.
LCT_FineReaderXIXPages	The limitation counts pages of the FineReader XIX module.
LCT_FineReaderXIXCharacters	The limitation counts characters of the FineReader XIX module.

See also

ILicense::VolumeRefreshingPeriod
ILicense::VolumeRemaining
ILicense::Volume

MemoryImageFormatEnum

MemoryImageFormatEnum enumeration constants describe formats of the memory images that ABBYY FineReader Engine can work with.

```
typedef enum {
      MIF_BlackAndWhite,
      MIF_Gray,
      MIF_Color
} MemoryImageFormatEnum;
```

Elements

Name	Description
MIF_BlackAndWhite	Black-and-white memory image file. One pixel is represented by one bit in memory. 0 corresponds to white color, 1 corresponds to black color.
MIF_Gray	Gray memory image file. One pixel is represented by one byte in memory. Thus, this image file format may have 255 shades of gray. 0 corresponds to white color, 255 corresponds to black color.
MIF_Color	Color memory image file. One pixel is represented by three bytes in memory. Each byte corresponds to one of the basic colors (red, green or blue). The value of 0 for a byte corresponds to the minimum color intensity, and the value of 255 corresponds to the maximum color intensity.

See also

IEngine::OpenMemoryImage
IEngine::PrepareMemoryImage
IEngine::PrepareAndOpenMemoryImage
Memory image format description

MessagesLanguageEnum

MessagesLanguageEnum enumeration constants describe different interface languages that ABBYY FineReader Engine supports. Some languages defined by this enumeration are not currently supported, and the specific set of languages that are supported by your system depends on the availability of resource modules. If you try to set a messages language that is not supported by ABBYY FineReader Engine, it will automatically be changed to the language with the lowest code available.

Note: The locale for the selected messages language must be installed on the computer.

```
typedef enum {
      ML_English = 0,
      ML_Russian = 1,
      ML_German = 2,
      ML_French = 3,
      ML_Ukrainian = 4,
      ML_Spanish = 5,
      ML_Italian = 6,
      ML_DutchStandard = 7,
      ML_Danish = 8,
      ML_Swedish = 9,
      ML_Slovak = 14,
      ML_Polish = 15,
      ML_Czech = 16,
      ML_Hungarian = 17,
      ML_Lithuanian = 18,
      ML_Estonian = 20,
      ML_Bulgarian = 23,
      ML_Turkish = 24,
      ML_PortugueseBrazilian = 27,
      ML_Korean = 63,
      ML_ChinesePRC = 64,
      ML_ChineseTaiwan = 65
} MessagesLanguageEnum;
```

Elements

Name	Description
ML_English	English language
ML_Russian	Russian language
ML_German	German language
ML_French	French language
ML_Ukrainian	Ukrainian language
ML_Spanish	Spanish language
ML_Italian	Italian language
ML_DutchStandard	Dutch language
ML_Danish	Danish language
ML_Swedish	Swedish language
ML_Slovak	Slovak language
ML_Polish	Polish language
ML_Czech	Czech language
ML_Hungarian	Hungarian language
ML_Lithuanian	Lithuanian language
ML_Estonian	Estonian language
ML_Bulgarian	Bulgarian language
ML_Turkish	Turkish language
ML_PortugueseBrazilian	Portuguese (Brazil) language
ML_Korean	Korean language

ML_ChinesePRC	Chinese (PRC) language
ML_ChineseTaiwan	Chinese (Taiwan) language

See also

IEngine::MessagesLanguage

MonospaceDetectionModeEnum

MonospaceDetectionModeEnum enumeration constants specify the mode of monospaced font detection.

```
typedef enum {
      MDM_Auto,
      MDM_NotMonospace,
      MDM_Monospace
} MonospaceDetectionModeEnum;
```

Elements

Name	Description
MDM_Auto	The font is detected automatically.
MDM_NotMonospace	Sets the font to non–monospaced.
MDM_Monospace	Sets the font to monospaced.

See also

IFontFormattingDetectionParams::MonospaceDetectionMode

MultiProcessingModeEnum

MultiProcessingModeEnum enumeration constants specify the mode of distribution analysis and recognition of multi-page documents to CPU cores.

```
typedef enum {
      MPM_Sequential,
      MPM_Auto,
      MPM_Parallel
} MultiProcessingModeEnum;
```

Elements

Name	Description
MPM_Sequential	Pages of a document are recognized sequentially in one process.
MPM_Auto	The number of processes is detected automatically. If one page is recognized or there is only one processor in the system, one process is used for recognition. Otherwise parallel recognition is used.
MPM_Parallel	Pages of a document are always recognized in parallel processes.

Notes:

- When parallel recognition is used, the number of processes is equal to the minimum of the following values:
    - the value of the **IMultiProcessingParams::RecognitionProcessesCount** property,
    - number of available physical or logical CPU cores (depending on the value of the **UseOnlyPhysicalCPUCores** property),
    - number of free CPU cores available in the license,
    - number of pages in the processing document.
- ABBYY FineReader Engine uses self-training recognition algorithm, and thus tunes itself for recognition of text of a certain type. Therefore it is good to use one Document analyzer instance for recognition of a number of pages of the same kind, as this improves speed and quality of recognition as compared with the situation when each page is recognized in a separate

document. Now ABBYY FineReader Engine cannot share the information about recognition between different processes, that is why the results of recognition of the same document in parallel processes and in a single process may be different.

- The distribution among CPU cores is available for the following methods:
  - the **Analyze**, **AnalyzePages**, **Recognize**, **RecognizePages**, **AnalyzeAndRecognize**, **AnalyzeAndRecognizePages**, and **Process** methods of the **FRDocument** object
  - the **AnalyzeAndRecognizePages**, **AnalyzePages**, **RecognizePages** methods of the **Engine** object
  - the **AnalyzeAndRecognizePages**, **AnalyzePages**, **RecognizePages** methods of the **DocumentAnalyzer** object.

See also

IMultiProcessingParams::MultiProcessingMode

NumberingStyleEnum

NumberingStyleEnum enumeration constants describe different styles of list numbering.

```
typedef enum {
      NS_None,
      NS_Decimal,
      NS_UpperRoman,
      NS_LowerRoman,
      NS_UpperLetter,
      NS_LowerLetter,
      NS_Ordinal,
      NS_CardinalText,
      NS_OrdinalText,
      NS_Hex,
      NS_Chicago,
      NS_IdeographDigital,
      NS_JapaneseCounting,
      NS_Aiueo,
      NS_Iroha,
      NS_DecimalFullWidth,
      NS_DecimalHalfWidth,
      NS_JapaneseLegal,
      NS_JapaneseDigitalTenThousand,
      NS_DecimalEnclosedCircle,
      NS_DecimalFullWidth2,
      NS_AiueoFullWidth,
      NS_IrohaFullWidth,
      NS_DecimalZero,
      NS_Bullet,
      NS_Ganada,
      NS_Chosung,
      NS_DecimalEnclosedFullstop,
      NS_DecimalEnclosedParen,
      NS_DecimalEnclosedCircleChinese,
      NS_IdeographEnclosedCircle,
      NS_IdeographTraditional,
      NS_IdeographZodiac,
      NS_IdeographZodiacTraditional,
      NS_TaiwaneseCounting,
      NS_IdeographLegalTraditional,
      NS_TaiwaneseCountingThousand,
      NS_TaiwaneseDigital,
      NS_ChineseCounting,
      NS_ChineseLegalSimplified,
      NS_ChineseCountingThousand,
      NS_ApplicationDefined,
      NS_KoreanDigital,
      NS_KoreanCounting,
      NS_KoreanLegal,
      NS_KoreanDigital2,
      NS_Hebrew1,
      NS_ArabicAlpha,
      NS_Hebrew2,
      NS_ArabicAbjad,
      NS_HindiVowels,
      NS_HindiConsonants,
      NS_HindiNumbers,
      NS_HindiCounting,
      NS_ThaiLetters,
      NS_ThaiNumbers,
      NS_ThaiCounting,
      NS_VietnameseCounting,
      NS_NumberInDash,
      NS_RussianLower,
      NS_RussianUpper
} NumberingStyleEnum;
```

Elements

Name	Description
NS_None	No numbering.
NS_Decimal	Decimal numbering. For example, 1, 2, 3, ... , 9, 10, 11.
NS_UpperRoman	Uppercase Roman numerals. For example, I, II, III.
NS_LowerRoman	Lowercase Roman numerals. For example, i, ii, iii.
NS_UpperLetter	Uppercase letters of the Latin alphabet. For example, A, B, C.
NS_LowerLetter	Lowercase letters of the Latin alphabet. For example, a, b, c.
NS_Ordinal	Ordinal numbers of the current language. For example, 1st, 2nd, 3rd.
NS_CardinalText	Cardinal numerals of the current language. For example, one, two, three.
NS_OrdinalText	Ordinal numerals of the current language. For example, first, second, third.
NS_Hex	Hexadecimal numbering. For example, 1, 2, 3, ... , 9, A, B.
NS_Chicago	Characters as defined in the Chicago Manual of Style. For example, *, †, ‡.
NS_IdeographDigital	Sequential numeric ideographs.
NS_JapaneseCounting	Sequential numbers from the Japanese counting system.
NS_Aiueo	Hiragana characters in the traditional a-i-u-e-o order.
NS_Iroha	Katakana characters in the iroha order.
NS_DecimalFullWidth	Double-byte Arabic numbering.
NS_DecimalHalfWidth	Single-byte Arabic numbering. For example, 1, 2, 3.
NS_JapaneseLegal	Sequential numbers from the Japanese legal counting system.
NS_JapaneseDigitalTenThousand	Sequential numbers from the Japanese digital ten thousand counting system.

NS_DecimalEnclosedCircle	Decimal numbering enclosed in a circle, using the enclosed alphanumeric glyph character. Once the specified sequence reaches 21, the numbers may be replaced with non-enclosed equivalents.
NS_DecimalFullWidth2	An alternative set of double-byte Arabic numbering, if one exists in the current font.
NS_AiueoFullWidth	Full-width hiragana characters in the traditional a-i-u-e-o order.
NS_IrohaFullWidth	Full-width katakana characters in the iroha order.
NS_DecimalZero	Arabic numbering with a zero added to numbers one through nine. For example, 01, 02, 03, ..., 09, 10.
NS_Bullet	Bullet characters. For example, ●.
NS_Ganada	Sequential numbers in the Korean Ganada format.
NS_Chosung	Sequential numbers in the Korean Chosung format.
NS_DecimalEnclosedFullstop	Decimal numbering followed by a period, using the enclosed alphanumeric glyph character. Once the specified sequence reaches 21, the numbers may be replaced with non-enclosed equivalents.
NS_DecimalEnclosedParen	Decimal numbering enclosed in parenthesis, using the enclosed alphanumeric glyph character. Once the specified sequence reaches 21, the numbers may be replaced with non-enclosed equivalents.
NS_DecimalEnclosedCircleChinese	Decimal numbering enclosed in a circle, using the enclosed alphanumeric glyph character. Once the specified sequence reaches 11, the numbers may be replaced with non-enclosed equivalents.
NS_IdeographEnclosedCircle	Sequential numerical ideographs enclosed in a circle, using the appropriate character. Once the specified sequence reaches 11, the numbers may be replaced with non-enclosed equivalents.
NS_IdeographTraditional	Sequential numerical traditional ideographs.
NS_IdeographZodiac	Zodiac ideographs.
NS_IdeographZodiacTraditional	Traditional zodiac ideographs.
NS_TaiwaneseCounting	Sequential numbers from the Taiwanese counting system.
NS_IdeographLegalTraditional	Sequential numerical traditional legal ideographs.
NS_TaiwaneseCountingThousand	Sequential numbers from the Taiwanese counting thousand system.
NS_TaiwaneseDigital	Sequential numbers from the Taiwanese digital counting system.
NS_ChineseCounting	Ascending numbers from the Chinese counting system.
NS_ChineseLegalSimplified	Sequential numbers in the Chinese simplified legal format.
NS_ChineseCountingThousand	Sequential numbers from the Chinese counting thousand system.
NS_ApplicationDefined	Application defined numbering. May be ignored.
NS_KoreanDigital	Sequential numbers from the Korean digital counting system.
NS_KoreanCounting	Sequential numbers from the Korean counting system.
NS_KoreanLegal	Sequential numbers from the Korean legal numbering system.
NS_KoreanDigital2	Sequential numbers from the Korean digital counting system alternate.
NS_Hebrew1	Hebrew numerals.
NS_ArabicAlpha	Characters of the Arabic alphabet.
NS_Hebrew2	Characters of the Hebrew alphabet.
NS_ArabicAbjad	Ascending Arabic Abjad numerals.
NS_HindiVowels	Hindi vowels.
NS_HindiConsonants	Hindi consonants.
NS_HindiNumbers	Hindi numbers.
NS_HindiCounting	Sequential numbers from the Hindi counting system.

NS_ThaiLetters	Thai letters. For example, ก, ข, ค.
NS_ThaiNumbers	Thai numerals. For example, ๒, ๓, ๔.
NS_ThaiCounting	Sequential numbers from the Thai counting system. For example, หนึ่ง, สอง, สาม.
NS_VietnameseCounting	Vietnamese numerals. For example, một, hai, ba.
NS_NumberInDash	Arabic numbering surrounded by dash characters. For example, - 1 -, - 2 -, - 3 -.
NS_RussianLower	Lowercase letters of the Russian alphabet.
NS_RussianUpper	Uppercase letters of the Russian alphabet.

See also

IListLevel::NumberingStyle

ObjectsColorEnum

ObjectsColorEnum enumeration constants describe available colors of the objects, which can be removed from the image.

```
typedef enum {
      OC_Red,
      OC_Green,
      OC_Blue,
      OC_Yellow
} ObjectsColorEnum;
```

Elements

Name	**Description**
OC_Red	Red tint.
OC_Green	Green tint.
OC_Blue	Blue tint.
OC_Yellow	Yellow tint.

See also

IImageDocument::RemoveColorObjects

ObjectsTypeEnum

ObjectsTypeEnum enumeration constants describe available types of color objects, which can be removed from the image.

```
typedef enum {
      OT_Full,
      OT_Background,
      OT_Stamp
} ObjectsTypeEnum;
```

Elements

Name	**Description**
OT_Full	All color object on the image.
OT_Background	Color objects on the background.
OT_Stamp	Color stamps and signatures.

See also

IImageDocument::RemoveColorObjects

OrientationDetectionModeEnum

OrientationDetectionModeEnum enumeration constants specify the mode of orientation detection.

```
typedef enum {
      ODM_Fast,
      ODM_Normal,
      ODM_Thorough
} OrientationDetectionModeEnum;
```

Elements

Name	Description
ODM_Fast	Fast mode. This mode provides the fastest speed of orientation detection at the cost of a moderately decreased quality.
ODM_Normal	Normal mode. The normal mode is an intermediate mode between thorough and fast modes.
ODM_Thorough	Thorough mode. This mode provides the best quality of orientation detection.

See also

IOrientationDetectionParams::OrientationDetectionMode

PageBlackSeparatorRoleEnum

PageBlackSeparatorRoleEnum enumeration constants specify the possible roles of page black separators in a page structure.

```
typedef enum {
      PBSR_Unclassified,
      PBSR_TablePart,
      PBSR_PicturePart,
      PBSR_TextPart,
      PBSR_RunningTitle,
      PBSR_FootNote,
      PBSR_Incut,
      PBSR_InterColumn,
      PBSR_InterSection,
      PBSR_ParagraphBorderBox,
      PBSR_IncutBorderBox
} PageBlackSeparatorRoleEnum;
```

Elements

Name	Description
PBSR_Unclassified	The role is undefined.
PBSR_TablePart	The separator is a part of table.
PBSR_PicturePart	The separator is a part of picture.
PBSR_TextPart	The separator is a part of text: underline, strikeout, or tableader.
PBSR_RunningTitle	Separates a header or footer from the main text.
PBSR_FootNote	Separates a footnote from the main text.
PBSR_Incut	Separates an incut from the main text.
PBSR_InterColumn	Separates two columns.
PBSR_InterSection	Separates two sections.
PBSR_ParagraphBorderBox,	The separator is a side of rectangle surrounding a paragraph.
PBSR_IncutBorderBox	The separator is a side of rectangle surrounding an incut.

See also

IPageBlackSeparator::Role

PageBlackSeparatorTypeEnum

PageBlackSeparatorTypeEnum enumeration constants specify the available types of page black separators.

```
typedef enum {
      PBST_Solid,
      PBST_Dotted
} PageBlackSeparatorTypeEnum;
```

Elements

Name	Description
PBST_Solid	A solid black separator.
PBST_Dotted	A dotted black separator.

See also

IPageBlackSeparator::Type

PageElementTypeEnum

PageElementTypeEnum enumeration constants are used to denote the page element type.

```
typedef enum {
      PET_Text,
      PET_Table,
      PET_Picture,
      PET_Barcode
} PageElementTypeEnum;
```

Elements

Name	Description
PET_Text	Text.
PET_Table	Table.
PET_Picture	Picture.
PET_Barcode	Barcode.

See also

IPageElement::Type

PageFlushingPolicyEnum

PageFlushingPolicyEnum enumeration constants are used to denote the modes of working with document pages (with their **ImageDocument** and **Layout** objects) in memory.

```
typedef enum {
      PFP_KeepInMemory,
      PFP_FlushToDisk,
      PFP_Auto
} PageFlushingPolicyEnum;
```

Elements

Name	Description
PFP_KeepInMemory	The document is always kept in memory.
PFP_FlushToDisk	If there are no references to the **ImageDocument** and the **Layout** objects for corresponding pages, these objects should be unloaded and saved to disk.
PFP_Auto	Automatic mode. If there are no more than 30 pages in the document, the document is kept in memory. Otherwise, its pages are unloaded and saved to disk, if there are no references to the **ImageDocument** and the **Layout** objects for corresponding pages.

See also

IFRDocument::PageFlushingPolicy
IFRPage::Flush

PageSplitDirectionEnum

PageSplitDirectionEnum enumeration constants describe different types of dual pages split that may be detected by means of ABBYY FineReader Engine.

```
typedef enum {
      PSD_HorizontalSplit,
      PSD_VerticalSplit,
      PSD_NoSplit
} PageSplitDirectionEnum;
```

Elements

Name	Description
PSD_HorizontalSplit	Horizontal split is detected.
PSD_VerticalSplit	Vertical split is detected.
PSD_NoSplit	No split is detected.

See also

IDocumentAnalyzer::FindPageSplitPosition
IFRPage::FindPageSplitPosition

ParagraphAlignmentEnum

ParagraphAlignmentEnum enumeration constants are used to denote different types of alignment for a paragraph in the recognized text.

```
typedef enum {
      PA_Left,
      PA_Center,
      PA_Right,
      PA_Justify
} ParagraphAlignmentEnum;
```

Elements

Name	Description
PA_Left	Left-aligned paragraph.
PA_Center	Centered paragraph.
PA_Right	Right-aligned paragraph.
PA_Justify	Justified paragraph (aligned both left and right).

See also

IParagraphParams::ParagraphAlignment

ParagraphExtractionModeEnum

ParagraphExtractionModeEnum enumeration constants describe different modes of paragraph extraction.

```
typedef enum {
      PEM_NormalExtraction,
      PEM_RoughExtraction,
      PEM_SingleLineParagraphsWithSpaceFormatting,
      PEM_SingleLineParagraphsWithWordSeparationOnly
} ParagraphExtractionModeEnum;
```

Elements

Name	Description
PEM_NormalExtraction	Normal paragraph extraction.
PEM_RoughExtraction	Extracts the minimal number of paragraphs (either one paragraph per block or only paragraphs which start with a dropped capital).
PEM_SingleLineParagraphsWithSpaceFormatting	Each line is extracted to a separate paragraph formatted with spaces.
PEM_SingleLineParagraphsWithWordSeparationOnly	Each line is extracted to a separate paragraph without space formatting, blank spaces are to separate words only.

See also

ISynthesisParamsForPage::ParagraphExtractionMode

ParagraphRoleEnum

ParagraphRoleEnum enumeration constants are used to describe the role of the paragraph in the logic structure of the document.

```
typedef enum {
      PR_Text,
      PR_TableText,
      PR_Heading,
      PR_TableHeading,
      PR_PictureCaption,
      PR_TableCaption,
      PR_TableOfContents,
      PR_Footnote,
      PR_Endnote,
      PR_RunningTitle,
      PR_Garbage,
      PR_Other,
      PR_Barcode
} ParagraphRoleEnum;
```

Elements

Name	Description
PR_Text	A paragraph of a text.
PR_TableText	A paragraph of a table cell text.
PR_Heading	A heading paragraph.
PR_TableHeading	A table heading paragraph.
PR_PictureCaption	A picture caption paragraph.
PR_TableCaption	A table caption paragraph.
PR_TableOfContents	A paragraph of a table of contents.
PR_Footnote	A footnote paragraph.
PR_Endnote	An endnote paragraph.
PR_RunningTitle	A running title paragraph.
PR_Garbage	A paragraph contains some garbage.
PR_Other	Some other paragraph role.
PR_Barcode	A barcode paragraph.

See also

IParagraphStyle::ParagraphRole
IGlobalStyleStorage::BaseStyleForParagraphRole

ParagraphTabAlignmentEnum

ParagraphTabAlignmentEnum enumeration constants denote available types of alignment for a single tab stop.

```
typedef enum {
      PTA_Left,
      PTA_Right,
      PTA_Center
} ParagraphTabAlignmentEnum;
```

Elements

Name	Description
PTA_Left	Left-aligned tab stop.
PTA_Right	Right-aligned tab stop.
PTA_Center	Center-aligned tab stop.

See also

ITabPosition::Alignment

PDFAComplianceModeEnum

PDFAComplianceModeEnum enumeration constants are used to set the compliance with PDF/A standard for output PDF files.

Note: ABBYY uses the Adobe Preflight utility (version 9.0) to examine the implementation of export to PDF/A for compliance with standard.

```
typedef enum {
  PCM_None,
  PCM_Pdfa_1b,
  PCM_Pdfa_1a
 } PDFAComplianceModeEnum;
```

Elements

Name	Description
PCM_None	Compliance with PDF/A standard is not necessary.
PCM_Pdfa_1b	The recognized text should be exported to PDF/A-1b format.
PCM_Pdfa_1a	The recognized text should be exported to PDF/A-1a format.

See also

IPDFExportParams::PDFAComplianceMode

PDFColorityModeEnum

PDFColorityModeEnum enumeration constants are used to define color setting for output PDF (PDF/A) files.

```
typedef enum {
      PCM_KeepColority,
      PCM_ForceToGray
} PDFColorityModeEnum;
```

Elements

Name	Description
PCM_KeepColority	Colors will be saved during PDF (PDF/A) export.
PCM_ForceToGray	PDF (PDF/A) files will be saved in gray.

See also

IPDFExportParams::Colority

PDFExportModeEnum

PDFExportModeEnum enumeration constants are used to set the mode of export into PDF format.

```
typedef enum {
      PEM_TextWithPictures,
      PEM_TextOnImage,
      PEM_ImageOnText,
      PEM_ImageOnly
} PDFExportModeEnum;
```

Elements

Name	Description
PEM_TextWithPictures	The recognized text is saved as text, and the pictures are saved as pictures.
PEM_TextOnImage	The entire image is saved as a picture. Text areas are saved as text over the image.
PEM_ImageOnText	The entire image is saved as a picture. The recognized text is put under it. This option is useful if you export your text to document archives: the full page layout is retained and the full-text search is available if you save in this mode.
PEM_ImageOnly	The entire image is saved as a picture. The recognized text and layout information are not used in this mode, so the recognition stage may be skipped.

See also

IPDFExportParams::ExportMode
IPDFAExportParams::ExportMode

PDFExportScenarioEnum

PDFExportScenarioEnum enumeration constants are used to set the scenario of export to PDF (PDF/A) format, which optimizes export for some parameters.

```
typedef enum {
      PES_MaxQuality,
      PES_Balanced,
      PES_MinSize,
      PES_MaxSpeed
} PDFExportScenarioEnum;
```

Elements

Name	Description
PES_MaxQuality	Optimize the PDF (PDF/A) export in order to receive the best quality of the resulting file.
PES_Balanced	The PDF (PDF/A) export will be balanced between the quality of the resulting file, its size and the time of processing.
PES_MinSize	Optimize the PDF (PDF/A) export in order to receive the minimum size of the resulting file.
PES_MaxSpeed	Optimize the PDF (PDF/A) export in order to receive the highest speed of processing.

See also

IPDFExportParams::Scenario

PDFKeyLengthEnum

PDFKeyLengthEnum enumeration constants are used to set the length of the encryption key used to encrypt the PDF file during export.

```
typedef enum {
      PDFKL_40Bit = 5,
      PDFKL_48Bit = 6,
      PDFKL_56Bit = 7,
      PDFKL_64Bit = 8,
      PDFKL_72Bit = 9,
      PDFKL_80Bit = 10,
      PDFKL_88Bit = 11,
      PDFKL_96Bit = 12,
      PDFKL_104Bit = 13,
      PDFKL_112Bit = 14,
      PDFKL_120Bit = 15,
      PDFKL_128Bit = 16
} PDFKeyLengthEnum;
```

Elements

Name	Description
PDFKL_40Bit	The key length is 40 bits.
PDFKL_48Bit	The key length is 48 bits.
PDFKL_56Bit	The key length is 56 bits.
PDFKL_64Bit	The key length is 64 bits.
PDFKL_72Bit	The key length is 72 bits.
PDFKL_80Bit	The key length is 80 bits.
PDFKL_88Bit	The key length is 88 bits.
PDFKL_96Bit	The key length is 96 bits.
PDFKL_104Bit	The key length is 104 bits.
PDFKL_112Bit	The key length is 112 bits.
PDFKL_120Bit	The key length is 120 bits.
PDFKL_128Bit	The key length is 128 bits.

See also

IPDFEncryptionInfo::KeyLength

PDFMRCCompressionLevelEnum

PDFMRCCompressionLevelEnum enumeration constants describe different levels of MRC compression.

```
typedef enum {
      PMRC_LowCompression,
      PMRC_AvgCompression,
      PMRC_MaxCompression,
      PMRC_Custom
} PDFMRCCompressionLevelEnum;
```

Elements

Name	Description
PMRC_LowCompression	This value specifies MRC with low compression. Appropriate for saving documents without quality loss.
PMRC_AvgCompression	This value specifies MRC with average compression ratio. Appropriate for saving the majority of documents. The value is used by default.
PMRC_MaxCompression	This value specifies MRC with maximum compression ratio. Appropriate for documents without background pictures.

PMRC_Custom	This value specifies MRC with user-defined compression ratio.

See also

IPDFMRCParams::CompressionLevel

PDFMRCCompressionLevelEnum

PDFMRCCompressionLevelEnum enumeration constants describe different levels of MRC compression.

```
typedef enum {
      PMRC_LowCompression,
      PMRC_AvgCompression,
      PMRC_MaxCompression,
      PMRC_Custom
} PDFMRCCompressionLevelEnum;
```

Elements

Name	Description
PMRC_LowCompression	This value specifies MRC with low compression. Appropriate for saving documents without quality loss.
PMRC_AvgCompression	This value specifies MRC with average compression ratio. Appropriate for saving the majority of documents. The value is used by default.
PMRC_MaxCompression	This value specifies MRC with maximum compression ratio. Appropriate for documents without background pictures.
PMRC_Custom	This value specifies MRC with user-defined compression ratio.

See also

IPDFMRCParams::CompressionLevel

PDFMRCModeEnum

PDFMRCModeEnum enumeration constants are used to define the mode of using MRC during export to PDF (PDF/A).

```
typedef enum {
  MRC_Auto,
  MRC_Always,
  MRC_Disable
 } PDFMRCModeEnum;
```

Elements

Name	Description
MRC_Auto	ABBYY FineReader Engine will use MRC, if it is necessary.
MRC_Always	Always use MRC.
MRC_Disable	Do not use MRC.

See also

IPDFExportParams::MRCMode

PDFResolutionTypeEnum

PDFResolutionTypeEnum enumeration constants designate the types of picture resolution used in output PDF (PDF/A) files.

```
typedef enum {
  PRT_Desired,
  PRT_Exact,
  PRT_Source
 } PDFResolutionTypeEnum;
```

Elements

Name	Description
PRT_Desired	Use the desired value of the resolution. In this case, picture resolution is defined as follows:

	• if the original resolution of the source image is less than or equal to the desired resolution, the original resolution is preserved, • if the original resolution is above the desired resolution, the program selects the value nearest to the desired resolution.
PRT_Exact	Use the specified resolution.
PRT_Source	Original resolution must be preserved.

See also

IPDFExportParams::ResolutionType

PDFVersionEnum

PDFVersionEnum enumeration constants are used to specify the version of the PDF file.

```
typedef enum {
      PVN_Auto,
      PVN_Version13,
      PVN_Version14,
      PVN_Version15,
      PVN_Version16,
      PVN_Version17
} PDFVersionEnum;
```

Elements

Name	Description
PVN_Auto	The version will be detected automatically.
PVN_Version13	The PDF file will be saved in the version 1.3.
PVN_Version14	The PDF file will be saved in the version 1.4.
PVN_Version15	The PDF file will be saved in the version 1.5.
PVN_Version16	The PDF file will be saved in the version 1.6.
PVN_Version17	The PDF file will be saved in the version 1.7.

See also

IPdfExtendedParams::PDFVersion
IPDFExportParamsOld::PDFVersion
IPDFAExportParamsOld::PDFVersion

ReadingTypeEnum

ReadingTypeEnum enumeration constants are used to designate a reading type of a text. A text on page can be divided into columns or written in a single column.

```
typedef enum {
      TRT_Unknown,
      TRT_LinesBased,
      TRT_ColumnsBased
} ReadingTypeEnum;
```

Elements

Name	Description
TRT_Unknown	The reading type is undefined.
TRT_LinesBased	The text is written in a single column.
TRT_ColumnsBased	The text on page is divided into several columns.

See also

ITextOrientation::ReadingType

RotationTypeEnum

RotationTypeEnum enumeration constants are used to denote the types of rotation that can be performed upon image, or the types of text orientation.

```
typedef enum {
      RT_UnknownRotation = -1,
      RT_NoRotation,
      RT_Clockwise,
      RT_Counterclockwise,
      RT_Upsidedown
} RotationTypeEnum;
```

Elements

Name	Description
RT_UnknownRotation	Rotation type or orientation is undefined.
RT_NoRotation	This value denotes no rotation, or normal orientation.
RT_Clockwise	Rotation 90 degrees clockwise, or clockwise orientation.
RT_Counterclockwise	Rotation 90 degrees counterclockwise, or counterclockwise orientation.
RT_Upsidedown	Rotation upside down, or upside down orientation.

See also

IImageProcessingParams::RotationType
IImageDocument::Transform
IImageDocument::ImageRotation
IPrepareImageMode::Rotation
IImageDocumentEvents::TransformationMade
ITextOrientation::RotationType

RTFPageOrientationEnum

RTFPageOrientationEnum enumeration constants are used to set page orientation during export in RTF/DOC/DOCX or XLSX format.

```
typedef enum {
      POM_Portrait,
      POM_Landscape,
      POM_Auto
} RTFPageOrientationEnum;
```

Elements

Name	Description
POM_Portrait	Sets portrait orientation.
POM_Landscape	Sets landscape orientation.
POM_Auto	The orientation is detected automatically.

See also

IRTFExportParams::PageOrientation
IXLExportParams::PageOrientation

RTFPageSynthesisModeEnum

RTFPageSynthesisModeEnum enumeration constants are used to denote modes of RTF file synthesis from the recognized text when exporting in RTF format.

```
typedef enum {
      PSM_Unknown,
      PSM_RTFPlainText,
      PSM_RTFFormatParagraphs,
      PSM_RTFColumns,
      PSM_RTFExactCopy,
      PSM_RTFEditableCopy
} RTFPageSynthesisModeEnum;
```

Elements

Name	Description
PSM_Unknown	The mode of file synthesis is not defined.
PSM_RTFPlainText	The text in output file is formatted in a single column. Frames are not used. Paragraphs are retained, while types and sizes of fonts are not retained.
PSM_RTFFormatParagraphs	Paragraphs and fonts types and sizes are retained. The text formatting inside paragraphs is not retained.
PSM_RTFColumns	Full formatting is retained using columns and frames. This mode is not suitable for export to clipboard. If it is set when the text is exported to clipboard, it is replaced with the **PSM_RTFFormatParagraphs** mode.
PSM_RTFExactCopy	Produces a document that maintains the formatting of the original. This option is recommended for documents with complex layouts, such as promotion booklets. Note, however, that this option limits the ability to change the text and formatting of the output document.
PSM_RTFEditableCopy	Produces a document that preserves the original format and text flow but allows easy editing.

See also

IRTFExportParams::PageSynthesisMode

RunningTitleModeEnum

RunningTitleModeEnum enumeration constants are used to denote modes of running titles saving.

```
typedef enum {
       RTM_WriteAsNative,
       RTM_WriteAsText,
       RTM_DontWrite
} RunningTitleModeEnum;
```

Elements

Name	Description
RTM_WriteAsNative	The running titles are written to the file according to the running title standard of the export format.
RTM_WriteAsText	The running titles are written to the file as plain text.
RTM_DontWrite	The running titles are not written to the file.

See also

IRTFExportParams::RunningTitleMode
IHTMLExportParams::RunningTitleMode
IXLExportParams::RunningTitleMode
IPPTExportParams::RunningTitleMode
ITextExportParams::RunningTitleMode
IPDFExportParamsOld::RunningTitleMode
IPDFAExportParamsOld::RunningTitleMode

ScanBrightnessControlEnum

ScanBrightnessControlEnum enumeration constants are used to set brightness control modes.

```
typedef enum {
       SBC_Fine,
       SBC_Scanner,
       SBC_Manual
}ScanBrightnessControlEnum;
```

Elements

Name	Description
SBC_Fine	Brightness is controlled by ABBYY FineReader Engine.
SBC_Scanner	Brightness is controlled by the scanner.
SBC_Manual	Brightness is set by the user.

See also

ScanSourceSettings::BrightnessControl

ScanOptionsInterfaceTypeEnum

ScanOptionsInterfaceTypeEnum enumeration constants are used to specify the interface type for the scanning options.

```
typedef enum {
      SOIT_None,
      SOIT_Twain,
      SOIT_Fine
}ScanOptionsInterfaceTypeEnum;
```

Elements

Name	Description
SOIT_None	If this parameter is used no interface will be displayed, the settings specified by using **IScanManager::ScanSourceSettings** will be used for scanning.
SOIT_Twain	Displays the Twain interface.
SOIT_Fine	Displays the ABBYY FineReader interface. 📝 **Note:** In order to use this interface, your license must support the Scanning module.

See also

IScanManager::ScanOptionsInterfaceType

ScanPageRotationAngleEnum

ScanPageRotationAngleEnum enumeration constants are used to set the image rotation angle (once the page has been scanned).

```
typedef enum {
      SPRA_Rotation0,
      SPRA_Rotation90,
      SPRA_Rotation180,
      SPRA_Rotation270
} ScanPageRotationAngleEnum;
```

Elements

Name	Description
SPRA_Rotation0	No rotation.
SPRA_Rotation90	The image to be rotated by 90 degrees.
SPRA_Rotation180	The image to be rotated by 180 degrees.
SPRA_Rotation270	The image to be rotated by 270 degrees.

See also

IScanSourceSettings::RotationAngle

ScanPaperSizeEnum

ScanPaperSizeEnum enumeration constants are used to set size of the scanned page.

```
typedef enum {
	SPS_None,
	SPS_Tabloid,
	SPS_Fanfold,
	SPS_Legal,
	SPS_Folio,
	SPS_Letter,
	SPS_Slide,
	SPS_Executive,
	SPS_Statement,
	SPS_GermanLegalFanfold,
	SPS_GermanFanfold,
	SPS_A0,
	SPS_A1,
	SPS_A2,
	SPS_A3,
	SPS_A4,
	SPS_A5,
	SPS_B1_ISO,
	SPS_B2_ISO,
	SPS_B3_ISO,
	SPS_B4_ISO,
	SPS_B5_ISO,
	SPS_B6_ISO,
	SPS_B4_JIS,
	SPS_B5_JIS,
	SPS_B6_JIS,
	SPS_C3,
	SPS_C4,
	SPS_C5,
	SPS_C6,
	SPS_RA2,
	SPS_RA3,
	SPS_RA4,
	SPS_QUARTO,
	SPS_DL,
	SPS_Envelope14,
	SPS_Envelope12,
	SPS_Envelope11,
	SPS_Envelope10,
	SPS_Envelope9,
	SPS_EnvelopeCheck,
	SPS_EnvelopeMonarch,
	SPS_Custom
}ScanPaperSizeEnum;
```

Elements

Name	Description	Page size in inches	Page size in mm
SPS_None	Page size is not defined.		
SPS_Tabloid	Page size is Tabloid.	11 x 17	279.4 x 431.8
SPS_Fanfold	Page size is Fanfold.	11 x 14.88	

SPS_Legal	Page size is Legal.	8.5 x 14	216 x 356
SPS_Folio	Page size is Folio.	8.5 x 13	
SPS_Letter	Page size is Letter.	8.5 x 11	216 x 279
SPS_Slide	Page size is Slide.	7.33 x 11	
SPS_Executive	Page size is Executive.	7.25 x 10.5	184 x 266
SPS_Statement	Page size is Statement.	5.5 x 8.5	140 x 216
SPS_GermanLegalFanfold	Page size is German Legal Fanfold.	8.5 x 13	
SPS_GermanFanfold	Page size is German Fanfold.	8.5 x 12	
SPS_A0	Page size is À0.	33.1 x 46.8	841 x 1189
SPS_A1	Page size is A1.	23.4 x 33.1	594 x 841
SPS_A2	Page size is A2.	16.5 x 23.4	420 x 594
SPS_A3	Page size is A3.	11.69 x 16.54	297 x 420
SPS_A4	Page size is A4.	8.27 x 11.69	210 x 297
SPS_A5	Page size is A5.	5.83 x 8.27	148 x 210
SPS_B1_ISO	Page size is B1 (ISO).	27.8 x 39.4	707 x 1000
SPS_B2_ISO	Page size is B2 (ISO).	19.7 x 27.8	500 x 707
SPS_B3_ISO	Page size is B3 (ISO).	13.9 x 19.7	353 x 500
SPS_B4_ISO	Page size is B4 (ISO).	9.8 x 13.9	250 x 353
SPS_B5_ISO	Page size is B5 (ISO).	6.9 x 9.8	176 x 250
SPS_B6_ISO	Page size is B6 (ISO).	4.9 x 6.9	125 x 176
SPS_B4_JIS	Page size is B4 (JIS).	10.12 x 14.33	257 x 364
SPS_B5_JIS	Page size is B5 (JIS).	7.17 x 10.12	182 x 257
SPS_B6_JIS	Page size is B6 (JIS).	5.06 x 7.17	128 x 182
SPS_C3	Page size is C3.	12.8 x 18.0	324 x 458
SPS_C4	Page size is C4.	9.0 x 12.8	229 x 324
SPS_C5	Page size is C5.	6.4 x 9.0	162 x 229
SPS_C6	Page size is C6.	4.5 x 6.4	114 x 162
SPS_RA2	Page size is RA2.		430 x 610
SPS_RA3	Page size is RA3.		305 x 430
SPS_RA4	Page size is RA4.		215 x 305
SPS_QUARTO	Page size is QUARTO.		215 x 275
SPS_DL	Page size is Envelope DL.	4.33 x 8.66	110 x 220
SPS_Envelope14	Page size is Envelope #14.	5 x 11.5	
SPS_Envelope12	Page size is Envelope #12.	4.75 x 11	
SPS_Envelope11	Page size is Envelope #11.	4.5 x 10.38	
SPS_Envelope10	Page size is Envelope #10.	4.13 x 9.5	104,8 x 241,3
SPS_Envelope9	Page size is Envelope #9.	3.88 x 8.88	
SPS_EnvelopeCheck	Page size is Envelope Check.	3.88 x 8.58	
SPS_EnvelopeMonarch	Page size is Envelope Monarch.	3.88 x 7.5	98,4 x 190,5
SPS_Custom	Page size is set by the user.		

See also

IScanSourceSettings::PaperSize

ScanPictureModeEnum

ScanPictureModeEnum enumeration constants are used to set image type.

```
typedef enum {
      SPM_BlackAndWhite,
      SPM_Grayscale,
      SPM_Color
}ScanPictureModeEnum;
```

Elements

Name	Description
SPM_BlackAndWhite	Black-and-white image.
SPM_Grayscale	Gray image.
SPM_Color	Color image.

See also

IScanSourceSettings::PictureMode

SkewCorrectionModeEnum

SkewCorrectionModeEnum enumeration constants are used to set skew correction modes.

```
typedef enum {
  SCM_Unknown,
  SCM_AccordingToPage,
  SCM_Always,
  SCM_Never
 }SkewCorrectionModeEnum;
```

Elements

Name	Description
SCM_Unknown	The mode of skew correction is not defined.
SCM_AccordingToPage	Skew correction is performed according to the page settings.
SCM_Always	Skew correction is always performed.
SCM_Never	Skew correction is not performed.

See also

ITextBlockAnalysisParams::SkewCorrectionMode

SeparatorTypeEnum

SeparatorTypeEnum enumeration constants are used to specify separator type.

```
typedef enum {
      ST_Unknown,
      ST_Solid,
      ST_Dotted
}SeparatorTypeEnum;
```

Elements

Name	Description
ST_Unknown	The separator type is undefined.
ST_Solid	The separator is a solid line.
ST_Dotted	The separator is a dotted line.

See also

ISeparatorBlock::Type

StreamElementAlignmentEnum

StreamElementAlignmentEnum enumeration constants are used to denote different types of alignment for an element.

```
typedef enum {
      SEA_None,
      SEA_Left,
      SEA_Center,
      SEA_Right,
      SEA_Justify
} StreamElementAlignmentEnum;
```

Elements

Name	Description
SEA_None	No alignment. The position of stream element in the column is defined by the **LeftIndent** and **RightIndent** properties of the **StreamElementLocationParams** object. If the width of the stream element with left indent and right indent is greater than the width of the column, both left and right indent are decreased by the same value.
SEA_Left	The left side of the stream element coincides with the left side of the column. The values of the **LeftIndent** and **RightIndent** properties of the **StreamElementLocationParams** object are ignored.
SEA_Center	The center of the stream element coincides with the vertical line through the center of the column. The values of the **LeftIndent** and **RightIndent** properties of the **StreamElementLocationParams** object are ignored.
SEA_Right	The right side of the stream element coincides with the right side of the column. The values of the **LeftIndent** and **RightIndent** properties of the **StreamElementLocationParams** object are ignored.
SEA_Justify	The left side of the stream element coincides with the left side of the column, the right side of the stream element coincides with the right side of the column. If the width of the stream element is not equal to the width of the column, the positions of the picture and barcode elements are the same as for SEA_Center, the position of the table element is the same as for SEA_Center or all the cells are stretched or squeezed proportionally in order the width of the stream element is equal to the width of the column. The values of the **LeftIndent** and **RightIndent** properties of the **StreamElementLocationParams** object are ignored.

See also

IStreamElementLocationParams::Alignment

StreamTypeEnum

StreamTypeEnum enumeration constants are used to specify the types of document and page streams.

```
typedef enum {
      ST_MainText,
      ST_Incut,
      ST_Footnote,
      ST_Artefact
} StreamTypeEnum;
```

Elements

Name	Description
ST_MainText	Main text. Each document section can have only one stream of the main text type.
ST_Incut	Incut.
ST_Footnote	Footnote.
ST_Artefact	Artefact. Document stream cannot be of this type.

See also

IDocumentSection::AddNewStream
IDocumentStream::Type
IPageStream::Type

StyleParamsEnum

StyleParamsEnum enumeration constants are used to denote different parameters of a font style. They are used as a mask in some methods of the **Paragraph** object. The mask is an OR combination of these constants and define what properties of the **CharParams** object should be taken into account in these methods. The constants are also used as a mask in the **IFontStyle::OverriddenStyleParams** property.

```
typedef enum {
      SF_Bold = 1,
      SF_Italic = 2,
      SF_Underlined = 4,
      SF_Strikeout = 8,
      SF_SmallCaps = 16,
      SF_FontSize = 0x10000,
      SF_FontName = 0x20000,
      SF_Scaling = 0x40000,
      SF_Spacing = 0x80000,
      SF_Color = 0x100000,
      SF_BackgroundColor = 0x200000,
      SF_BaseLineRise = 0x400000
} StyleParamsEnum;
```

Elements

Name	Description
SF_Bold	Designates the ICharParams::IsBold or IFontStyle::IsBold property.
SF_Italic	Designates the ICharParams::IsItalic or IFontStyle::IsItalic property.
SF_Underlined	Designates the ICharParams::IsUnderlined or IFontStyle::IsUnderlined property.
SF_Strikeout	Designates the ICharParams::IsStrikeout or IFontStyle::IsStrikeout property.
SF_SmallCaps	Designates the ICharParams::IsSmallCaps or IFontStyle::IsSmallCaps property.
SF_FontSize	Designates the ICharParams::FontSize or IFontStyle::FontSize property.
SF_FontName	Designates the ICharParams::FontName or IFontStyle::FontName property.
SF_Scaling	Designates the ICharParams::HorizontalScale or IFontStyle::HorizontalScale property.
SF_Spacing	Designates the ICharParams::Spacing or IFontStyle::Spacing property.
SF_Color	Designates the ICharParams::Color or IFontStyle::Color property.
SF_BackgroundColor	Designates the IParagraphParams::BackgroundColor property.
SF_BaseLineRise	Designates the ICharParams::BaseLine or IFontStyle::BaseLine property.

See also

IParagraph::SetCharParams
IParagraph::NextGroup
IFontStyle::OverriddenStyleParams

TabLeaderTypeEnum

TabLeaderTypeEnum enumeration constants denote available types of tab leaders.

```
typedef enum {
      TLT_None,
      TLT_Dots,
      TLT_MiddleDots,
      TLT_Hyphens,
      TLT_Underline
} TabLeaderTypeEnum;
```

Elements

Name	Description
TLT_None	No tab leader.
TLT_Dots	Dots on the base line.
TLT_MiddleDots	Dots in the middle of the line (not on the base line).
TLT_Hyphens	Hyphens are used as tab leaders.
TLT_Underline	Underline is used as tab leader.

See also

ITabPosition::TabLeaderType

TableCellVertAlignmentEnum

TableCellVertAlignmentEnum enumeration constants are used to denote different types of vertical alignment of the text in table cells.

```
typedef enum {
      TCVA_Top,
      TCVA_Center,
      TCVA_Bottom
} TableCellVertAlignmentEnum;
```

Elements

Name	Description
TCVA_Top	Align top.
TCVA_Center	Align center.
TCVA_Bottom	Align bottom.

See also

ITextTableCell::VertAlignment

TableSeparatorTypeEnum

TableSeparatorTypeEnum enumeration constants are used to denote different types of table separators.

```
typedef enum {
      TST_Absent,
      TST_Unknown,
      TST_Invisible,
      TST_Explicit,
      TST_Multiple
} TableSeparatorTypeEnum;
```

Elements

Name	Description
TST_Absent	This type of table separator is used inside a merged cell.
TST_Unknown	This type is initially assigned to table separators created by user.
TST_Invisible	This type of separator may be assigned as a result of layout recognition. As a rule, this type of separator appears where the original table does not have one but where it "should be".

TST_Explicit	This type of separator may be assigned as a result of layout recognition. It corresponds to an ordinary black or color table separator of the original table.
TST_Multiple	This type of separator may be assigned as a result of table editing.

See also

ITableSeparator::Type
ITableSeparator::SetType
TableSeparator

TextCategoryEnum

TextCategoryEnum enumeration constants describe different categories of text, that may be recognized using a text language.

```
typedef enum {
      TC_Unknown,
      TC_NaturalText,
      TC_TableCells,
      TC_FormFields,
      TC_Listing
} TextCategoryEnum;
```

Elements

Name	Description
TC_Unknown	This value specifies the text of any type.
TC_NaturalText	This value specifies the text in a natural language. It consists of sentences, sentences in turn consist of words in the natural language with rare inclusions of digits, punctuation marks, abbreviations, URL, etc.
TC_TableCells	This constant describes the text located in table cells. Generally it contains numbers, single words or phrases. Contents of the cells are semantically unrelated (or the relation is of a very general type).
TC_FormFields	This constant describes the text located in fields of a filled in form. It contains single words, phrases, numbers. Allowed syntax is frequently limited. Punctuation and spaces arrangement rules are often not kept.
TC_Listing	This value corresponds to a text in a programming language or some other formal language.

See also

ITextLanguage::ImpliedTextCategory

TextEncodingTypeEnum

TextEncodingTypeEnum enumeration constants are used to denote possible types of the output file encoding for export in TXT and CSV formats.

```
typedef enum {
      TET_Simple,
      TET_UTF8,
      TET_UTF16,
      TET_Auto
} TextEncodingTypeEnum;
```

Elements

Name	Description
TET_Simple	Simple encoding, one byte per symbol.
TET_UTF8	Unicode UTF8 format. UTF8 is a code page that uses a string of bytes to represent a 16-bit Unicode string where ASCII text (<=U+007F) remains unchanged as a single byte, U+0080-07FF (including Latin, Greek, Cyrillic, Hebrew, and Arabic) is converted to a 2-byte sequence, and U+0800-FFFF (Chinese, Japanese, Korean, and others) becomes a 3-byte sequence.
TET_UTF16	Native Unicode format where every symbol is represented by two-byte sequence.
TET_Auto	Encoding is selected automatically.

See also

ITextExportParams::EncodingType
IHTMLExportParams::EncodingType
IPlainText::SaveToTextFile

TextLanguageLetterSetEnum

TextLanguageLetterSetEnum enumeration constants describe different types of letter sets that may be assigned to a text language.

```
typedef enum {
      TLLS_InterwordPunctuators,
      TLLS_ProhibitedLetters,
      TLLS_Prefixes,
      TLLS_Suffixes
} TextLanguageLetterSetEnum;
```

Elements

Name	Description
TLLS_InterwordPunctuators	This value denotes punctuation marks that may be found between words. There is no analogue of such letter set for a base language, as it represents the language of a word.
TLLS_ProhibitedLetters	This value denotes a set of letters that are prohibited for the current text language. They will never appear in the recognized text.
TLLS_Prefixes	This value denotes punctuation marks that may appear immediately before a word. These punctuation marks are additional to those defined by the base language.
TLLS_Suffixes	This value denotes punctuation marks that may appear immediately after a word. These punctuation marks are additional to those defined by the base language.

See also

ITextLanguage::LetterSet

TextRoleEnum

TextRoleEnum enumeration constants are used to set text role.

```
typedef enum {
      TR_MainText,
      TR_Footnote,
      TR_Incut,
      TR_RunningTitle,
      TR_PictureCaption,
      TR_TableCaption,
      TR_Other,
      TR_CompoundText,
      TR_AbstractText
} TextRoleEnum;
```

Elements

Name	Description
TR_MainText	Main text.
TR_Footnote	Footnote body.
TR_Incut	Incut.
TR_RunningTitle	Running title.
TR_PictureCaption	Picture caption.
TR_TableCaption	Table caption.
TR_Other	Some other role (garbage, artefacts, line numbers in legal document, etc.)

TR_CompoundText	Whole text of a text block. This constant is used for compatibility. **Note:** If a text has such role, its role cannot be changed.
TR_AbstractText	Text, which does not refer to any particular place in the document. This constant is used for compatibility. **Note:** If a text has such role, its role cannot be changed.

See also

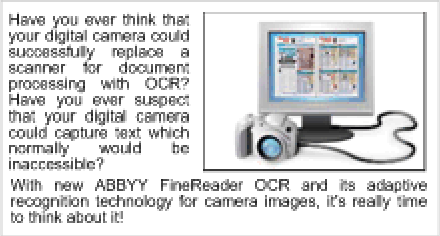
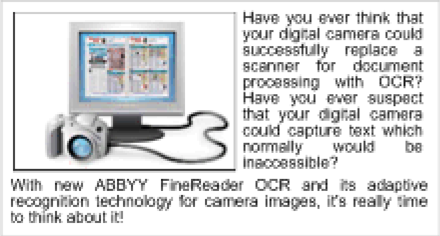
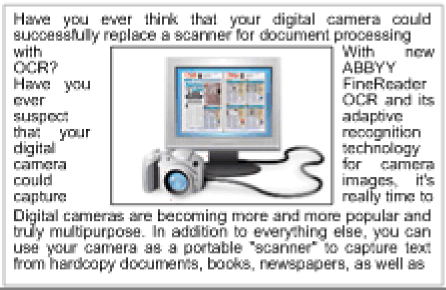
IText::TextRole

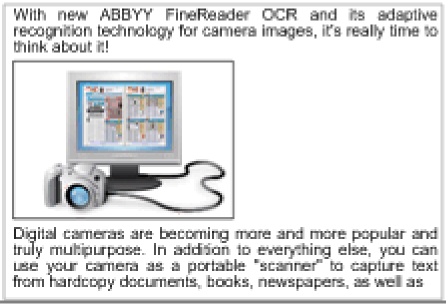
TextWrappingEnum

TextWrappingEnum enumeration constants are used to designate the different types of text wrapping around an incut.

```
typedef enum {
	TW_Undefined,
	TW_OnTheLeft,
	TW_OnTheRight,
	TW_Around,
	TW_None
} TextWrappingEnum;
```

Elements

Name	Description
TW_Undefined	The text streamline is undefined. The value has not been set yet.
TW_OnTheLeft	The text is to the left of the frame.  Have you ever think that your digital camera could successfully replace a scanner for document processing with OCR? Have you ever suspect that your digital camera could capture text which normally would be inaccessible? With new ABBYY FineReader OCR and its adaptive recognition technology for camera images, it's really time to think about it!
TW_OnTheRight	The text is to the right of the frame.  Have you ever think that your digital camera could successfully replace a scanner for document processing with OCR? Have you ever suspect that your digital camera could capture text which normally would be inaccessible? With new ABBYY FineReader OCR and its adaptive recognition technology for camera images, it's really time to think about it!
TW_Around	The text is both to the left and to the right of the frame.  Have you ever think that your digital camera could successfully replace a scanner for document processing with OCR? Have you ever suspect that your digital camera could capture With new ABBYY FineReader OCR and its adaptive recognition technology for camera images, it's really time to Digital cameras are becoming more and more popular and truly multipurpose. In addition to everything else, you can use your camera as a portable "scanner" to capture text from hardcopy documents, books, newspapers, as well as
TW_None	The text break. There is no text streamline.

	 With new ABBYY FineReader OCR and its adaptive recognition technology for camera images, it's really time to think about it! Digital cameras are becoming more and more popular and truly multipurpose. In addition to everything else, you can use your camera as a portable "scanner" to capture text from hardcopy documents, books, newspapers, as well as

See also

IIncut::TextWrapping

TextTableSeparatorTypeEnum

TextTableSeparatorTypeEnum enumeration constants are used to denote different types of table separators in the table which contains text.

```
typedef enum {
      TTST_CellSeparator,
      TTST_TableInvisibleSeparator,
      TTST_TableVisibleSeparator,
} TextTableSeparatorTypeEnum;
```

Elements

Name	Description
TTST_CellSeparator	This type of separator is used inside a cell.
TTST_TableInvisibleSeparator	This type of separator appears where the original table does not have one but where it "should be".
TTST_TableVisibleSeparator	This type of separator corresponds to an ordinary black or color table separator of the original table.

See also

ITextTable::SetVSeparator
ITextTable::SetHSeparator
ITextTable::VSeparatorType
ITextTable::HSeparatorType

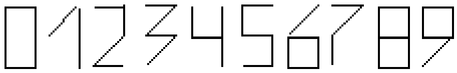
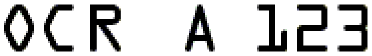
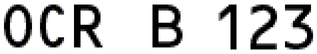
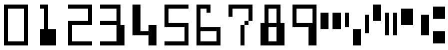
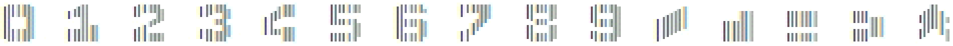
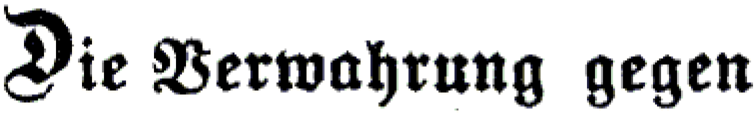
TextTypeEnum

TextTypeEnum enumeration constants are used to describe the type of recognized text.

```
typedef enum {
      TT_Normal       = 0x00000001,
      TT_Typewriter   = 0x00000002,
      TT_Matrix       = 0x00000004,
      TT_Index        = 0x00000008,
      TT_Handprinted  = 0x00000010,
      TT_OCR_A        = 0x00000020,
      TT_OCR_B        = 0x00000040,
      TT_MICR_E13B    = 0x00000080,
      TT_MICR_CMC7    = 0x00000100,
      TT_Gothic       = 0x00000200,
      TT_ToBeDetected = 0
} TextTypeEnum;
```

Elements

Name	Description

TT_Normal	This value corresponds to a common typographic type of text.
TT_Typewriter	This value tells ABBYY FineReader Engine to presume that the text on the recognized image is typed on a typewriter.
TT_Matrix	This value tells ABBYY FineReader Engine to presume that the text on the recognized image is printed on a dot-matrix printer.
TT_Index	This constant corresponds to a special set of characters including only digits written in ZIP-code style. They look as follows: 
TT_Handprinted	This value corresponds to handprinted text. It may look as follows: **Note** that automatic analysis is not available for handprinted text. The coordinates of blocks containing handprinted text should be set manually.
TT_OCR_A	This value corresponds to a monospaced font, designed for Optical Character Recognition. Largely used by banks, credit card companies and similar businesses. It may look as follows: 
TT_OCR_B	This value corresponds to a font designed for Optical Character Recognition. It may look as follows: 
TT_MICR_E13B	This value corresponds to a special set of numeric characters printed with special magnetic inks. MICR (Magnetic Ink Character Recognition) characters are found in a variety of places, including personal checks. It may look as follows: 
TT_MICR_CMC7	This value corresponds to a special MICR barcode font (CMC-7). It may look as follows: 
TT_Gothic	This value tells ABBYY FineReader Engine to presume that the text on the recognized image is printed with the Gothic type. It may look as follows:  For this text type, ABBYY FineReader Engine currently supports only "Fraktur" font.
TT_ToBeDetected	This value tells ABBYY FineReader Engine to automatically detect the type of the text. It may be used as the value of the **IRecognizerParams::TextType** property. The possible values of type to detect are stored as the **PossibleTextTypes** property of the **RecognizerParams** object.

See also

IRecognizerParams::TextTypes
Using Text Type Autodetection
IRecognizerParams::TextType
IRecognizerParams::PossibleTextTypes
Text Types

TrainingImageFormatEnum

TrainingImageFormatEnum enumeration constants are used to denote the types of image which are used during pattern training.

```
typedef enum {
      TIF_Binarized,
      TIF_Gray
} TrainingImageFormatEnum;
```

Elements

Name	Description
TIF_Binarized	Black and white image, 1 bit per pixel.
TIF_Gray	Gray image, 8 bits per pixel.

See also

ICharParams::FontType
ICharParams::SetFont

TXTExportFormatEnum

TXTExportFormatEnum enumeration constants are used to denote the format of export to TXT and CSV files.

```
typedef enum {
	TEF_TXT,
	TEF_CSVFullLayout,
	TEF_CSVTablesOnly
} TXTExportFormatEnum;
```

Elements

Name	Description
TEF_TXT	TXT format.
TEF_CSVFullLayout	CSV format with full layout retained.
TEF_CSVTablesOnly	CSV format with text from table blocks only.

See also

ITextExportParams::ExportFormat

WordModelTypeEnum

WordModelTypeEnum enumeration constants are used to describe the type of the word model.

```
typedef enum {
	WMT_MonolingualWord,
	WMT_RegExpWord,
	WMT_BilingualComposite,
	WMT_Acronym,
	WMT_Number,
	WMT_NumberWithQualifier,
	WMT_WordNumberComposite,
	WMT_BilingualWordNumberComposite,
	WMT_RomanNumber,
	WMT_MonolingualWordWithExtras,
	WMT_MixedFormDictionaryWord,
	WMT_PhoneNumber,
	WMT_Punctuation,
	WMT_FileName,
	WMT_UrlOrEmail,
	WMT_NoSuitableModel
} WordModelTypeEnum;
```

Elements

Name	Description

WMT_MonolingualWord	A common word. Its grammar is determined by the language alphabet. Besides that, the word can contain characters-separators, e.g., "/" or "-".
WMT_RegExpWord	A word from the language which grammar is described by a regular expression.
WMT_BilingualComposite	A bilingual compound word with an explicit dividing point.
WMT_Acronym	An acronym consisting of capital letters. The word can contain digits and separators, e.g., "B2B", "C.E.R.N.".
WMT_Number	A word consisting of digits and punctuators, e.g., "123", "4.56", "#789".
WMT_NumberWithQualifier	A word with a prefix or suffix that serves as a qualifier or inflexion, e.g., "USD250", "1.2GHz", "2nd".
WMT_WordNumberComposite	A compound word with an explicit dividing point consisting of a word and a digit, e.g., "2-meter".
WMT_BilingualWordNumberComposite	A compound word with explicit dividing points consisting of two words belonging to different languages and a number, e.g., "Windows-2000-kompatibel".
WMT_RomanNumber	A Roman number.
WMT_MonolingualWordWithExtras	A word consisting of the language alphabet characters and special characters, digits, etc., e.g., "Alias|Wavefront".
WMT_MixedFormDictionaryWord	A word belonging to the mixed form dictionary. It can contain any characters including characters from the alphabets of different languages. Non-dictionary words are not allowed.
WMT_PhoneNumber	A phone number. A prefix is allowed, e.g., "Ph.(495)123-45678".
WMT_Punctuation	A set of punctuation marks separated from a word by a space(s).
WMT_FileName	A DOS/Windows or UNIX file name, e.g., "README.TXT", "C:\WINNT\system32", "/etc/motd.rc".
WMT_UrlOrEmail	An URL or e-mail address, e.g., "http://www.abbyy.com", "engine_support@abbyy.com".
WMT_NoSuitableModel	A word that does not meet any word model. Every word character is recognized separately, without context. The recognition result may be a meaningless character sequence.

See also

IWordRecognitionVariant::ModelType

WritingStyleEnum

WritingStyleEnum enumeration constants are used to describe available writing styles of handprinted letters.

```
typedef enum {
      WS_Default,
      WS_American,
      WS_German,
      WS_Russian,
      WS_Polish,
      WS_Thai,
      WS_Japanese,
      WS_Arabic,
      WS_Baltic,
      WS_British,
      WS_Bulgarian,
      WS_Canadian,
      WS_Czech,
      WS_Croatian,
      WS_French,
      WS_Greek,
      WS_Hungarian,
      WS_Italian,
      WS_Romanian,
      WS_Slovak,
      WS_Spanish,
      WS_Turkish,
      WS_Ukrainian,
      WS_Common,
      WS_Chinese,
      WS_Azerbaijan,
      WS_Kazakh,
      WS_Kirgiz,
      WS_Latvian
} WritingStyleEnum;
```

Elements

Name	Description
WS_Default	The writing style is selected depending on the current language of the operating system. This constant cannot be the return value of the **IRecognizerParams::WritingStyle** property. If this property was set to WS_Default, it returns the writing style corresponding to the language of the operating system.
WS_American	The American writing style.
WS_German	The German writing style.
WS_Russian	The Russian writing style.
WS_Polish	The Polish writing style.
WS_Thai	The Thai writing style.
WS_Japanese	The Japanese writing style.
WS_Arabic	The Arabic writing style.
WS_Baltic	The Baltic writing style.
WS_British	The British writing style.
WS_Bulgarian	The Bulgarian writing style.
WS_Canadian	The Canadian writing style.
WS_Czech	The Czech writing style.
WS_Croatian	The Croatian writing style.
WS_French	The French writing style.
WS_Greek	The Greek writing style.
WS_Hungarian	The Hungarian writing style.
WS_Italian	The Italian writing style.
WS_Romanian	The Romanian writing style.
WS_Slovak	The Slovak writing style.
WS_Spanish	The Spanish writing style.
WS_Turkish	The Turkish writing style.
WS_Ukrainian	The Ukrainian writing style.
WS_Common	The Esperanto writing style.
WS_Chinese	The Chinese writing style.
WS_Azerbaijan	The Azerbaijan writing style.
WS_Kazakh	The Kazakh writing style.

WS_Kirgiz	The Kirgiz writing style.
WS_Latvian	The Latvian writing style.

See also

IRecognizerParams::WritingStyle
Recognizing Handprinted Texts

XLFileFormatEnum

XLFileFormatEnum enumeration constants are used to describe formats available for the XLS file format.

```
typedef enum {
      XLFF_BIFF8,
      XLFF_BIFF5,
      XLFF_DoubleStream
} XLFileFormatEnum;
```

Elements

Name	Description
XLFF_BIFF8	This is a newer format of XLS-formatted clipboard data.
XLFF_BIFF5	This format is previous to the XLFF_BIFF8 one and may be used for compatibility with the older versions of MS Excel.
XLFF_DoubleStream	Both formats (MS Excel 5 and 8 versions) are saved to the same file.

See also

IXLExportParams::XLFileFormat

XLSXPaperSizeEnum

XLSXPaperSizeEnum enumeration constants are used to denote available paper sizes for output XLSX files.

```
typedef enum {
      XLPS_NotSpecified,
      XLPS_Autodetect,
      XLPS_Letter,
      XLPS_Legal,
      XLPS_Statement,
      XLPS_Executive,
      XLPS_A2,
      XLPS_A3,
      XLPS_A4,
      XLPS_A5,
      XLPS_B4,
      XLPS_B5,
      XLPS_Folio,
      XLPS_11x17,
      XLPS_Envelope10,
      XLPS_EnvelopeDL,
      XLPS_EnvelopeC5,
      XLPS_EnvelopeB5,
      XLPS_EnvelopeMonarch,
      XLPS_JapanesePostcard,
      XLPS_Quarto,
      XLPS_10x14,
      XLPS_C,
      XLPS_D,
      XLPS_E,
        XLPS_9x11,
        XLPS_10x11,
        XLPS_Letter_Extra,
        XLPS_Legal_Extra,
        XLPS_Tabloid_Extra,
        XLPS_A4_Extra,
        XLPS_SuperA,
        XLPS_SuperB,
        XLPS_A4_Plus,
        XLPS_A3_Extra,
        XLPS_A5_Extra,
        XLPS_ISO_B5
} XLSXPaperSizeEnum;
```

Elements

Name	Description
XLPS_NotSpecified	Paper size should not be specified in the output file.
XLPS_Autodetect	The paper size should be selected automatically. The program selects the minimal paper size which encloses all the layouts of the exporting pages.
XLPS_Letter	Letter (8-1/2 in. × 11 in.)
XLPS_Legal	Legal (8-1/2 in. × 14 in.)
XLPS_Statement	Statement (5-1/2 in. × 8-1/2 in.)
XLPS_Executive	Executive (7-1/4 in. × 10-1/2 in.)
XLPS_A2	A2 (420 mm × 594 mm)
XLPS_A3	A3 (297 mm × 420 mm)
XLPS_A4	A4 (210 mm × 297 mm)
XLPS_A5	A5 (148 mm × 210 mm)
XLPS_B4	B4 (JIS) (257 mm × 364 mm)
XLPS_B5	B5 (176 mm × 250 mm)
XLPS_Folio	Folio (8-1/2 in. × 13 in.)
XLPS_11x17	11 in. × 17 in.
XLPS_Envelope10	Envelope #10 (4-1/8 in. × 9-1/2 in.)
XLPS_EnvelopeDL	Envelope DL (110 mm × 220 mm)
XLPS_EnvelopeC5	Envelope C5 (162 mm × 229 mm)
XLPS_EnvelopeB5	Envelope B5 (176 mm × 250 mm)
XLPS_EnvelopeMonarch	Envelope Monarch (3-7/8 in. × 7-1/2 in.)
XLPS_JapanesePostcard	Japanese Postcard (100 mm × 148 mm)
XLPS_Quarto	Quarto paper (215 mm × 275 mm)
XLPS_10x14	Standard paper (10 in. × 14 in.)
XLPS_C	C paper (17 in. × 22 in.)
XLPS_D	D paper (22 in. × 34 in.)
XLPS_E	E paper (34 in. × 44 in.)
XLPS_9x11	9 in. × 11 in.
XLPS_10x11	10 in. × 11 in.
XLPS_Letter_Extra	Letter extra paper (9.275 in. × 12 in.)

XLPS_Legal_Extra	Legal extra paper (9.275 in. × 15 in.)
XLPS_Tabloid_Extra	Tabloid extra paper (11.69 in. × 18 in.)
XLPS_A4_Extra	A4 extra paper (236 mm × 322 mm)
XLPS_SuperA	SuperA (227 mm × 356 mm)
XLPS_SuperB	SuperB paper (305 mm × 487 mm)
XLPS_A4_Plus	A4 plus paper (210 mm × 330 mm)
XLPS_A3_Extra	A3 extra paper (322 mm × 445 mm)
XLPS_A5_Extra	A5 extra paper (174 mm × 235 mm)
XLPS_ISO_B5	ISO B5 extra paper (201 mm × 276 mm)

See also

IXLExportParams::PaperSize

XMLCharAttributesEnum

XMLCharAttributesEnum enumeration constants are used to describe groups of character attributes to be written in files in XML format.

```
typedef enum {
  XCA_None,
  XCA_Ascii,
  XCA_Basic,
  XCA_Extended
 } XMLCharAttributesEnum;
```

Elements

Name	Description
XCA_None	No character attributes are to be written in files in XML format.
XCA_Ascii	Character coordinates and character confidence are to be written in files in XML format. Exactly the same format is used by **IPlainText::SaveToAsciiXMLFile**.
XCA_Basic	Character coordinates are to be written in files in XML format.
XCA_Extended	Character coordinates, character confidence and extended character attributes are to be written in files in XML format. The following extended attributes are written: • whether the word was found in the dictionary, • whether the word was recognized with a standard or user-defined language, • whether the word is a number, • whether the word is an identifier, • probability that a character is written with a Serif font, • penalty for discordance of characters in a word, • the mean width of stroke in the RLE representation of a word image.

See also

IXMLExportParams::WriteCharAttributes

VolumeRefreshingPeriodEnum

VolumeRefreshingPeriodEnum enumeration constants are used to denote the period during which the ABBYY FineReader Engine license limits the number of the recognition and export operations.

```
typedef enum {
  VRP_Day,
  VRP_Week,
  VRP_Month,
  VRP_Quarter,
  VRP_HalfYear,
  VRP_Year,
  VRP_Infinite
 } VolumeRefreshingPeriodEnum;
```

Elements

Name	Description
LLP_Day	The remaining units counter is refreshed at the beginning of each day.
LLP_Week	The remaining units counter is refreshed at the beginning of each week.
LLP_Month	The remaining units counter is refreshed at the beginning of each month.
LLP_Quarter	The remaining units counter is refreshed at the beginning of each quarter.
LLP_HalfYear	The remaining units counter is refreshed at the beginning of each half a year.
LLP_Year	The remaining units counter is refreshed at the beginning of each year.
LLP_Infinite	The remaining units counter is never refreshed.

See also

ILicense::VolumeRefreshingPeriod

Standard Return Codes

Here is a list of the standard return codes of ABBYY FineReader Engine functions and properties.

Return code	Value	Description
S_OK	0 (&H00000000L)	Method completed successfully.
E_OUTOFMEMORY	-2147024882 (&H8007000EL)	There was not enough memory to perform the operation.
E_UNEXPECTED	-2147418113 (&H8000FFFFL)	Unexpected internal error.
E_ABORT	-2147467260 (&H80004004L)	Operation was aborted by the user.
E_NOTIMPL	-2147467263 (&H80004001L)	Method is not implemented.
E_POINTER	-2147467261 (&H80004003L)	Invalid pointer argument.
E_INVALIDARG	-2147024809 (&H80070057L)	One or more arguments are invalid.
CO_E_OBJNOTCONNECTED	-2147220995 (&H800401FDL)	A pointer to an object was passed that is no longer valid (this object was destroyed).
CLASS_E_NOTLICENSED	-2147221230 (&H80040112L)	This copy of ABBYY FineReader Engine is not registered.
CO_E_NOT_SUPPORTED	-2147467231 (&H80004021L)	Some property or method is not available under the current license.
E_FAIL	-2147467259 (&H80004005L)	Unspecified error.

Note: These return codes you can find in the Microsoft® Platform Software Development Kit (SDK) header file winerror.h.

Here is a list of interface-specific return codes of ABBYY FineReader Engine functions and properties. All these codes are defined in the ABBYY FineReader Engine type library.

Return code	Value	Description
FREN_E_PATTERN_TRAINING_ABORTED	-2147221503 (&H80040001)	Pattern training was aborted by the user.

Other return codes are possible, specifically those related to file system errors.

See also

Error Handling

Licensing

A special protection technology is used to protect ABBYY FineReader Engine 10 from illegal copying and distribution. This technology effectively excludes unauthorized use of ABBYY products by persons who have not signed a License Agreement with the software copyright owner.

Developer and Runtime Licenses

ABBYY FineReader Engine has two types of licenses:

- **Developer License**
  This license grants an SDK customer the right to use ABBYY FineReader Engine for development purposes only or for internal use of the developed applications only under the terms of Software Developer License Agreement. Developer License does not allow developers to distribute their applications with ABBYY FineReader Engine functions inside or to use the developed applications internally.

- **Runtime License**
  This license grants developers the right to distribute ABBYY FineReader Engine functions inside developer's applications. Runtime licensing is regulated by Runtime License Agreement with ABBYY.
  ⚠**Important!** The Runtime License should correspond to the Developer License under which your application was compiled.

Each license defines available ABBYY FineReader Engine functionality by the set of included modules. For details about functionality your license includes see the description of ABBYY FineReader Engine 10 Modules.

Standalone and Network Licenses

Both Developer and Runtime Licenses can be used either locally on a single computer or in a network. Consequently, each ABBYY FineReader Engine license can have one of the following types:

- **Standalone** – for local work on a single computer;

- **Network** – licenses will be located on the server and passed down to workstations through the network.

Use and management of the licenses is performed with the License Manager utility.

Hardware and software protection keys

ABBYY FineReader Engine 10 will not function without a protection key. All ABBYY FineReader Engine licenses support two types of protection keys:

- **Hardware protection key** – This is a USB dongle that contains the license parameters. For the correct operation of the hardware protection key, you need to install the corresponding drivers. See the Installing the Hardware Key Drivers section for details.

- **Software protection key** – This is an activation file that should be obtained from the ABBYY server during a license activation process. Activation is carried out with the help of a special utility (the License Manager utility) which is supplied by ABBYY as an integral part of the ABBYY FineReader Engine package.

The details about use of protection keys and license activation you can find in the Activation section.

For additional licensing information, please contact the ABBYY office serving your region. You can find the list of ABBYY offices in the How to Buy section.

See also

Activation
ABBYY FineReader Engine 10 Modules

About ABBYY FineReader Engine 10 Activation

ABBYY FineReader Engine must be activated before use. If you have a Standalone license, you should activate ABBYY FineReader Engine on the same computer on which ABBYY FineReader Engine is installed. In the case of Network license, you should activate ABBYY FineReader Engine on a network server – a computer which will manage and distribute licenses among workstations in a network. However ABBYY FineReader Engine may be installed both on the network server and on workstations.

Both Standalone and Network licenses require License Service (LicensingService.exe) for correct operation of ABBYY FineReader Engine. The License Service can be installed automatically during the Developer installation and the Runtime installation of the ABBYY FineReader Engine library in automatic mode. If you need to install it manually, see for details Installing the License Service.

Note: The Licensing Service settings are provided in the LicensingSettings.xml file. The file is required for network installation and for standalone installation if Hardware protection key is used. This file is generated automatically during automatic installation. When installing manually, you must specify the correct settings in this file. The XML scheme of the settings is located in the LicensingSettings.xsd file. You can find this file in the Bin folder of the ABBYY FineReader Engine distribution package. The detailed description of the settings is provided in the Working with the LicensingSettings.xml File section.

For managing licenses ABBYY FineReader Engine provides the License Manager utility. With the help of this utility you can add, remove, activate, deactivate, update licenses and view license properties. The License Manager utility allows you to work with licenses with both types of protection keys:

- **Software protection key** – This is an activation file that should be obtained from the ABBYY server during an activation process.

- **Hardware protection key** – This is a USB dongle that contains the license parameters. In the case of a hardware protection key, license activation is not required.

If you choose a hardware protection key

If you choose the hardware protection key, the Hardware Key drivers must be installed on the computer where the License Service is installed. See the Installing the Hardware Key Drivers section for details. Once the installation is completed, connect the hardware protection key to the USB port of the computer. Make sure that you do it before the first run of the program. No license activation is required. To view license properties, use the License Manager utility.

If you choose a software protection key

A software protection key requires the activation of its serial number by means of the License Manager utility.

How is activation carried out?

Activation takes very little time and is carried out with the help of an **Activation Wizard**. This wizard is built into the License Manager utility. The Activation Wizard has a friendly interface and is used for sending the necessary activation information to ABBYY. The same wizard is used for loading the **ABBYY License File** (*.ABBYY.License file) which you receive from ABBYY during activation.

Activation information is sent as a code (Installation ID) which is generated on the basis of information about the computer on which the program is being installed. No personal information about the user or computer is used for generating this code and this code cannot be used for identifying the user.

Activation methods:

- **Via the Internet**
  Activation is carried out automatically and takes only a few seconds. An Internet connection is required for this type of activation.

- **By e-mail**
  The user needs to send an e-mail message generated by the program and containing information required for activation. To ensure a quick reply from the mail robot, do not alter the information in the message body or Subject field.

- **By e-mail from another computer**
  This method is suitable, if your computer does not have an Internet connection. The program will generate an e-mail message containing information required for activation and offer you to copy the message and send it to ABBYY from another computer.

In the case of activation via the Internet, the whole process is carried out automatically. In the case of activation by e-mail, the user needs to enter the path to the Activation File received from ABBYY in the corresponding field of the Activation Wizard.

Once the activation is complete, the program can be used.

Reactivation

ABBYY FineReader Engine 10 can be reinstalled on one and the same computer an unlimited number of times without reactivation. However, if you make major upgrades, format your hard drive, or reinstall the operating system on the computer where the License Service is installed, an additional activation may be required.

Deactivation

ABBYY FineReader Engine 10 license can be deactivated. The deactivated license can be then activated on another computer. The number of allowed deactivations can be restricted by your license.

Deactivation takes very little time and is carried out with the help of a **Deactivation Wizard**. This wizard is built into the License Manager utility. During the deactivation the Activation File (*.ABBYY.License file) which you receive from ABBYY during activation is deleted. Any copy of this file cannot be used for activation again.

The deactivation can be performed only via the Internet. Deactivation is carried out automatically and takes only a few seconds. An Internet connection is required. Once the deactivation is complete, the license can be activated on another computer.

License update

If you have purchased additional modules or an additional amount of pages for ABBYY FineReader Engine 10 and your license does not allow you to use them, you need to update the license. The license update process is similar to the activation process. The update process is carried out with the help of the **Update Wizard** and can be performed via the Internet or by e-mail. Once the update is complete, the newest functionality of the program can be used.

See also

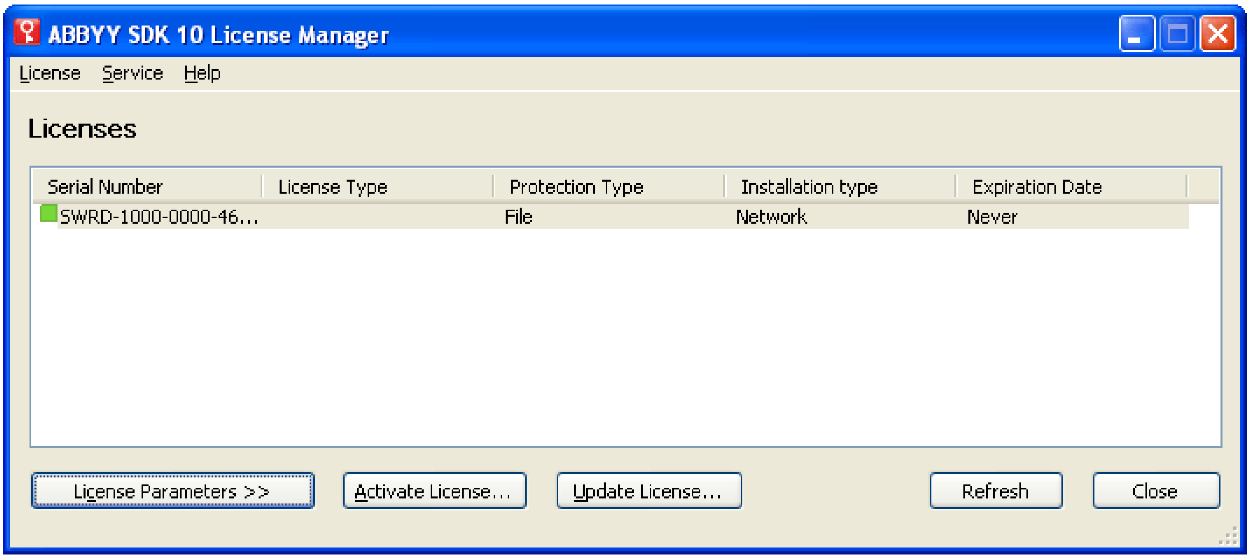
Licensing

License Manager Utility

The License Manager utility (LicenseManager.exe) allows you to manage ABBYY FineReader Engine licenses of all types. In the ABBYY SDK 10 License Manager dialog box you can activate, deactivate, or update license and view the properties of an activated license.

The License Manager utility is installed automatically during a Developer installation or during a Runtime ABBYY FineReader Engine library installation in automatic mode together with the License Service. This utility is accessible through **Start > Programs > ABBYY FineReader Engine 10 > License Manager** or in the **Bin** folder. This utility is distributed along with other ABBYY FineReader Engine 10 files allowed for distribution and is used for Runtime Licenses activation.

ABBYY SDK 10 License Manager dialog box



Serial Number	License Type	Protection Type	Installation type	Expiration Date
🟩 SWRD-1000-0000-46...		File	Network	Never

The following information about your ABBYY FineReader Engine 10 license is available in the **ABBYY SDK 10 License Manager** dialog box:

Column	Description
Serial number	The ABBYY FineReader Engine 10 serial number.
License type	The license type. For Developer's licenses the type of the license or the type of the emulated license is displayed depending on the license status.
Protection type	The protection type: • **File** — software protection key; • **Hardlock** — hardware protection key.
Installation type	The installation type: • **Standalone** — the license is used on a local computer;

	- **Network** — the license is located on a network computer.
Expiration date	The expiration date.

More details about the license you can find in the License Parameters table. To show or hide license parameters, use the **License Parameters/Hide License Parameters** button.

Activating, updating, or deactivating the license

To activate, update, or deactivate the license, press the corresponding button, or select the corresponding item in the menu, and follow the instructions in the dialog box that opens. See details about license activation, deactivation and update in the Activation section.

Buttons

- **License Parameters/Hide License Parameters**
  Shows or hides license parameters.

- **Activate license...**
  Starts the License Activation Wizard.

- **Update license...**
  Starts the License Update Wizard for the selected license.

- **Refresh**
  Updates the license list.

- **Close**
  Closes the License Manager.

Menu items

Item		Description
License	**Activate...**	Starts the License Activation Wizard.
	Update...	Starts the License Update Wizard for the selected license.
	Deactivate...	Starts the License Deactivation Wizard for the selected license.
	Copy Serial Number	Copies the selected license.
	Close	Closes the License Manager.
Service	**License Use Statistic...**	Shows the statistics of license usage on the workstations. Available only for the Network licenses with the CPU cores limitation.
	Refresh	Updates the license list.
Help	**Help**	Opens this help file.

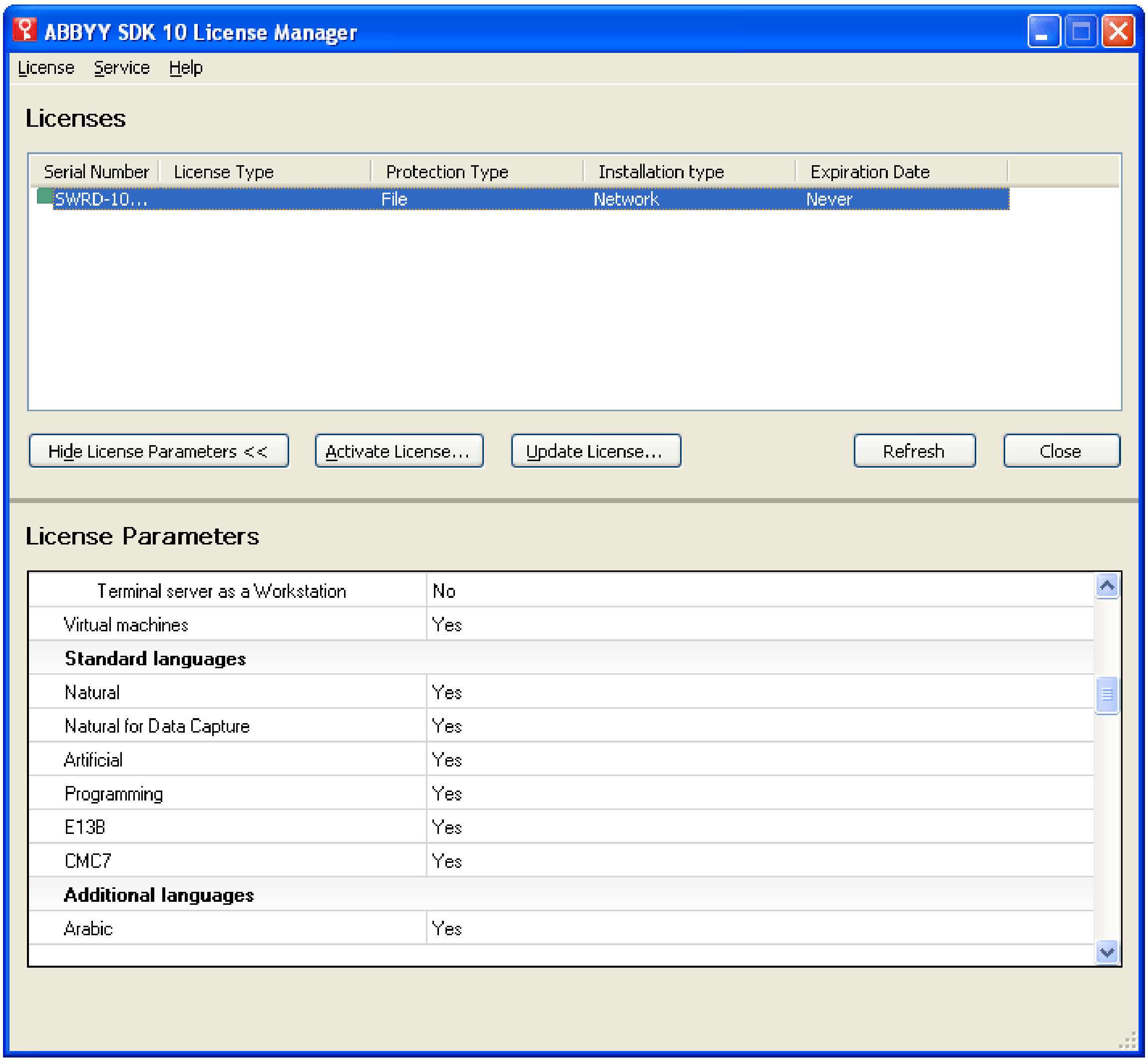
See also

Licensing
Activation

License Parameters

The license parameters are displayed in the table below the list of the licenses in the License Manager. To show or hide license parameters, use the **License Parameters/Hide License Parameters** button in the main window of the License Manager.

License Parameters/Hide License Parameters



Licenses

Serial Number	License Type	Protection Type	Installation type	Expiration Date
SWRD-10...		File	Network	Never

License Parameters

Terminal server as a Workstation	No
Virtual machines	Yes
Standard languages	
Natural	Yes
Natural for Data Capture	Yes
Artificial	Yes
Programming	Yes
E13B	Yes
CMC7	Yes
Additional languages	
Arabic	Yes

The License Parameters table provides information about your license and the mode of using the Developer's License (Developer/Runtime Emulated).

The following information about your ABBYY FineReader Engine 10 license is available:

- License type;
- Type of protection (software or hardware protection key);
- ABBYY FineReader Engine 10 serial number;
- Expiration date for your ABBYY FineReader Engine 10 license;
- Performance limitation: CPU core limit (the number of CPU Core which can be used for recognition), minimum number of CPU cores which can be used on a station, performance limit (e.g. characters per second);
- Environment limitation: usage in a network and on virtual machines;
- List of features that are allowed by your license (text types, export formats, additional modules, etc.).

See also

License Manager Utility

Working with the LicensingSettings.xml File

The LicensingSettings.xml file contains the ABBYY FineReader Engine protection settings. This file is necessary for correct work of the Licensing Service in the network. When Licensing Service is used on a local computer, this file is required if you use a Hardware protection key.

The file is generated automatically during Developer or Runtime installation in automatic mode. When installing manually, you must specify the correct settings in this file. The XML scheme of the settings is located in the LicensingSettings.xsd file. You can find both these files in the Bin folder of the ABBYY FineReader Engine distribution package.

Description of Tags

Tag	Type	Multiplicity	Parent Tag	Description
LicensingSettings	LicensingSettings. Elements: • LocalLicenseServer • LicensingServers	1	no	Protection settings.
LocalLicenseServer	LocalLicenseServerSettings. Elements: • ConnectionProtocol • EnableIKeyLicenses	0...1	LicensingSettings	The parameters of the connection with the local Licensing Service located on the same computer.
ConnectionProtocol	Complex Type. Attributes: • *ProtocolType* – the protocol type: LocalInterprocessCommunication, NamedPipes, or TCP/IP. 📝**Note:** This is an additional protocol type for the local License Service. It is not necessary to specify this protocol type for Standalone installation, as Standalone licenses are always used with the LocalInterprocessCommunication protocol type. • *EndPointName* – (optional)	0...1	LocalLicenseServer	The parameters of the connection protocol.
EnableIKeyLicenses	Complex Type. Attributes: • *Enable* – specifies whether Hardware protection keys can be used on the computer (set it to "yes" or "no")	0...1	LocalLicenseServer	Specifies whether Hardware protection keys can be used on the computer.
LicensingServers	Complex Type. Elements: • MainNetworkLicenseServer	0...1	LicensingSettings	The list of network servers where the Licensing Service is installed.
MainNetworkLicenseServer	NetworkServerAddress. Attributes: • *ServerAddress* – the DNS name or IP address of the computer where	1	LicensingServers	The parameters of the connection with the main network server

	the Licensing Service is installed. • *ProtocolType* – the protocol type: LocalInterprocessCommunication, NamedPipes, or TCP/IP. • *EndPointName* – (optional)			where the Licensing Service is installed.

See also

ABBYY FineReader Engine Distribution Kit
Distribution of Applications Which Use the ABBYY FineReader Engine Library
Installing the License Service

Installing the Hardware Key Drivers

The **Hardware Key** drivers must be installed before the USB key itself is plugged to the computer.

Warning! You must instruct your customers to close any iKey-dependent applications before running your installation program. If any of the iKey components are already on their computer and in use when the iKey installer is run, an incomplete iKey installation may result.

You should call the *Ikeydrvr.exe* installer program from your own installation program. The *Ikeydrvr.exe* is located in the **\USB Drivers** folder of your ABBYY FineReader Engine installation in the case of a 32-bit system, or **\USB Drivers\64** folder in the case of a 64-bit system.

Syntax:

Ikeydrvr.exe [self-extracting options] [installation options]

where

Self-extracting Options	
[-s]	Runs the self-extracting installer in silent mode. This option must precede the [-a] option, if defined. **Note:** Hardware Key drivers are installed in silent mode when you run “Ikeydrvr.exe -s” from the CD-ROM drive. Otherwise the attended installation is performed. To install Hardware Key drivers in silent mode from a hard drive, you can use the MSI installation. The Ikeydrvr.msi file can be downloaded from the SafeNet site (http://www.safenet-inc.com/Support_and_Downloads/Download_Drivers/iKey_Drivers.aspx). The installation instruction is distributed with the installation file.
[-a]	Specifies command line options for the Setup program. This option must be specified if any Installation options (see below) are specified.
Installation Options	
[LOGPATH=<path of log file>]	If this option is defined, a log file is created in the path specified by <path of log file>. See "Log File Format" later in this document for information about the format of the log file. The path defined must be an absolute path, without the trailing backslash character ('\'). It also must be defined as a short path (DOS 8.3). By default, if the LOGFILE option is not defined, the default log file name of IKASETUP.LOG, is used.
[LOGFILE=<log file name>]	If this option is defined, a log file is created in the path specified by LOGPATH, with the file name defined by <log file name>. This option requires the LOGPATH option to be defined. The file name must be defined as a short file name (DOS 8.3). See "Log File Format" later in this document for information about the format of the log file.

Installation Log File Format

Your applications can use the log file when spawning the installer from your own installation program. The log file is formatted as an .ini file and has the following format:

[InstallShield Silent]

File=Log File

[ResponseResult]

ResultCode=<Status Code>

[RequiredAction]

ActionCode=<Action Code>

[Application]

Name=iKey Components

Version=<version of installer>

Company=SafeNet

The <version of installer> value is formatted as follows: <major>.<minor>.<revision>.<build>

Example: Version=3.4.0.93 is Version 3.4.0 Build 93 of the installer.

Status codes can only be retrieved if the [LOGPATH] option is specified. Status codes define the status of the installation — warnings and error messages are logged using status codes.

Status Code	Description
0	The operation completed successfully.
1	The operation completed successfully. Changes will not be in effect until you restart your system
2	The operation completed successfully. A system restart is required to enable Smart Card Services.
3	The operation completed successfully. A power down of the system is required.
100	Warning, the iKey Device Driver has been installed with Smart Card Services disabled.
101	Warning, the iKey Device Driver has been installed with Smart Card Services disabled. A system restart is required to complete the installation.
102	Warning, the maximum number of readers supported by this platform has been exceeded. The iKey Device Driver has been installed with Smart Card Services disabled.
103	Warning, the maximum number of readers supported by this platform has been exceeded. The iKey Device Driver has been installed with Smart Card Services disabled. A system restart is required to complete the installation.
200	Error, administration privileges are required to install this product.
201	Error, this system does not support USB devices.
202	Error, the version of this operating system is not supported.
203	Error, installation canceled.
204	Error, invalid command line option.
205	Error, another vendors' NT 4.0 USB stack exists. The user must uninstall this USB stack before installing this product.
206	Error, one or more services are marked for deletion. A system restart is required prior to installing this program.
207	Error, must uninstall previous version of package.
-1	Error, installation failed. (General error.)

Action codes can only be retrieved if the [LOGPATH] option is specified. Action codes define the actions you, the developer, must take after the iKey Installers have finished.

Action Code	Description
0	No action required.
1	Insert a token to complete the installation. Windows Device Notification events CANNOT be used to wait for token insertion.
2	Insert a token to complete the installation. Windows Device Notification events CAN be used to wait for token insertion.
3	Re-insert the token to complete the installation. Windows Device Notification events CANNOT be used to wait for token insertion.
4	Re-insert the token to complete the installation. Windows Device Notification events CAN be used to wait for token insertion.
5	Must remove all tokens to install or uninstall the product.

Note: Connect an iKey to a USB port on the computer after rebooting at the end of the iKey drivers installation.

See also

Activation
Distribution

ABBYY FineReader Engine 10 Modules

The functionality of ABBYY FineReader Engine 10 is represented by a set of modules. Each module is a group of Engine functions. Some modules can be included in ABBYY FineReader Engine licenses as predefined modules and others as additional ones.

For additional licensing information, please contact the ABBYY office serving your region. You can find the list of ABBYY offices in the How to Buy section.

An ABBYY FineReader Engine 10 License allows you to process a certain number of pages per period (usually per month). This means that the user can process (analyze, recognize, or export to any format) no more pages than is allowed by the user's license. The counter is incremented by 1 when processing an A4 page or smaller. When processing a page which is n times larger than A4, the counter will be incremented by n.

The modules available in ABBYY FineReader Engine 10 are listed in the table below.

Module	Description
Standard Languages	
Natural	This module provides access to the all languages supported by ABBYY FineReader Engine except the ones defined in special groups (see below).
Natural for Data Capture	This module is currently not supported.
Artificial	This module provides access to the Esperanto, Ido, Interlingua, Occidental recognition languages.
Programming	This module provides access to the following recognition languages: Basic, C/C++, COBOL, Fortran, Java, Pascal.
E13B	This module provides access to E13B language and MICR text type (TextTypeEnum::TT_MICR_E13B).
CMC7	This module provides access to CMC7 language and MICR text type (TextTypeEnum::TT_MICR_CMC7).
Additional Languages	
Arabic	This module provides access to the Arabic recognition language.
Chinese	This module provides access to the Chinese (PRC), Chinese (Taiwan) recognition languages.
Japanese	This module provides access to the Japanese recognition language.
Korean	This module provides access to the Korean, Korean (Hangul) recognition language.
FineReader XIX	This module provides access to Gothic text type (TextTypeEnum::TT_Gothic), Latvian language written in Gothic script and Old European languages: Old English, Old French, Old German, Old Italian, and Old Spanish.
Thai	This module provides access to the Thai recognition language.
Vietnamese	This module provides access to the Vietnamese recognition language.
Hebrew	This module provides access to the Hebrew recognition language.
Yiddish	This module provides access to the Yiddish recognition language.
User (Custom) OCR Languages	This module provides access to creating, editing, and using user languages. If this module is not available, the only way to set the recognition language is to use the IRecognizerParams::SetPredefinedTextLanguage method.
OCR fonts	
Matrix	This module provides access to Matrix text type (TextTypeEnum::TT_Matrix)
Normal	This module provides access to Normal text type (TextTypeEnum::TT_Normal)
Advanced	This module provides access to Normal text type with low resolution (IRecognizerParams::LowResolutionMode)
OCR A	This module provides access to OCR-A text type (TextTypeEnum::TT_OCR_A)
OCR B	This module provides access to OCR-B text type (TextTypeEnum::TT_OCR_B)
Typewriter	This module provides access to Typewriter text type (TextTypeEnum::TT_Typewriter)

User Patterns			This module allows you to perform recognition with user patterns and train user patterns using the IEngine::TrainUserPattern method. In order user patterns training and editing via the GUI elements are available, your license must support the User Patterns Training module. **Note:** Pattern training is not supported for hieroglyphic languages.
Data Capture (ICR/OMR)			
ICR			This module provides access to Handprinted text type (TextTypeEnum::TT_Handprinted)
Cyrillic ICR			This module allows you to recognize Cyrillic hand-printed texts.
Index			This module provides access to Index text type (TextTypeEnum::TT_Index)
OMR			This module allows you to recognize checkmarks.
Barcodes			
Barcode Autolocation			This module provides access to the IFRPage::ExtractBarcodes, IDocumentAnalyzer::ExtractBarcodes methods, and *BarcodeRecognition* profile. **Note:** This module can be used if barcodes of any type are available.
1D Barcodes			This module provides access to recognition of 1D barcodes. If this module is included, one-dimensional barcodes can be recognized in the following ways: • Create a barcode block manually, set the required parameters and then call one of the recognition methods that does not perform layout analysis (e.g. the IFRPage::Recognize, IFRPage::RecognizeBlocks). • Analyze the page and detect the barcodes by setting the IPageProcessingParams::DetectBarcodes property to TRUE (only if the *Document Analysis* module is available), then call one of the recognition methods that does not perform layout analysis (e.g. the IFRPage::Recognize, IFRPage::RecognizeBlocks). • Analyze and recognize the page (e.g. using the AnalyzeAndRecognizePage, RecognizeImageFile (only if the *Document Analysis* module is available), RecognizeImageAsPlainText, or RecognizeImageDocumentAsPlaintText method). To detect barcodes, set the IPageProcessingParams::DetectBarcodes property to TRUE. • Call the ExtractBarcodes method (only if the *Barcode Autolocation* module is available)
2D Barcodes	*Aztec*		This module provides access to recognition of 2D barcodes of type Aztec. If this module is included, the barcodes can be recognized in the same way as 1D barcodes (see description of *1D Barcodes* module).
	DataMatrix		This module provides access to recognition of 2D barcodes of type Data Matrix. If this module is included, the barcodes can be recognized in the same way as 1D barcodes (see description of *1D Barcodes* module).
	PDF417		This module provides access to recognition of 2D barcodes of type PDF 417. If this module is included, the barcodes can be recognized in the same way as 1D barcodes (see description of *1D Barcodes* module).
	QR Code		This module provides access to recognition of 2D barcodes of type QR Code. If this module is included, the barcodes can be recognized in the same way as 1D barcodes (see description of *1D Barcodes* module).
	MaxiCode		This module is currently not supported.
PDF Support			
PDF Opening			This module allows you to process PDF files.
PDF Export	*Modes*	*ImageOnly*	This mode allows you to export to PDF Image Only format. Recognition is not required, it is enough to open an image and then export it to PDF Image Only setting IPDFExportParams::TextExportMode to PEM_ImageOnly and using FileExportFormatEnum::FEF_PDF to select the export format. If PDF/A module is available, then this mode also allows you to export to PDF/A Image Only.
		All	This mode allows you to export to PDF file format (FileExportFormatEnum::FEF_PDF), including PDF Image Only. If PDF/A module is available, then this mode also allows you to export to PDF/A in all modes.
	PDF/A		This module allows you to export to PDF/A file format (FileExportFormatEnum::FEF_PDFA). Availability of export modes depends on the value of the *Modes* parameter.
	MRC		This module allows you to tune Mixed Raster Content parameters during export to PDF (PDFExportParams::MRCMode, IPDFExportParamsOld::MRCParams). If PDF/A module is

		available, then this module also allows you to export to PDF/A with MRC (PDFExportParams::MRCMode, IPDFAExportParamsOld::MRCParams).
Export		
MS Office	*RTF, DOC, DOCX*	This module allows you to export to RTF, DOC, DOCX file formats (FileExportFormatEnum::FEF_RTF, FileExportFormatEnum::FEF_DOCX).
	XLS, XLSX	This module allows you to export to XLS, XLSX file format (FileExportFormatEnum::FEF_XLS, FileExportFormatEnum::FEF_XLSX).
	PPTX	This module allows you to export to PPTX file format (FileExportFormatEnum::FEF_PPTX).
HTML		This module allows you to export to HTML file format (FileExportFormatEnum::FEF_HTML).
Text		This module allows you to export to TXT file format (FileExportFormatEnum::FEF_Text).
ABBYY XML		This module allows you to export to XML file format (FileExportFormatEnum::FEF_XML).
Open Office Document (ODT)		This module is currently not supported.
FB2		This module is currently not supported.
EPUB		This module is currently not supported.
ALTO		This module is currently not supported.
Extended Character Info		This module provides access to the following properties and methods: • the BaseLine, Color, FontName, FontSize, FontType, HorizontalScale, IsBold, IsItalic, IsSmallCaps, IsStrikeout, IsSubscript, IsSuperscript, IsUnderlined, Spacing, CharacterRecognitionVariants, CharacterRecognitionVariantIndex, SelectedCharacterRecognitionVariant, WordRecognitionVariants properties of the CharParams object; • the GetWordRecognitionVariants method of the Paragraph object; • the BaseLine property of the ParagraphLine object; • the MeanStrokeWidth property of the WordRecognitionVariant object; • the SerifProbability properties of the CharacterRecognitionVariant object; • the WriteWordRecognitionVariants and WriteCharacterRecognitionVariants properties of the XMLExportParams object.
Processing		
Document Analysis		This module provides access to the Layout object obtained as a result of automatic analysis of the document. The following methods for analysis and recognition are available: Analyze***, RecognizeImageFile. **Note:** If this module is not available, you create a Layout object manually, add blocks to it and recognize the page.
DA for Full-Text Indexing		This module is used to extract data from a document, including text in pictures. Note that the program retains both the picture and the text in it. Text extracted from a picture block can only be exported to XML, TXT and PDF formats. This data is used for later full-text indexing and search. The program retains the logical reading order, pictures and tables. This module provides access to the IObjectsExtractionParams::FullTextIndexDA property.
DA for Invoices		This module is used to preprocess invoices. Usually they are noisy, low-quality images. This mode extracts all text from the image, including tables, pictures, small text areas, and noise. The result is plain text without table blocks and picture blocks. This module provides access to the IObjectsExtractionParams::FlexiFormsDA property.
Balanced Mode		This module provides access to the IRecognizerParams::BalancedMode property.
Fast Mode		This module provides 2-2.5 times faster recognition speed at the cost of a moderately increased error rate (1.5-2 times more errors). This module provides access to the IRecognizerParams::FastMode property.
Camera OCR		This module provides access to: • Blurred images correction (IImageDocument::RemoveCameraBlur) • ISO noise reduction (IImageDocument::RemoveCameraNoise)
Color Filtering		This module provides access to image color filtering (IImageDocument::RemoveColorObjects).

ASCII License Basic Modules	This module allows you to export to ASCII XML file format. There are two ways of exporting to ASCII XML: • Using the IPlainText::SaveToAsciiXMLFile method; • Setting the IXMLExportParams::WriteCharAttributes property to XCA_Ascii and using FileExportFormatEnum::FEF_XML to select the export format
Visual Components	
Image Viewing and Blocks Drawing	This module is currently not supported.
Document Batch Managing	This module is currently not supported.
Text Viewing and Editing	This module is currently not supported.
Full-Text Verification	This module is currently not supported.
Scanning	This module provides access to the scanning interfaces of ABBYY FineReader Engine (ScanOptionsInterfaceTypeEnum::SOIT_Twain).
User Patterns Training	This module allows you to train user patterns and edit them via the GUI elements provided by ABBYY FineReader Engine (IEngine::EditUserPattern method and User Pattern dialog box, IRecognizerParams::TrainUserPatterns property set to TRUE and Pattern Training dialog box).

License-related errors

When unavailable method is called, unsupported value is assigned to an object property or passed as an argument to an object method, the operation will fail, and the CO_E_NOT_SUPPORTED error code will be returned.

When number of processed pages exceeds the value allowed by **Limited** modification, the analysis and recognition methods will fail, and the E_FAIL error code will be returned.

When the license has been expired or not loaded, only the **StartLogging**, **StopLogging** methods and the **CurrentLicense** and **Licenses** properties of the **Engine** object are available. Other methods of the **Engine** object will return CLASS_E_NOTLICENSED error code.

See also

Licensing

Copyright and Trademark Notices

ABBYY FineReader Engine 10 is a version of ABBYY FineReader intended for developers.

© 2010 ABBYY. All rights reserved
ABBYY, FINEREADER, and ABBYY FineReader are either registered trademarks or trademarks of ABBYY Software Ltd.

This program is built on proprietary ABBYY technologies but also includes a number of third-party solutions:

- Windows® is a registered trademark of Microsoft Corporation in the United States and other countries

- Adobe PDF Library is used for opening and processing PDF files:
  © 1984-2008 Adobe Systems Incorporated and its licensors. All rights reserved.
  Protected by U.S. Patents 5,929,866; 5,943,063; 6,289,364; 6,563,502; 6,185,684; 6,205,549; 6,639,593;7,213,269; 7,246,748; 7,272,628; 7,278,168; 7,343,551; 7,395,503; 7,389,200; 7,406,599;6,754,382; Patents Pending.
  Adobe®, the Adobe logo, Acrobat®, the Adobe PDF logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. All other trademarks are the property of their respective owners.

- Using Type 1 fonts for exporting to PDF format:
  © 2001 ParaType Inc.
  © 2003 ParaType Inc.

- Opening DjVu image format:
  Portions of this computer program are copyright © 1996-2007 LizardTech, Inc. All rights reserved. DjVu is protected by U.S. Patent No. 6,058,214. Foreign Patents Pending.

- Working with JPEG image format:
  This software is based in part on the work of the Independent JPEG Group.

- Unicode support:
  © 1991-2009 Unicode, Inc. All rights reserved.

- Intel® Performance Primitives:
  Copyright © 2002-2008 Intel Corporation.
- Font support:
  Portions of this software are copyright © 1996-2002, 2006 The FreeType Project (www.freetype.org). All rights reserved.
- U.S. Patent Nos. 5,258,855, 5,369,508, 5,625,465, 5,768,416 and 6,094,505.

Copyright and Trademark Notices for APPLICATION's Help

You should include these copyright and trademark notices to the Help file of your application:

The application contains recognition technologies of ABBYY® FineReader® Engine 10 for Windows® © 2010.

ABBYY, FINEREADER, and ABBYY FineReader are either registered trademarks or trademarks of ABBYY Software Ltd.

Windows® is a registered trademark of Microsoft Corporation in the United States and other countries

Adobe PDF Library is used for opening and processing PDF files:

© 1984-2008 Adobe Systems Incorporated and its licensors. All rights reserved.
Protected by U.S. Patents 5,929,866; 5,943,063; 6,289,364; 6,563,502; 6,185,684; 6,205,549; 6,639,593;7,213,269; 7,246,748; 7,272,628; 7,278,168; 7,343,551; 7,395,503; 7,389,200; 7,406,599;6,754,382; Patents Pending.
Adobe®, the Adobe logo, Acrobat®, the Adobe PDF logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. All other trademarks are the property of their respective owners.

Opening DjVu image format:

Portions of this computer program are copyright © 1996-2007 LizardTech, Inc. All rights reserved. DjVu is protected by U.S. Patent No. 6,058,214. Foreign Patents Pending.

Working with JPEG image format:

This software is based in part on the work of the Independent JPEG Group.

Unicode support:

© 1991-2009 Unicode, Inc. All rights reserved.

Intel® Performance Primitives:

Copyright © 2002-2008 Intel Corporation.

Font support:

Portions of this software are copyright © 1996-2002, 2006 The FreeType Project (www.freetype.org). All rights reserved.

Copyright and Trademark Notices for APPLICATION's Marketing Materials

The application contains recognition technologies of ABBYY® FineReader® Engine 10 for Windows® © 2010.

ABBYY, FINEREADER, and ABBYY FineReader are either registered trademarks or trademarks of ABBYY Software Ltd.

Windows® is a registered trademark of Microsoft Corporation in the United States and other countries

Adobe PDF Library is used for opening and processing PDF files:

© 1984-2008 Adobe Systems Incorporated and its licensors. All rights reserved.
Protected by U.S. Patents 5,929,866; 5,943,063; 6,289,364; 6,563,502; 6,185,684; 6,205,549; 6,639,593;7,213,269; 7,246,748; 7,272,628; 7,278,168; 7,343,551; 7,395,503; 7,389,200; 7,406,599;6,754,382; Patents Pending.
Adobe®, the Adobe logo, Acrobat®, the Adobe PDF logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. All other trademarks are the property of their respective owners.

Opening DjVu image format:

Portions of this computer program are copyright © 1996-2007 LizardTech, Inc. All rights reserved. DjVu is protected by U.S. Patent No. 6,058,214. Foreign Patents Pending.

Working with JPEG image format:

This software is based in part on the work of the Independent JPEG Group.

Intel® Performance Primitives:

Copyright © 2002-2008 Intel Corporation.

Font support:

The minimum terms of End User License Agreement (EULA)

The following terms, or substantially similar terms, are required to be included in Sublicensee's End User License Agreement for products with ABBYY FineReader Engine integrated in:

- **EULA Terms**
  - The EULA must include the following terms and conditions governing the use of the ABBYY SDK or the APPLICATION as a whole:
  - The End User is granted a Runtime License for the ABBYY SDK contained in the APPLICATION on condition that the End User complies with the terms and conditions of the EULA which apply to the ABBYY SDK or to the APPLICATION as a whole. The Runtime License may be time-, performance- or function-limited and protected from unauthorized copying by means of a hardware or software protection key which is an integral part of the ABBYY SDK.
  - The End User may not perform or make it possible for other persons to perform any activities included in the list below:
    - Disassemble or decompile (i.e. extract the source code from the object code) the ABBYY SDK (applications, databases, and other ABBYY SDK components), except, and only to the extent, that such activity is expressly permitted by applicable law notwithstanding this limitation.
    - Modify the ABBYY SDK, including making changes to the object code of the applications and databases contained in the ABBYY SDK other than those provided for by the ABBYY SDK and described in the documentation.
    - Transfer any rights granted to the End User hereby and other rights related to the ABBYY SDK to any other person, not authorized to use the ABBYY SDK.
    - Make it possible for any person not entitled to use the ABBYY SDK and working in the same multi-user system as the End User to use the ABBYY SDK.
  - ABBYY SDK is supplied “as is.” ABBYY does not guarantee that the ABBYY SDK will carry no errors, nor will it be liable for any damages, either direct or indirect, including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss resulting from the use of ABBYY SDK, or damages caused by possible errors or misprints in the ABBYY SDK.
  - Export Rules. If purchased in the United States, the ABBYY SDK shall not be exported or re-exported in violation of any export provisions of the United States or any other applicable legislation.
  - If any part of the EULA is found void and unenforceable, it will not affect the validity of the balance of the EULA, which shall remain valid and enforceable according to its terms. The EULA shall not prejudice the statutory rights of any party dealing as a consumer.
- **ADOBE PDF LIBRARY EULA TERMS**
  - If the APPLICATION(s) includes ABBYY SDK parts which contain components of Adobe® PDF Library™ (APDFL) functionality then the Developer must comply with APDFL terms and conditions stated in this paragraph below (provided that “you” means the End User):
    - **Adobe® PDF Library**. “Adobe Software” means Adobe® PDF Library for Windows NT, 2000, XP, 98, Me and related documentation, and any upgrades, modified versions, updates, additions, and copies thereof. The ABBYY SDK uses the Adobe Software for converting PDF files into image files.
    - **License Grant and Restrictions**. ABBYY grants you a non-exclusive right to use the Adobe Software incorporated into the ABBYY SDK under the terms of this EULA. You may make one backup copy of the Adobe Software incorporated into the ABBYY SDK, provided the backup copy is not installed or used on any computer.

  - **Intellectual Property Rights**. The Adobe Software incorporated into the ABBYY SDK is owned by Adobe and its suppliers, and its structure, organization and code are the valuable trade secrets of Adobe and its suppliers. The Adobe Software is also protected by the United States Copyright Law and International Treaty provisions. You may not copy the Adobe Software incorporated into ABBYY SDK, except as provided in this EULA. Any copies that you are permitted to make pursuant to this EULA must contain the same copyright and other proprietary notices that appear on or in Adobe Software and the ABBYY SDK. You agree not to modify, adapt, translate, reverse engineer, decompile, disassemble or otherwise attempt to discover the source code of the Adobe Software incorporated into ABBYY SDK. Except as stated above, this EULA does not grant you any intellectual property rights in the Adobe Software.

  - **Font License**. If Adobe Software incorporated into the ABBYY SDK includes font software, you may embed the font software, or outlines of the font software, into your electronic documents to the extent that the font vendor copyright owner allows for such embedding. The fonts contained in this package may contain both Adobe and non-Adobe owned fonts. You may fully embed any font owned by Adobe.

  - **Warranty**. ABBYY AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE RESULTS YOU MAY OBTAIN BY USING THE ADOBE SOFTWARE INCORPORATED INTO THE ABBYY SDK.

  - THE FOREGOING STATES THE SOLE AND EXCLUSIVE REMEDIES FOR ABBYY'S BREACH OF WARRANTY. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, ADOBE AND ITS SUPPLIERS MAKE NO WARRANTY, EXPRESS OR IMPLIED AS TO MERCHENTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR NON-INFRIGEMENT. IN NO EVENT WILL ADOBE OR ITS SUPPLIERS BE LIABLE TO YOU FOR ANY CONSEQUENTAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, EVEN IF AN ADOBE REPRESENTATIVE HAS BEEN ADVICED OF POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY.
    Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to you. To the extent permissible, any implied warranties are limited to thirty (30) days. This warranty gives you specific legal rights. You may have other rights, which vary from state to state or jurisdiction to jurisdiction.

  - **Export Rules**. You agree that the Adobe Software incorporated into the ABBYY SDK will not be shipped, transferred or exported into any country or used in any manner prohibited by the United States Export Administration Act or any other export laws, restrictions or regulations (collectively the "Export Laws"). In addition, if the Adobe Software incorporated into the ABBYY SDK is identified as export controlled items under the Export Laws, you represent and warrant that you are not a citizen, or otherwise located within, an embargoed nation and that you are not otherwise prohibited under the Export Laws from receiving the Adobe Software incorporated into ABBYY SDK. All rights to use the Adobe Software incorporated into the ABBYY SDK are granted on condition that such rights are forfeited if you fail to comply with the terms of this EULA.

  - **Trademarks**. Adobe and Adobe PDF Library are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

- **LIZARDTECH EULA Terms**

  The APPLICATION(s) includes ABBYY SDK parts which contain software licensed by ABBYY from LIZARDTECH, INC and the Developer must include undermentioned terms and conditions and comply with terms and conditions stated in this paragraph below (provided that "you" means the End User):

  - You have acquired a product ("PRODUCT") that includes software licensed by ABBYY from LIZARDTECH, INC. Those installed software products of LIZARDTECH origin, as well as any associated media, printed materials, and "online" or electronic documentation ("SOFTWARE") are protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE is licensed, not sold.

  - If you do not agree to this End User License Agreement ("EULA"), do not use the PRODUCT. Promptly contact ABBYY for instructions on return of the unused PRODUCT(S) for a refund. Any use of the SOFTWARE, including but not limited to use of the PRODUCT, will constitute your agreement to this EULA (or ratification of any previous consent).

  - Grant of License. You are granted a personal, nonsublicensable, nontransferable, nonexclusive license to use the SOFTWARE as integrated in the PRODUCT (as well as any associated documentation). You will not rent, sell, lease or otherwise distribute the SOFTWARE or any part of it.

  - NO WARRANTIES FOR THE SOFTWARE. The SOFTWARE is provided "AS IS" and with all faults. THE ENTIRE RISK AS TO SATISFACTORY QUALITY, PERFORMANCE, ACCURACY, AND EFFORT (INCLUDING LACK OF NEGLIGENCE) IS WITH YOU. ALSO, THERE IS NO WARRANTY AGAINST INTERFERENCE WITH YOUR ENJOYMENT OF THE

SOFTWARE OR AGAINST INFRINGEMENT. IF YOU HAVE RECEIVED ANY WARRANTIES REGARDING THE PRODUCT OR THE SOFTWARE, THOSE WARRANTIES DO NOT ORIGINATE FROM, AND ARE NOT BINDING ON, LIZARDTECH.

- NO LIABILITY FOR CERTAIN DAMAGES. EXCEPT AS PROHIBITED BY LAW, LIZARDTECH SHALL HAVE NO LIABILITY FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES ARISING FROM OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE SOFTWARE. THIS LIMITATION SHALL APPLY EVEN IF ANY REMEDY FAILS OF ITS ESSENTIAL PURPOSE.

- Limitations on Reverse Engineering, Decompilation, and Disassembly. You may not reverse engineer, decompile, or disassemble the SOFTWARE, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

- Export Restrictions. You acknowledge that the SOFTWARE, or any part thereof, or any process or service that is the direct product of the SOFTWARE (the foregoing collectively referred to as the “Restricted Components”) are of U.S. origin. You agree to comply with all applicable international and national laws that apply to these products, including the U.S. Export Administration Regulations, as well as end-user, end-use and destination restrictions issued by U.S. and other governments.

Distribution of Applications Which Use the ABBYY FineReader Engine Library

You developed an application using the ABBYY FineReader Engine functions inside and you want to distribute this application. In this case you need a Runtime License for distribution. If you plan your application to work locally on a single computer, you must have a Standalone Runtime License. If your applications work in a network, you will need a Network Runtime License. The Runtime License should correspond to the Developer License under which your application was compiled. The serial number of the Developer License is passed as the input parameter of the **GetEngineObject** function.

Distribution of applications using the ABBYY FineReader Engine library includes two steps:

1. Installing the application and the ABBYY FineReader Engine library on the local disk of the workstation
2. Activating the ABBYY FineReader Engine Library with the Runtime License

The ABBYY FineReader Engine distribution package includes the System Administrator's Guide. This guide contains complete information about local and network distribution of applications which use the ABBYY FineReader Engine library.

See also

ABBYY FineReader Engine Distribution Kit
Licensing

Installing the ABBYY FineReader Engine Library

The first step of distribution of applications using the ABBYY FineReader Engine library is installation of the application and the ABBYY FineReader Engine library on the local disk of the workstation.

After installing your application on a workstation, you should install the ABBYY FineReader Engine library. It may be installed in automatic or manual mode. See for details:

- Installing the ABBYY FineReader Engine Library in Automatic Mode
- Installing the ABBYY FineReader Engine Library in Manual Mode

On the workstation, the following components should be installed:

- Microsoft® Internet Explorer 5.0 or higher
- If your application uses any of the ABBYY FineReader Engine methods producing user interface elements (dialogs), e.g. Pattern Training, User Pattern, Dictionary dialogs
    1. Windows Common Controls must have version 5.80 or later.
    2. Rich Edit Control must have version 3.0 or later.

The following folders and registry branches should be accessible from the workstation:

- folder with ABBYY FineReader Engine binary files – full control
- %TEMP% folder – full control
- %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\FineReader Engine – full control
- "HKEY_CURRENT_USER\Software\ABBYY\SDK\10\FineReader Engine" – full control
- "HKEY_CURRENT_USER\Software\ABBYY\SDK\10" – full control for installation only
- "HKEY_LOCAL_MACHINE\Software\ABBYY\SDK\10" – full control for installation only

⚠**Important!** Never redistribute ABBYY FineReader Engine type library and files with API description (FREngine.tlb, FREngine.h, FREngine_i.c).

See also

Installing the ABBYY FineReader Engine Library in Automatic Mode
Installing the ABBYY FineReader Engine Library in Manual Mode
ABBYY FineReader Engine Distribution Kit
Licensing

Installing the ABBYY FineReader Engine Library in Automatic Mode

The runtime installation of the ABBYY FineReader Engine library in automatic mode can be performed only from the command line in silent mode. Run the setup.exe file from the installation CD-ROM with the command line options described below.

Option	**Default Value**	**Description**
INSTALLDIR="<destination path>"	%ProgramFiles%\ABBYY SDK\10\FineReader Engine\	The path to the folder where the ABBYY FineReader Engine library will be installed.
SN=<serial number>		The ABBYY FineReader Engine 10 serial number
MODULES=<*list of library modules*> | None | All	All	The list of library modules that you want to install. The list of available modules see below. The modules must be separated by a comma (,). For example, MODULES=ICR,PDF,BasicLang. If you do not want to install any of these modules, set this option to No. If you want to install all the modules, set this option to All.
IKEYDR = Yes | No	No	Specifies whether hardware key drivers must be installed.
LICENSESRV = Yes | No	Yes	Specifies whether License Service must be installed. If the IKEYDR = Yes, the License Service is installed automatically and cannot be excluded from the installation. ⚠**Important!** If you have a Standalone license, you should install the License Service on the same computer on which ABBYY FineReader Engine is installed. In the case of Network license, you should install the License Service on a network server – a computer which will manage and distribute licenses among workstations in a network.
SERVERNAME=<the DNS name or IP address>		The DNS name or IP address of the computer where the License Service is installed.
/v		The start of the installation. This is mandatory option.
/q		Silent mode. This is mandatory option because the runtime installation can be performed only in silent mode. Use the **/qb** option if you want a progress bar to be displayed during the installation. No other dialog boxes will be displayed.

Library modules

Each library module determines the license modules which must be available in a Runtime License, and resource files which will be installed (see the ABBYY FineReader Engine Distribution Kit). The license modules and resource files corresponding to each library module are listed in the table below:

Library module	**The license modules which must be available in a Runtime License**	**The resource files which will be installed**
ICR	Index, Handprinted, OMR	The files for recognition of checkmarks and handprinted text.
PDF	PDF Opening	The files which are listed in the ABBYY FineReader Engine Distribution Kit: PDF section.
VC	Scanning, User Patterns Training	The files for scanning and user patterns training.
BasicLang	Natural	The files for basic predefined languages, except the ones defined in special groups.
DataCaptureLang	Natural for Data Capture	The module is currently not supported.
Arabic	Arabic	The files for recognition of texts in Arabic language.

Chinese	Chinese	The files for recognition of texts in Chinese language.
Japan	Japanese	The files for recognition of texts in Japanese language.
Korean	Korean	The files for recognition of texts in Korean language.
FRXIX	FineReader XIX	The files for recognition of texts in Old European languages.
Hebrew	Hebrew	The files for recognition of texts in Hebrew language.
Thai	Thai	The files for recognition of texts in Thai language.
Vietnamese	Vietnamese	The files for recognition of texts in Vietnamese language.

Note: When you use silent mode, the /q option must precede the /v option, for example: setup.exe /q /v or setup.exe /qb /v

For example

```
setup.exe /q /v MODULES=PDF,ICR IKEYDR = Yes SN=XXXX-XXXX-XXXX-XXXX-XXXX
```

This command line will install (in silent mode) the PDF and ICR library modules into the %ProgramFiles%\ABBYY SDK\10\FineReader Engine\ folder using the serial number XXXX-XXXX-XXXX-XXXX-XXXX of the Standalone Runtime License. The hardware protection key will be used.

```
setup.exe /qb /v INSTALLDIR="C:\MyFolder" SN=XXXX-XXXX-XXXX-XXXX-XXXX
```

This command line will install (in silent mode) all library modules into the C:\MyFolder folder using the serial number XXXX-XXXX-XXXX-XXXX-XXXX of the Standalone Runtime License, a progress bar will be displayed. The software protection key will be used.

```
setup.exe /q /v SERVERNAME=MyServer
```

This command line will install all library modules into the %ProgramFiles%\ABBYY SDK\10\FineReader Engine\ folder in silent mode, and the Network Runtime License is stored on the MyServer computer.

See also

Distribution
ABBYY FineReader Engine Distribution Kit
Modules

Installing the ABBYY FineReader Engine Library in Manual Mode

To install the ABBYY FineReader Engine library in manual mode, please do the following:

- Copy files marked as "mandatory" in the table of the ABBYY FineReader Engine Distribution Kit section. They are system modules and main recognition databases.
- Copy recognition databases for handprinted text, if you want to recognize handprinted text.
- Copy resource files for interface languages that will be used in your application.
- Copy dictionary support files for recognition languages that your application will support. If the recognition languages include languages with the Latin alphabet, make sure that you copy the Univers.amd and Univers.amm files.
- Copy scanning modules, scanning-specific resources and Twain modules if your application will perform scanning via the ABBYY FineReader Engine interface.
- Create the %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\FineReader Engine folder. All FineReader Engine users must have read and write permissions to this folder.

Important! Never redistribute ABBYY FineReader Engine type library and files with API description (FREngine.tlb, FREngine.h, FREngine_i.c).

After you have copied all necessary ABBYY FineReader Engine library files, you need to activate the library.

See also

Distribution
Installing the ABBYY FineReader Engine Library
Activating the Runtime License

Activating the ABBYY FineReader Engine Library with the Runtime License

The second step of distribution of applications using the ABBYY FineReader Engine library is activation of the library with the Runtime License.

If you use a Standalone Runtime License, activation is performed on the workstation where the ABBYY FineReader Engine library is installed. If you use a Network Runtime License, you should activate ABBYY FineReader Engine on a network server – a computer which will manage and distribute licenses among workstations in a network.

You will need to use one of the variants described in the table below. The selection depends on the type of your protection key and the mode of the ABBYY FineReader Engine library installation.

Protection Key	**Mode of Library Installation**	
	Automatic	**Manual**
software	1. Run the License Manager utility. 2. Activate a license.	1. Install the License Service. 2. Run the License Manager utility and activate a license.
hardware	License activation is not required. Connect the hardware protection key to the USB port of the computer. You can view license properties with the License Manager utility.	1. Copy the ikeydrvr.exe file into the ..\USB Drivers subfolder of your application root folder in the case of a 32-bit system, or ..\USB Drivers\64 subfolder in the case of a 64-bit system. 2. Install the Hardware Key driver (see Installing Hardware Key Drivers for details). 3. Install the License Service. 4. License activation is not required. Connect the hardware protection key to the USB port of the computer. You can view license properties with the License Manager utility.

Changing the Type of the Protection Key

If you changed a software protection key to a hardware protection key, you must install the USB key driver from the ABBYY FineReader Engine 10 installation CD-ROM (ABBYY FineReader Engine 10\USB Drivers\Ikeydrvr.exe in the case of a 32-bit system, or ABBYY FineReader Engine 10\USB Drivers\64\Ikeydrvr.exe in the case of a 64-bit system) on the computer on which the License Service is installed.

See also

Installing the ABBYY FineReader Engine Library in Automatic Mode
Installing the ABBYY FineReader Engine Library in Manual Mode
ABBYY FineReader Engine Distribution Kit
Licensing

Installing the License Service

For correct operation of applications using the ABBYY FineReader Engine 10 library, the License Service (LicensingService.exe) is required.

The License Service is installed automatically during the Developer and Runtime installation in automatic mode. If you use manual installation, follow the instructions below. After the installation of the License Service is complete, run the License Manager to manage licenses.

The Licensing Service settings are provided in the LicensingSettings.xml file. This file is generated automatically during automatic installation. When installing manually, you must specify the correct settings in this file. The XML scheme of the settings is located in the LicensingSettings.xsd file. You can find this file in the Bin folder of the ABBYY FineReader Engine distribution package. The detailed description of the settings is provided in the Working with the LicensingSettings.xml File section.

Installing in manual mode

⚠**Important**! Administrator access rights are necessary for the installation.

For Standalone installation:

1. Copy the files for the Licensing Service and the License Manager utility on the workstation: AbbyyZlib.dll, FineNet.dll, FineObj.dll, FObjEventSrc.dll, msvcr90.dll, Protection.dll, LicensingSchema.dll, ProductLicensingSchema.dll, LicensingService.exe, LicenseManager.exe, LicensingSettings.xml, and the Microsoft.VC90.CRT folder. Copy the resource files ProtectionRes*.dll for the languages you need and ProtectionResShared.dll. See the ABBYY FineReader Engine Distribution Kit for details.
   Note: You may not copy the LicensingSettings.xml file if you do not use hardware protection and need not specify any additional parameters.

2. Create the %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\Licenses folder. Everyone must have read and write permissions to this folder.

3. If necessary, specify parameters of the LocalLicenseServer in the LicensingSettings.xml file.

4. Run LicensingService.exe with the "/install" parameter: LicensingService.exe /install. (To uninstall the service, use the "/uninstall" parameter.)

For Network installation:

1. Copy the files for the Licensing Service and the License Manager utility on the computer which will be used as a license server: AbbyyZlib.dll, FineNet.dll, FineObj.dll, FObjEventSrc.dll, msvcr90.dll, Protection.dll, LicensingSchema.dll, ProductLicensingSchema.dll, LicensingService.exe, LicenseManager.exe, LicensingSettings.xml, and the Microsoft.VC90.CRT folder. Copy the resource files ProtectionRes*.dll for the languages you need and ProtectionResShared.dll. See the ABBYY FineReader Engine Distribution Kit for details.

2. Create the %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\Licenses folder on the server. Everyone must have read and write permissions to this folder.

3. In the LicensingSettings.xml file specify:

   - the ProtocolType attribute of the ConnectionProtocol element of the LocalLicenseServer tag;

   - the ServerAddress and ProtocolType attributes of the MainNetworkLicenseServer tag.

4. Copy the LicensingSettings.xml file with the specified settings into the Bin folder of the FineReader Engine library installation on all the workstations.

5. Run LicensingService.exe with the "/install" parameter: LicensingService.exe /install. (To uninstall the service, use the "/uninstall" parameter.)

See also

Activation
Distribution of Applications Using the ABBYY FineReader Engine Library

ABBYY FineReader Engine Distribution Kit

ABBYY FineReader Engine library is implemented as a set of dynamic link libraries (DLL) and additional modules. After you've installed the library with Developer License, its type library is registered in the system registry.

The description of the files of the library is given in the table below. The list of files supplied in different ABBYY FineReader Engine distribution kits may not be the same as in the list below and may vary depending on the product's version. All paths are given as relative to the root folder of the ABBYY FineReader Engine distribution package. The root folder is set up during ABBYY FineReader Engine installation. This table also specifies what files should be distributed as a part of your application, and what should not.

File or folder	Description	Distribution
Root folder		
Readme.htm	Readme file.	No.
\Inc[1]		
FREngine.tlb, FREngine.h, FREngine_i.c	ABBYY FineReader Engine type library description and API declaration files.	Never distribute these files. They are intended for developer purposes only. **Note:** Only for script languages, you must distribute this folder and register the FREngine.tlb file.

FineReader10-schema-v1.xsd	Scheme of an XML document.	No.
\Inc\.Net interops	The folder contains .NET wrappers for FineReader Engine type libraries. These wrappers are generated for Microsoft .NET Framework version 1.1, 2.0, and 3.5. The wrappers corresponding to each version are placed in the v1.1, v2.0, and v3.5 subfolders, respectively.	Redistribute the Interop.FREngine.dll file for suitable .NET Framework version, if you use .NET developer tools.
\Samples[1]	This folder contains sample code illustrating ABBYY FineReader Engine usage in C++ with and without Native COM support, in C#, in Visual Basic, in Visual Basic .NET, and in Delphi 5.0. See the Description of the ABBYY FineReader Engine Samples section for details.	No.
\Help		
FREngine10.chm	This manual.	No.
FREngine10UserGuide.pdf	User's guide.	No.
FREngine10AdminGuide.pdf	System administrator's guide.	No.
FREngine10_Distribution.csv	The list of files in the Bin folder saved in CSV format. Can be used to create automatically a list of files to be distributed.[2]	No.
\USB Drivers		
Ikeydrvr.exe	Hardware Key drivers installation utility for 32-bit systems.	Redistribute this file if you choose to use ABBYY FineReader Engine activation by the Hardware Key.
\64\Ikeydrvr.exe	Hardware Key drivers installation utility for 64-bit systems.	Redistribute this file if you choose to use ABBYY FineReader Engine activation by the Hardware Key.
\Bin		
AbbyyZlib.dll, Awl.dll, AwlGdi.dll, Barcode.dll, DocumentProcessing.dll, DocumentAnalysis.BarcodesFinder.dll, DocumentAnalysis.Objects.dll, DocumentAnalysis.ObjectsExtraction.dll, DocumentAnalysis.PageServices.dll, DocumentAnalysis.Segmentation.dll, Export.dll, FineNet.dll, FineObj.dll, FObjEventSrc.dll, FontSupport.dll, FREngine.dll, FREngine.dlp, FREngineProcessor.exe, FREngineProcessor.dlp, LangInfo.dll, LangInfoUnicode.dll, Morphology.dll, msvcr90.dll, NLCMorphology.dll, Recognizer.dll, RecPage.dll, RegExp.dll, Splrt.dll, Synthesis.dll, TextLayout.dll, Training.dll	ABBYY FineReader Engine system modules.	Mandatory.
RecPageHP.dll	ABBYY FineReader Engine system module.	Resource file is only necessary if you intend your application to recognize checkmarks or handprinted text.
DL90ACE.dll, DL90AdobeXMP.dll, DL90AGM.dll, DL90ARE.dll, DL90AXE8SharedExpat.dll, DL90BIB.dll, DL90BIBUtils.dll, DL90CoolType.dll, DL90JP2KLib.dll, DL90PDFL.dll, icucnv36.dll, icudt36.dll, pdfport.dll, pdfsettings.dll, Image.Format.Pdf.dll, Image.Helper.Pdf.dll	ABBYY FineReader Engine system modules for processing files in PDF format.	Resource files are only necessary if you intend your application to process PDF files. **Note:** See detailed list of files required for PDF processing at ABBYY FineReader Engine Distribution Kit: PDF article.

Image.Codec.AbbyyLossless.dll, Image.Codec.Ccitt.dll, Image.Codec.Jbig2.dll, Image.Codec.Jpeg.dll, Image.Codec.Lzw.dll, Image.Codec.Packbits.dll, Image.Codec.Zip.dll, Image.Format.Bmp.dll, Image.Format.DjVu.dll, Image.Format.Gif.dll, Image.Format.Jbig2.dll, Image.Format.Jpeg2k.dll, Image.Format.Jpeg.dll, Image.Format.Pcx.dll, Image.Format.Png.dll, Image.Format.Tiff.dll, Image.Format.Wdp.dll, Image.Format.Wic.dll[3], Image.Helper.DjVu.dll, Image.Services.Advanced.dll, Image.Services.Core.dll	ABBYY FineReader Engine system modules for processing image files.	Mandatory.
FREngine0.dll, FREngine1.dll, FREngine2.dll, FREngine3.dll, FREngine4.dll, FREngine5.dll, FREngine6.dll, FREngine7.dll, FREngine8.dll, FREngine9.dll, FREngine14.dll, FREngine15.dll, FREngine16.dll, FREngine17.dll, FREngine18.dll, FREngine20.dll, FREngine23.dll, FREngine24.dll, FREngine27.dll, FREngine63.dll, FREngine64.dll, FREngine65.dll	ABBYY FineReader Engine resource modules. Each module name has number as a postfix. The meaning of these numbers is the same as that for the Engine*.dll. The meaning of these numbers is: 0 — for English interface language, 1 — for Russian interface language, 2 — for German interface language, 3 — for French interface language, 4 — for Ukrainian interface language, 5 — for Spanish interface language, 6 — for Italian interface language, 7 — for Dutch interface language, 8 — for Danish interface language, 9 — for Swedish interface language, 14 — for Slovak interface language, 15 — for Polish interface language, 16 — for Czech interface language, 17 — for Hungarian interface language, 18 — for Lithuanian interface language, 20 — for Estonian interface language, 23 — for Bulgarian interface language, 24 — for Turkish interface language, 27 — for Portuguese (Brazil) interface language, 63 — for Korean interface language, 64 — for Chinese (PRC) interface language, 65 — for Chinese (Taiwan) interface language	Resource files are only necessary if you intend your application to display messages in a certain language. You may redistribute only resource modules corresponding to the interface language you want to use. **Note**: Corresponding MorphoRes*.dll, NlcMorphoRes*.dll, TechResources*.dll, TrainingUI*.dll, FREngineProcessor*.dll are required.
FREngineProcessor0.dll, FREngineProcessor1.dll, FREngineProcessor2.dll, FREngineProcessor3.dll, FREngineProcessor4.dll, FREngineProcessor5.dll, FREngineProcessor6.dll, FREngineProcessor7.dll, FREngineProcessor8.dll, FREngineProcessor9.dll, FREngineProcessor14.dll, FREngineProcessor15.dll, FREngineProcessor16.dll, FREngineProcessor17.dll, FREngineProcessor18.dll,	ABBYY FineReader Engine resource modules. Each module name has number as a postfix. The meaning of these numbers is the same as that for the FREngine*.dll.	Resource files are only necessary if you intend your application to display messages in a certain language. You may redistribute only resource modules corresponding to the interface language you want to use. **Note**: Corresponding FREngine*.dll, MorphoRes*.dll, NlcMorphoRes*.dll, TechResources*.dll, TrainingUI*.dll are required.

FREngineProcessor20.dll, FREngineProcessor23.dll, FREngineProcessor24.dll, FREngineProcessor27.dll, FREngineProcessor63.dll, FREngineProcessor64.dll, FREngineProcessor65.dll		
MorphoRes0.dll, MorphoRes1.dll, MorphoRes2.dll, MorphoRes3.dll, MorphoRes4.dll, MorphoRes5.dll, MorphoRes6.dll, MorphoRes7.dll, MorphoRes8.dll, MorphoRes9.dll, MorphoRes14.dll, MorphoRes15.dll, MorphoRes16.dll, MorphoRes17.dll, MorphoRes18.dll, MorphoRes20.dll, MorphoRes23.dll, MorphoRes24.dll, MorphoRes27.dll, MorphoRes63.dll, MorphoRes64.dll, MorphoRes65.dll, NlcMorphoRes0.dll, NlcMorphoRes1.dll, NlcMorphoRes2.dll, NlcMorphoRes3.dll, NlcMorphoRes4.dll, NlcMorphoRes5.dll, NlcMorphoRes6.dll, NlcMorphoRes7.dll, NlcMorphoRes8.dll, NlcMorphoRes9.dll, NlcMorphoRes14.dll, NlcMorphoRes15.dll, NlcMorphoRes16.dll, NlcMorphoRes17.dll, NlcMorphoRes18.dll, NlcMorphoRes20.dll, NlcMorphoRes23.dll, NlcMorphoRes24.dll, NlcMorphoRes27.dll, NlcMorphoRes63.dll, NlcMorphoRes64.dll, NlcMorphoRes65.dll, TechResources0.dll, TechResources1.dll, TechResources2.dll, TechResources3.dll, TechResources4.dll, TechResources5.dll, TechResources6.dll, TechResources7.dll, TechResources8.dll, TechResources9.dll, TechResources14.dll, TechResources15.dll, TechResources16.dll, TechResources17.dll, TechResources18.dll, TechResources20.dll, TechResources23.dll, TechResources24.dll, TechResources27.dll, TechResources63.dll, TechResources64.dll, TechResources65.dll, TrainingUI0.dll, TrainingUI1.dll, TrainingUI2.dll, TrainingUI3.dll, TrainingUI4.dll, TrainingUI5.dll, TrainingUI6.dll, TrainingUI7.dll, TrainingUI8.dll, TrainingUI9.dll, TrainingUI14.dll, TrainingUI15.dll, TrainingUI16.dll, TrainingUI17.dll, TrainingUI18.dll, TrainingUI20.dll, TrainingUI23.dll, TrainingUI24.dll, TrainingUI27.dll, TrainingUI63.dll, TrainingUI64.dll, TrainingUI65.dll	ABBYY FineReader Engine resource modules. Each module name has number as a postfix. The meaning of these numbers is the same as that for the FREngine*.dll.	Resource files are only necessary if you intend your application to display messages in a certain language. You may redistribute only resource modules corresponding to the interface language you want to use. **Note**: Corresponding FREngine*.dll, FREngineProcessor*.dll are required.
Bold.pat, Bold.ptc, Bold.rseg, Bold.str,	Recognition databases.	Mandatory.

Italic.pat, Italic.ptc, Italic.pts, Italic.rseg, Italic.str, Normal.pat, Normal.pdi, Normal.ptc, Normal.pts, Normal.spt, Normal.str, Normal.pseg, Normal.rseg, Part.pat, Part.ptc, Part.pts, Underlin.pat, Underlin.ptc, Underlin.rseg, Underlin.str		
Normal.arabic	Recognition databases.	Resource files are only necessary if you want your application to recognize texts in Arabic.
Normal.ccjk, Normal.cjk, Normal.ecjk, Normal.fcjk, Normal.ssc, Normal.slp, NrmlPart.ssc, NrmlPart.slp, KrnPart.slp, KrnPart.ssc, Korean.ccjk, Korean.cjk, Korean.ecjk, Korean.fcjk, Korean.slp, Korean.ssc	Recognition databases.	Resource files are only necessary if you want your application to recognize texts in Chinese, Japanese and Korean.
Default.fch, DefaultBold.fch, DefaultBoldItalic.fch, DefaultItalic.fch	Recognition databases.	Mandatory.
Printer.pat, Printer.ptc, Printer.pts, Printer.rseg, Printer.spt, Printer.str	Recognition databases.	Resource files are only necessary if you intend your application to detect text type (PossibleTextTypes property of the RecognizerParams object is used).
Typewrit.pat, Typewrit.ptc, Typewrit.pts, Typewrit.rseg, Typewrit.str	Recognition databases.	Resource files are only necessary if you intend your application to recognize text printed on a typewriter.
Checkmark.pts, Checkmark.ptv, Checkmark.spt, Checkmark.str	Recognition databases.	For recognition of checkmarks only.
E13B.pat, E13B.ptc, E13B.pts, E13B.rseg, E13B.spt, E13B.str	Recognition databases.	For recognition of MICR (Magnetic Ink Character Recognition) characters only.
CMC7.pat, CMC7.ptc, CMC7.pts, CMC7.rseg, CMC7.spt, CMC7.str	Recognition databases.	For recognition of MICR CMC-7 characters only.
OCR_A.pat, OCR_A.ptc, OCR_A.pts, OCR_A.rseg, OCR_A.spt, OCR_A.str	Recognition databases.	For recognition of OCR-A font only.
OCR_B.pat, OCR_B.ptc, OCR_B.pts, OCR_B.rseg, OCR_B.spt, OCR_B.str	Recognition databases.	For recognition of OCR-B font only.
Handprin.ptc, Handprin.pte, Handprin.pto, Handprin.pts, Handprin.ptv, Handprin.seg, Handprin.spt, Handprin.str, Erasure.spt, Erasure.str	Recognition databases for handprinted text.	Redistribute these files if you only intend to support handprint recognition in your application.
Fax.pat, Fax.ptc, Fax.pts, Fax.rseg, Fax.str	Recognition databases.	Resource files are only necessary if you intend your application to recognize texts on an image with low resolution (the IRecognizerParams::LowResolutionMode property is used).
Index.pat, Index.ptc, Index.pts, Index.rseg, Index.spt, Index.str	Recognition databases.	Resource files are only necessary if you intend your application to recognize Index text type.
Gothic.pat, Gothic.pdi, Gothic.ptc, Gothic.pts, Gothic.rseg, Gothic.spt, Gothic.str	Recognition databases.	For recognition of Gothic fonts only.
StdFonts.mtr StdFonts.psa	Files with font metrics necessary for the recognized text export in PDF format.	Redistribute these files if you only intend to support the recognized text export in PDF format by means of ABBYY FineReader Engine in your application.

LicenseManager.exe, LicensingSchema.dll, ProductLicensingSchema.dll, LicenseManager10.chm	ABBYY FineReader Engine licensing and protection modules.	Mandatory.
LicensingSettings.xml	This file includes the ABBYY FineReader Engine activation and protection settings.	Distribution of this file is mandatory, if your application works with a network license, or with a standalone license with hardware protection. **Note:** See detailed description of LicensingSettings.xml file in Working with the LicensingSettings.xml file section.
LicensingSettings.xsd	XML schema for the LicensingSettings.xml file.	No.
Protection.dll	ABBYY FineReader Engine licensing and protection module. It used for Runtime licenses only.	Mandatory.
Protection.Developer.dll	ABBYY FineReader Engine licensing and protection module. It is used for developer purpose only.	No.
ProtectionRes0.dll, ProtectionRes1.dll, ProtectionRes2.dll, ProtectionRes3.dll, ProtectionRes4.dll, ProtectionRes5.dll, ProtectionRes6.dll, ProtectionRes7.dll, ProtectionRes8.dll, ProtectionRes9.dll, ProtectionRes14.dll, ProtectionRes15.dll, ProtectionRes16.dll, ProtectionRes17.dll, ProtectionRes18.dll, ProtectionRes20.dll, ProtectionRes23.dll, ProtectionRes24.dll, ProtectionRes27.dll, ProtectionRes63.dll, ProtectionRes64.dll, ProtectionRes65.dll, ProtectionResShared.dll	ABBYY FineReader Engine licensing and protection resource modules. Each module name has number as a postfix. The meaning of these numbers is the same as that for the FREngine*.dll.	Resource files are only necessary if you intend your application to display messages in a certain language. You may redistribute only resource modules corresponding to the interface languages you want to use.
License.JasPer.txt	JasPer Software License (JPEG2000).	Mandatory.
LinksSetter.exe, SamplesConfig.exe	Auxiliary utilities for Code Samples Library and Samples configuration.	No.
FineUI.dll, FineUIRes.dll, FREngine.GUI.dll, ScanManager.dll, ScanTwain.exe, ScanWia.exe, twain.dat, wia.dat	These files are necessary for scanning.	Redistribute only if you intend to use scanning.
FineUI0.dll, FineUI1.dll, FineUI2.dll, FineUI3.dll, FineUI4.dll, FineUI5.dll, FineUI6.dll, FineUI7.dll, FineUI8.dll, FineUI9.dll, FineUI14.dll, FineUI15.dll, FineUI16.dll, FineUI17.dll, FineUI18.dll, FineUI20.dll, FineUI23.dll, FineUI24.dll, FineUI27.dll, FineUI63.dll, FineUI64.dll, FineUI65.dll, Scan0.dll, Scan1.dll, Scan2.dll, Scan3.dll, Scan4.dll, Scan5.dll, Scan6.dll, Scan7.dll, Scan8.dll, Scan9.dll, Scan14.dll, Scan15.dll, Scan16.dll, Scan17.dll, Scan18.dll, Scan20.dll, Scan23.dll, Scan24.dll, Scan27.dll, Scan63.dll, Scan64.dll, Scan65.dll	Scanning resource modules. Store scanning-specific resources in different interface languages. The codes of the interface languages are the same as those for the FREngine*.dll.	Redistribute only if you intend to use scanning and only for the interface languages you want to use.
Pictures.oce, Cjk.BigLetterVsTrash.oce, Cjk.BigPunctuationVsTrash.oce, Cjk.BigWordVsTrash.oce, Cjk.LettersForWordBuilder.oce, Cjk.ProbablyLetter.oce, Cjk.SmallLetterVsTrash.oce,	Recognition databases.	Mandatory.

Cjk.SmallPunctuationVsTrash.oce, Cjk.WordsForWordBuilder.oce, Arabic.Punctuation.oce, Arabic.Text.oce		
RecPage.v6.dll, RecPage.v6.Thunk.dll	System modules of ABBYY FineReader Engine 6.0.	Mandatory.
Asian.imageDoc, European.imagedoc	The files are used by the **IEngine::LoadModule** method.	Redistribute only if your application uses the **IEngine::LoadModule** method.
Bin\Microsoft.VC90.CRT	Microsoft C Run-Time Libraries	Mandatory.
Bin\v6	Dictionaries and recognition databases of ABBYY FineReader Engine 6.0.	Mandatory.
Bin\FontCache	Contains files with font metrics.	We recommend that you redistribute these files, but they may be excluded to save space. If the files are excluded, they will be generated during the first recognition process. First recognition, however, will slow down.
Bin\Resource		
See detailed list of files at ABBYY FineReader Engine Distribution Kit: PDF article.	ABBYY FineReader Engine system modules for processing files in PDF format.	Resource files are only necessary if you intend your application to process PDF files.
Bin\Support		
AInfo.exe, AInfo.ini	The utility, which allows you to save all necessary diagnostic information about ABBYY FineReader Engine to a ZIP file. Please provide a ZIP file which is created by the utility when contacting the technical support service.	The utility is only necessary for saving diagnostic information.
AInfo0.dll, AInfo1.dll, AInfo2.dll, AInfo3.dll, AInfo4.dll, AInfo5.dll, AInfo6.dll, AInfo7.dll, AInfo8.dll, AInfo9.dll, AInfo14.dll, AInfo15.dll, AInfo16.dll, AInfo17.dll, AInfo18.dll, AInfo20.dll, AInfo23.dll, AInfo24.dll, AInfo27.dll, AInfo63.dll, AInfo64.dll, AInfo65.dll	The resource modules for AInfo utility. Each resource module name has number as a postfix. The meaning of these numbers is the same as that for the FREngine*.dll.	Redistribute only if you intend to use AInfo utility and only for the interface languages you want to use.
Bin\ExtendedDictionaries		
Arabic.amd, Arabic.amm, Arabic.amt	Arabic language support.	For recognition of Arabic language only.
ChinesePRC.amd	Chinese (PRC) language support.	For recognition of Chinese (PRC) language only.
ChineseTaiwan.amd	Chinese (Taiwan) language support.	For recognition of Chinese (Taiwan) language only.
Japanese.amd, Japanese.amm	Japanese language support.	For recognition of Japanese language only.
Korean.amd, Korean.amm	Korean and Korean (Hangul) language support.	For recognition of Korean and Korean (Hangul) languages only.
Vietnamese.amd, Vietnamese.amm	Vietnamese language support.	For recognition of Vietnamese language only.
\Bin		
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}.amd	Additional files with key words for all languages.	For recognition of all languages.
Abkhaz.amd	Abkhaz language support.	For recognition of Abkhaz language only.

Adyghe.amd	Adyghe language support.	For recognition of Adyghe language only.
Afrikns.amd	Afrikaans language support.	For recognition of Afrikaans language only.
Agul.amd	Agul language support.	For recognition of Agul language only.
Albanian.amd	Albanian language support.	For recognition of Albanian language only.
Altaic.amd	Altaic language support.	For recognition of Altaic language only.
ArmEast.amd, ArmEast.amm, ArmEast.amt	Armenian (Eastern) language support.	For recognition of Armenian (Eastern) language only.
ArmGrab.amd, ArmGrab.amm, ArmGrab.amt	Armenian (Grabar) language support.	For recognition of Armenian (Grabar) language only.
ArmWest.amd, ArmWest.amm, ArmWest.amt	Armenian (Western) language support.	For recognition of Armenian (Western) language only.
Awar.amd	Awar language support.	For recognition of Awar language only.
Aymara.amd	Aymara language support.	For recognition of Aymara language only.
AzeriCyr.amd	Azerbaijani (Cyrillic) language support.	For recognition of Azerbaijani (Cyrillic) language only.
AzeriLat.amd	Azererbaijani (Latin) language support.	For recognition of Azererbaijani (Latin) language only.
Bashkir.amd, Bashkir.amm, Bashkir.amt	Bashkir language support.	For recognition of Bashkir language only.
Basic.amd	Basic programming language support.	For recognition of Basic programming language only.
Basque.amd	Basque language support.	For recognition of Basque language only.
Bemba.amd	Bemba language support.	For recognition of Bemba language only.
Blackft.amd	Blackfoot language support.	For recognition of Blackfoot language only.
Brazil.amd, Brazil.amm, Brazil.amt	Portuguese (Brazilian) language support.	For recognition of Portuguese (Brazilian) language only.
Breton.amd	Breton language support.	For recognition of Breton language only.
Bugotu.amd	Bugotu language support.	For recognition of Bugitu language only.
Bulgar.amd, Bulgar.amm, Bulgar.amt	Bulgarian language support.	For recognition of Bulgarian language only.
Buryat.amd	Buryat language support.	For recognition of Buryat language only.
Byelorus.amd	Belarussian language support.	For recognition of Belarussian language only.
C.amd	C/C++ programming language support.	For recognition of C/C++ programming language only.
Catalan.amd, Catalan.amm, Catalan.amt	Catalan language support.	For recognition of Catalan language only.
Chamorro.amd	Chamorro language support.	For recognition of Chamorro language only.
Chechen.amd	Chechen language support.	For recognition of Chechen language

		only.
Chemistry.amd	"Simple chemical formulas" language support.	For recognition of simple chemical formulas only.
Chukcha.amd	Chukchee language support.	For recognition of Chukchee language only.
Chuvash.amd	Chuvash language support.	For recognition of Chuvash language only.
CMC7.amd	MICR CMC-7 language support.	For recognition of MICR CMC-7 language only.
Cobol.amd	Cobol programming language support.	For recognition of Cobol programming language only.
Corsican.amd	Corsican language support.	For recognition of Corsican language only.
CrimTat.amd	Crimean Tatar language support.	For recognition of Crimean Tatar language only.
Croatian.amd, Croatian.amm, Croatian.amt	Croatian language support.	For recognition of Croatian language only.
Crow.amd	Crow language support.	For recognition of Crow language only.
Czech.amd, Czech.amm, Czech.amt	Czech language support.	For recognition of Czech language only.
Danish.amd, Danish.amm, Danish.amt	Danish language support.	For recognition of Danish language only.
Dargwa.amd	Dargwa language support.	For recognition of Dargwa language only.
Dungan.amd	Dungan language support.	For recognition of Dungan language only.
Dutch.amd, Dutch.amm, Dutch.amt	Dutch language support.	For recognition of Dutch language only.
E13B.amd	Support of the language for MICR text type.	For recognition of language for MICR text type.
English.amd, English.amm, English.amt	English language support.	For recognition of English language only.
EnglishLaw.amd, EnglishLaw.amm, EnglishLaw.amt	Legal English language support.	For recognition of English language only.
EnglishMedical.amd, EnglishMedical.amm, EnglishMedical.amt	Medical English language support.	For recognition of English language only.
EskimoC.amd	Escimo (Cyrillic) language support.	For recognition of Escimo (Cyrillic) language only.
EskimoL.amd	Escimo (Latin) language support.	For recognition of Escimo (Latin) language only.
Esperan.amd	Esperanto language support.	For recognition of Esperanto language only.
Eston.amd, Eston.amm, Eston.amt	Estonian language support.	For recognition of Estonian language only.
Even.amd	Even language support.	For recognition of Even language only.
Evenki.amd	Evenki language support.	For recognition of Evenki language only.
Faeroese.amd	Faroese language support.	For recognition of Faroese language only.
Fijian.amd	Fijian language support.	For recognition of Fijian language only.

Finnish.amd, Finnish.amm, Finnish.amt	Finnish language support.	For recognition of Finnish language only.
Flemmish.amd, Flemmish.amm, Flemmish.amt	Dutch (Belgian) language support.	For recognition of Dutch (Belgian) language only.
Fortran.amd	Fortran language support.	For recognition of Fortran language only.
French.amd, French.amm, French.amt	French language support.	For recognition of French language only.
Frisian.amd	Frisian language support.	For recognition of Frisian language only.
Friulian.amd	Friulian language support.	For recognition of Friulian language only.
GaelicSc.amd	Scottish Gaelic language support.	For recognition of Scottish Gaelic language only.
Gagauz.amd	Gagauz language support.	For recognition of Gagauz language only.
Galician.amd	Galician language support.	For recognition of Galician language only.
Ganda.amd	Ganda language support.	For recognition of Ganda language only.
German.amd, German.amm, German.amt	German language support.	For recognition of German language only.
GermanLaw.amd, GermanLaw.amm, GermanLaw.amt	Legal German language support.	For recognition of German language only.
GermanLx.amd	German (Luxembourg) language support.	For recognition of German (Luxembourg) language only.
GermanMedical.amd, GermanMedical.amm, GermanMedical.amt	Medical German language support.	For recognition of German language only.
GermanNS.amd, GermanNS.amm, GermanNS.amt	German (new spelling) language support.	For recognition of German (new spelling) language only.
GermanNSLaw.amd, GermanNSLaw.amm, GermanNSLaw.amt	Legal German (new spelling) language support.	For recognition of German (new spelling) language only.
GermanNSMedical.amd, GermanNSMedical.amm, GermanNSMedical.amt	Medical German (new spelling) language support.	For recognition of German (new spelling) language only.
Greek.amd, Greek.amm, Greek.amt	Greek language support.	For recognition of Greek language only.
Guarani.amd	Guarani language support.	For recognition of Guarani language only.
Hani.amd	Hani language support.	For recognition of Hani language only.
Hausa.amd	Hausa language support.	For recognition of Hausa language only.
Hawaiian.amd	Hawaiian language support.	For recognition of Hawaiian language only.
Hebrew.amd, Hebrew.amm, Hebrew.amt	Hebrew language support.	For recognition of Hebrew language only.
Hungar.amd, Hungar.amm, Hungar.amt	Hungarian language support.	For recognition of Hungarian language only.
Iceland.amd	Icelandic language support.	For recognition of Icelandic language only.
Ido.amd	Ido language support.	For recognition of Ido language only.

Indones.amd, Indones.amm, Indones.amt	Indonesian language support.	For recognition of Indonesian language only.
Ingush.amd	Ingush language support.	For recognition of Ingush language only.
Interlin.amd	Interlingua language support.	For recognition of Interlingua language only.
Irish.amd	Irish language support.	For recognition of Irish language only.
Italian.amd, Italian.amm, Italian.amt	Italian language support.	For recognition of Italian language only.
Java.amd	Java programming language support.	For recognition of Java programming language only.
Kabard.amd	Kabardian language support.	For recognition of Kabardian language only.
Kalmyk.amd	Kalmyk language support.	For recognition of Kalmyk language only.
Karachay.amd	Karachay-balkar language support.	For recognition of Karachay-balkar language only.
Karakalp.amd	Karakalpak language support.	For recognition of Karakalpak language only.
Kasub.amd	Kasub language support.	For recognition of Kasub language only.
Kawa.amd	Kawa language support.	For recognition of Kawa language only.
Kazakh.amd	Kazakh language support.	For recognition of Kazakh language only.
Khakas.amd	Khakass language support.	For recognition of Khakass language only.
Khanty.amd	Khanty language support.	For recognition of Khanty language only.
Kikuyu.amd	Kikuyu language support.	For recognition of Kikuyu language only.
Kirgiz.amd	Kirgiz language support.	For recognition of Kirgiz language only.
Kongo.amd	Kongo language support.	For recognition of Kongo language only.
Koryak.amd	Koryak language support.	For recognition of Koryak language only.
Kpelle.amd	Kpelle language support.	For recognition of Kpelle language only.
Kumyk.amd	Kumyk language support.	For recognition of Kumyk language only.
KurdishL.amd	Kurdish language support.	For recognition of Kurdish language only.
Lak.amd	Lak language support.	For recognition of Lak language only.
Lappish.amd	Sami (Lappish) language support.	For recognition of Sami (Lappish) language only.
Latin.amd	Latin language support.	For recognition of Latin language only.
Latvian.amd, Latvian.amm, Latvian.amt	Latvian language support.	For recognition of Latvian language only.
LatvianGothic.amd	Support of Latvian language written in Gothic script.	For recognition of Latvian language written in Gothic script only.
Lezgin.amd	Lezgi language support.	For recognition of Lezgi language only.
Lithuan.amd, Lithuan.amm, Lithuan.amt	Lithuanian language support.	For recognition of Lithuanian language

		only.
Luba.amd	Luba language support.	For recognition of Luba language only.
Macedon.amd	Macedonian language support.	For recognition of Macedonian language only.
Malagasy.amd	Malagasy language support.	For recognition of Malagasy language only.
Malay.amd	Malay (Malaysian) language support.	For recognition of Malay language only.
Malinke.amd	Malinke language support.	For recognition of Malinke language only.
Maltese.amd	Maltese language support.	For recognition of Maltese language only.
Mansi.amd	Mansi language support.	For recognition of Mansi language only.
Maori.amd	Maori language support.	For recognition of Maori language only.
Mari.amd	Mari language support.	For recognition of Mari language only.
Maya.amd	Maya language support.	For recognition of Maya language only.
Miao.amd	Miao language support.	For recognition of Miao language only.
Minankab.amd	Minangkabau language support.	For recognition of Minangkabau language only.
Mohawk.amd	Mohawk language support.	For recognition of Mohawk language only.
Moldav.amd	Moldavian language support.	For recognition of Moldavian language only.
Mongol.amd	Mongol language support.	For recognition of Mongol language only.
Mordvin.amd	Mordvin language support.	For recognition of Mordvin language only.
Nahuatl.amd	Nahuatl language support.	For recognition of Nahuatl language only.
Nenets.amd	Nenets language support.	For recognition of Nenets language only.
Nivkh.amd	Nivkh language support.	For recognition of Nivkh language only.
Nogay.amd	Nogay language support.	For recognition of Nogay language only.
NorwBok.amd, NorwBok.amm, NorwBok.amt	Norwegian (Bokmal) language support.	For recognition of Norwegian (Bokmal) language only.
NorwNyn.amd, NorwNyn.amm, NorwNyn.amt	Norwegian (Nynorsk) language support.	For recognition of Norwegian (Nynorsk) language only.
Numbers.amd	Digits language support.	For recognition of digits.
Nyanja.amd	Nyanja language support.	For recognition of Nyanja language only.
Occident.amd	Occidental language support.	For recognition of Occidental language only.
Ojibway.amd	Ojibway language support.	For recognition of Ojibway language only.
OldEnglish.amd, OldEnglish.amm, OldEnglish.amt	Old English language support.	For recognition of Old English language only.
OldFrench.amd, OldFrench.amm, OldFrench.amt	Old French language support.	For recognition of Old French language only.

OldGerman.amd, OldGerman.amm, OldGerman.amt	Old German language support.	For recognition of Old German language only.
OldItalian.amd, OldItalian.amm, OldItalian.amt	Old Italian language support.	For recognition of Old Italian language only.
OldSpanish.amd, OldSpanish.amm, OldSpanish.amt	Old Spanish language support.	For recognition of Old Spanish language only.
Ossetic.amd	Ossetian language support.	For recognition of Ossetian language only.
Papiamen.amd	Papiamento language support.	For recognition of Papiamento language only.
Pascal.amd	Pascal programming language support.	For recognition of Pascal programming language only.
Pidgin.amd	Tok Pisin language support.	For recognition of Tok Pisin language only.
Polish.amd, Polish.amm, Polish.amt	Polish language support.	For recognition of Polish language only.
Portug.amd, Portug.amm, Portug.amt	Portuguese language support.	For recognition of Portuguese language only.
Provenc.amd	Provencal language support.	For recognition of Provencal language only.
Quechua.amd	Quechua language support.	For recognition of Quechua language only.
Rhaetian.amd	Rhaeto-Romanic language support.	For recognition of Rhaeto-Romanic language only.
Roman.amd, Roman.amm, Roman.amt	Romanian language support.	For recognition of Romanian language only.
Romany.amd	Romany language support.	For recognition of Romany language only.
Ruanda.amd	Rwanda language support.	For recognition of Rwanda language only.
Rundi.amd	Rundi language support.	For recognition of Rundi language only.
RusOS.amd	Russian (Old Spelling) language support.	For recognition of Russian (Old Spelling) language only.
Russian.amd, Russian.amm, Russian.amt	Russian language support.	For recognition of Russian language only.
Samoan.amd	Samoan language support.	For recognition of Samoan language only.
Selkup.amd	Selkup language support.	For recognition of Selkup language only.
Serbian.amd	Serbian (Cyrillic) language support.	For recognition of Serbian (Cyrillic) language only.
SerbianL.amd	Serbian (Latin) language support.	For recognition of Serbian (Latin) language only.
Shona.amd	Shona language support.	For recognition of Shona language only.
Sioux.amd	Sioux language support.	For recognition of Sioux language only.
Slovak.amd, Slovak.amm, Slovak.amt	Slovak language support.	For recognition of Slovak language only.
Sloven.amd, Sloven.amm, Sloven.amt	Slovenian language support.	For recognition of Slovenian language only.
Somali.amd	Somali language support.	For recognition of Somali language only.

Sorbian.amd	Sorbian language support.	For recognition of Sorbian language only.
Sotho.amd	Sotho language support.	For recognition of Sotho language only.
Spanish.amd, Spanish.amm, Spanish.amt	Spanish language support.	For recognition of Spanish language only.
Sunda.amd	Sunda language support.	For recognition of Sunda language only.
Swahili.amd	Swahili language support.	For recognition of Swahili language only.
Swazi.amd	Swazi language support.	For recognition of Swazi language only.
Swedish.amd, Swedish.amm, Swedish.amt	Swedish language support.	For recognition of Swedish language only.
Tabassar.amd	Tabasaran language support.	For recognition of Tabasaran language only.
Tagalog.amd	Tagalog language support.	For recognition of Tagalog language only.
Tahitian.amd	Tahitian language support.	For recognition of Tahitian language only.
Tajik.amd	Tajik language support.	For recognition of Tajik language only.
Tatar.amd, Tatar.amm, Tatar.amt	Tatar language support.	For recognition of Tatar language only.
Thai.amd, Thai.amm, Thai.amt	Thai language support.	For recognition of Thai language only.
Tinpo.amd	Jingpo language support.	For recognition of Jingpo language only.
Tongan.amd	Tongan language support.	For recognition of Tongan language only.
Tswana.amd	Tswana language support.	For recognition of Tswana language only.
Tun.amd	Tun language support.	For recognition of Tun language only.
Turkish.amd, Turkish.amm, Turkish.amt	Turkish language support.	For recognition of Turkish language only.
Turkmen.amd	Turkmen language support.	For recognition of Turkmen language only.
Tuvin.amd	Tuvinian language support.	For recognition of Tuvinian language only.
Udmurt.amd	Udmurt language support.	For recognition of Udmurt language only.
UighurC.amd	Uighur (Cyrillic) language support.	For recognition of Uighur (Cyrillic) language only.
UighurL.amd	Uighur (Latin) language support.	For recognition of Uighur (Latin) language only.
Ukrain.amd, Ukrain.amm, Ukrain.amt	Ukrainian language support.	For recognition of Ukrainian language only.
Univers.amd, Univers.amm	Additional for all languages that include Latin letters.	Redistribute these files if you use at least one recognition language with Latin letters.
UzbekCyr.amd	Uzbek (Cyrillic) language support.	For recognition of Uzbek (Cyrillic) language only.
UzbekLat.amd	Uzbek (Latin) language support.	For recognition of Uzbek (Latin) language only.

Visayan.amd	Cebuano language support.	For recognition of Cebuano language only.
Welsh.amd	Welsh language support.	For recognition of Welsh language only.
Wolof.amd	Wolof language support.	For recognition of Wolof language only.
Xhosa.amd	Xhosa language support.	For recognition of Xhosa language only.
Yakut.amd	Yakut language support.	For recognition of Yakut language only.
Yiddish.amd	Yiddish language support.	For recognition of Yiddish language only.
Zapotec.amd	Zapotec language support.	For recognition of Zapotec language only.
Zulu.amd	Zulu language support.	For recognition of Zulu language only.
Extra.amd	Additional for special language units.	For recognition of all languages.

[1] — You can find this folder in:

- folder **%ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\FineReader Engine** — for Windows 2000, Windows XP, Windows Server 2003;
- folder **%ProgramData%\ABBYY\SDK\10\FineReader Engine** — for Windows Vista, Windows Server 2008, Windows 7.

[2] — The FREngine10_Distribution.csv file can be used to automatically create a list of files required for your application to function. This file provides the following data for each file of the Bin folder:

- *Path* — file path in the root installation folder,
- *FileName* — file name,
- *RequiredByModule* — module that uses this file,
- *RequiredByLanguage* — language, for working with which this file is used,
- *Responsibility* — file's area of responsibility providing additional information on file usage (for example, in the case of a file used for working with a language, it may be stated whether this file is used to display messages in this language or to recognize texts),
- *Optional* — whether the file is required for the current module or language.

[3] — The file is provided for Windows Imaging Component support. If your application use it, the COM library must be initialized before getting the **Engine** object.

See also

List of the predefined languages in ABBYY FineReader Engine
ABBYY FineReader Engine Distribution Kit: PDF

ABBYY FineReader Engine Distribution Kit: PDF

Resource files listed below are only necessary if you intend your application to process PDF files.

File or folder	Distribution
\Bin	
DL90ACE.dll, DL90AdobeXMP.dll, DL90AGM.dll, DL90ARE.dll, DL90AXE8SharedExpat.dll, DL90BIB.dll, DL90BIBUtils.dll, DL90CoolType.dll, DL90JP2KLib.dll,	Mandatory.

DL90PDFL.dll, icucnv36.dll, icudt36.dll, pdfport.dll, pdfsettings.dll, Image.Format.Pdf.dll, Image.Helper.Pdf.dll	
\Bin\Resource\Cmap	
Adobe-GB1-2 Adobe-GB1-4 Adobe-CNS1-0 Adobe-CNS1-3 Adobe-Japan1-2 Adobe-Japan1-4 Adobe-Korea1-1 UniGB-UCS2-H UniGB-UCS2-V UniCNS-UCS2-H UniCNS-UCS2-V UniJIS-UCS2-H UniJIS-UCS2-V UniKS-UCS2-H UniKS-UCS2-V Identity-H Identity-V	Mandatory.
\Bin\Resource\Cmap	
78-EUC-H 78-EUC-V 78-H 78ms-RKSJ-H 78ms-RKSJ-V 78-RKSJ-H 78-RKSJ-V 78-V 83pv-RKSJ-H 90msp-RKSJ-H 90msp-RKSJ-V 90ms-RKSJ-H 90ms-RKSJ-UCS2 90ms-RKSJ-V 90pv-RKSJ-H 90pv-RKSJ-UCS2 90pv-RKSJ-UCS2C 90pv-RKSJ-V Add-H Add-RKSJ-H Add-RKSJ-V Add-V Adobe-CNS1-1 Adobe-CNS1-2 Adobe-CNS1-4 Adobe-CNS1-5 Adobe-CNS1-B5pc Adobe-CNS1-ETen-B5 Adobe-CNS1-H-CID Adobe-CNS1-H-Host Adobe-CNS1-H-Mac Adobe-CNS1-UCS2 Adobe-GB1-0 Adobe-GB1-1 Adobe-GB1-3 Adobe-GB1-5 Adobe-GB1-GBK-EUC	We recommend that you redistribute these files, but they may be excluded to save space.

Adobe-GB1-GBpc-EUC Adobe-GB1-H-CID Adobe-GB1-H-Host Adobe-GB1-H-Mac Adobe-GB1-UCS2 Adobe-Japan1-0 Adobe-Japan1-1 Adobe-Japan1-3 Adobe-Japan1-5 Adobe-Japan1-6 Adobe-Japan1-90ms-RKSJ Adobe-Japan1-90pv-RKSJ Adobe-Japan1-H-CID Adobe-Japan1-H-Host Adobe-Japan1-H-Mac Adobe-Japan1-PS-H Adobe-Japan1-PS-V Adobe-Japan1-UCS2 Adobe-Japan2-0 Adobe-Korea1-0 Adobe-Korea1-2 Adobe-Korea1-H-CID Adobe-Korea1-H-Host Adobe-Korea1-H-Mac Adobe-Korea1-KSCms-UHC Adobe-Korea1-KSCpc-EUC Adobe-Korea1-UCS2 B5-H B5pc-H B5pc-UCS2 B5pc-UCS2C B5pc-V B5-V CNS1-H CNS1-V CNS2-H CNS2-V CNS-EUC-H CNS-EUC-V ETen-B5-H ETen-B5-UCS2 ETen-B5-V ETenms-B5-H ETenms-B5-V ETHK-B5-H ETHK-B5-V EUC-H EUC-V Ext-H Ext-RKSJ-H Ext-RKSJ-V Ext-V GB-EUC-H GB-EUC-V GB-H GBK2K-H GBK2K-V GBK-EUC-H GBK-EUC-UCS2 GBK-EUC-V GBKp-EUC-H GBKp-EUC-V GBpc-EUC-H GBpc-EUC-UCS2 GBpc-EUC-UCS2C	

GBpc-EUC-V GBT-EUC-H GBT-EUC-V GBT-H GBTpc-EUC-H GBTpc-EUC-V GBT-V GB-V H Hankaku Hiragana HKdla-B5-H HKdla-B5-V HKdlb-B5-H HKdlb-B5-V HKgccs-B5-H HKgccs-B5-V HKm314-B5-H HKm314-B5-V HKm471-B5-H HKm471-B5-V HKscs-B5-H HKscs-B5-V Hojo-EUC-H Hojo-EUC-V Hojo-H Hojo-V Katakana KSC-EUC-H KSC-EUC-V KSC-H KSC-Johab-H KSC-Johab-V KSCms-UHC-H KSCms-UHC-HW-H KSCms-UHC-HW-V KSCms-UHC-UCS2 KSCms-UHC-V KSCpc-EUC-H KSCpc-EUC-UCS2 KSCpc-EUC-UCS2C KSCpc-EUC-V KSC-V NWP-H NWP-V RKSJ-H RKSJ-V Roman UCS2-90ms-RKSJ UCS2-90pv-RKSJ UCS2-B5pc UCS2-ETen-B5 UCS2-GBK-EUC UCS2-GBpc-EUC UCS2-KSCms-UHC UCS2-KSCpc-EUC UniCNS-UTF8-H UniCNS-UTF8-V UniCNS-UTF16-H UniCNS-UTF16-V UniCNS-UTF32-H UniCNS-UTF32-V UniGB-UTF8-H UniGB-UTF8-V UniGB-UTF16-H	

UniGB-UTF16-V UniGB-UTF32-H UniGB-UTF32-V UniHojo-UCS2-H UniHojo-UCS2-V UniHojo-UTF8-H UniHojo-UTF8-V UniHojo-UTF16-H UniHojo-UTF16-V UniHojo-UTF32-H UniHojo-UTF32-V UniJIS2004-UTF8-H UniJIS2004-UTF8-V UniJIS2004-UTF16-H UniJIS2004-UTF16-V UniJIS2004-UTF32-H UniJIS2004-UTF32-V UniJISB-UCS2-H UniJISPro-UCS2-HW-V UniJISPro-UCS2-V UniJISPro-UTF8-V UniJIS-UCS2-HW-H UniJIS-UCS2-HW-V UniJIS-UTF8-H UniJIS-UTF8-V UniJIS-UTF16-H UniJIS-UTF16-V UniJIS-UTF32-H UniJIS-UTF32-V UniJISX0213-UTF32-H UniJISX0213-UTF32-V UniJISX02132004-UTF32-H UniJISX02132004-UTF32-V UniKS-UTF8-H UniKS-UTF8-V UniKS-UTF16-H UniKS-UTF16-V UniKS-UTF32-H UniKS-UTF32-V V WP-Symbol	
\Bin\Resource\Font	
AdobePiStd.otf CourierStd.otf CourierStd-Bold.otf CourierStd-BoldOblique.otf CourierStd-Oblique.otf sy______.pfb sy______.pfm zx______.mmm zx______.pfb zx______.pfm zy______.mmm zy______.pfb zy______.pfm	Mandatory.
\Bin\Resource\Font	
AdobeHeitiStd-Regular.otf AdobeMingStd-Light.otf AdobeMyungjoStd-Medium.otf AdobeSongStd-Light.otf KozGoPr6N-Medium.otf KozMinPr6N-Regular.otf	We recommend that you redistribute these files especially if you intend your application to process PDF files in Chinese, Japanese, or Korean.

\Bin\Resource\Unicode\ICU	
ctl_gb18030.cnv icudt26l.dat	Mandatory.
\Bin\Resource\Unicode\Mappings\Adobe	
HKSCS.txt Japanese83pv.txt JISX0208.txt JISX0213.txt readme.txt stdenc.txt symbol.txt zdingbat.txt	Mandatory.
\Bin\Resource\Unicode\Mappings\Mac	
ARABIC.TXT CENTEURO.TXT CHINSIMP.TXT CHINTRAD.TXT CORPCHAR.TXT CROATIAN.TXT CYRILLIC.TXT DEVANAGA.TXT DINGBATS.TXT FARSI.TXT GREEK.TXT GUJARATI.TXT GURMUKHI.TXT HEBREW.TXT ICELAND.TXT JAPANESE.TXT KOREAN.TXT README.TXT ROMAN.TXT ROMANIAN.TXT SYMBOL.TXT THAI.TXT TURKISH.TXT UKRAINE.TXT	Mandatory.
\Bin\Resource\Unicode\Mappings\Win	
CP874.TXT CP932.TXT CP936.TXT CP949.TXT CP950.TXT CP1250.TXT CP1251.TXT CP1252.TXT CP1253.TXT CP1254.TXT CP1255.TXT CP1256.TXT CP1257.TXT CP1258.TXT	Mandatory.

See also

ABBYY FineReader Engine Distribution Kit

Specifications

This section contains the descriptions of ABBYY FineReader Engine 10 general features:

- Supported Image Formats
- List of the Predefined Languages
- Text Types
- Barcode Types
- Export Formats
- What's New in ABBYY FineReader Engine 10
- Compatibility with ABBYY FineReader Engine 9.0
- Version History
- System Requirements

Supported Image Formats

The ABBYY FineReader Engine 10 opens and saves image files in the following formats:

Format	Extension	Open	Save
BMP: uncompressed black and white 4- and 8-bit — uncompressed Palette 16-bit — uncompressed, uncompressed Mask 24-bit — uncompressed 32-bit — uncompressed, uncompressed Mask	bmp	+	+
BMP: 4- and 8-bit — RLE compressed Palette	bmp	+	
DCX: black and white 2-, 4- and 8-bit palette 24-bit color	dcx	+	+
PCX: black and white 2-, 4- and 8-bit palette 24-bit color	pcx	+	+
PNG: black and white, gray, color	png	+	+
JPEG 2000: gray — Part 1 color — Part 1	jp2, jpc	+	+
JPEG: gray, color	jpg, jpeg, jfif	+	+
PDF (Version 1.7 or earlier)	pdf	+	+
TIFF: black and white — uncompressed, CCITT3, CCITT3FAX, CCITT4, Packbits, ZIP, LZW gray — uncompressed, Packbits, JPEG, ZIP, LZW 24-bit color — uncompressed, JPEG, ZIP, LZW 1-, 4-, 8-bit palette — uncompressed, Packbits, ZIP, LZW	tif, tiff	+	+

(including multipage TIFF)			
GIF: black and white — LZW-compressed 2-, 3-, 4-, 5-, 6-, 7-, 8-bit palette — LZW-compressed	gif	**+**	
DjVu: black and white, gray, color	djvu, djv	+	
JBIG2: black and white	jb2	+	+
WDP: black and white, gray, color (WIC or Microsoft .NET Framework 3.0 required)	wdp	+	

Note: The ABBYY FineReader Engine will not open images larger than 32512*32512 pixels.

See also

Image Quality Requirements

Predefined Languages in ABBYY FineReader Engine

Here is the list of internal names of the predefined languages that are supported in ABBYY FineReader Engine. Availability of this or that predefined recognition language depends on the availability of the corresponding modules among ABBYY FineReader Engine files. See the Installation section to know which recognition languages correspond to which ABBYY FineReader Engine modules. In addition, not all recognition languages are available for Handprint recognition. These languages are marked by special comment. The most of the predefined languages are simple ones. Comments are given for the group languages. Comments are also given for the languages that have full built-in dictionary support. ABBYY FineReader Engine provides its own system dictionaries for the languages that has full built-in dictionary support.

Internal name	**Recognition language**	**Full dictionary support available**	**Can be used for ICR with full dictionary support**	**Can be used for ICR**
Abkhaz	Abkhaz			
Adyghe	Adyghe			
Afrikaans	Afrikaans			+
Agul	Agul			
Albanian	Albanian			+
Altaic	Altaic			
Arabic	Arabic (Saudi Arabia)	+		
ArmenianEastern	Armenian (Eastern)	+		
ArmenianGrabar	Armenian (Grabar)	+		
ArmenianWestern	Armenian (Western)	+		
Awar	Avar			
Aymara	Aymara			+
AzeriCyrillic	Azerbaijani (Cyrillic)			
AzeriLatin	Azerbaijani (Latin)			+
Bashkir	Bashkir	+		
Basque	Basque			+
Belarusian	Belarussian			
Bemba	Bemba			+
Blackfoot	Blackfoot			+

Breton	Breton			+
Bugotu	Bugotu			+
Bulgarian	Bulgarian	+	+	
Buryat	Buryat			+
Catalan	Catalan	+		
Chamorro	Chamorro			+
Chechen	Chechen			
ChinesePRC	Chinese Simplified			
ChineseTaiwan	Chinese Traditional			
Chukcha	Chukcha			
Chuvash	Chuvash			
Corsican	Corsican			+
CrimeanTatar	Crimean Tatar			+
Croatian	Croatian	+	+	
Crow	Crow			+
Czech	Czech	+	+	
Danish	Danish	+		
Dargwa	Dargwa			
Dungan	Dungan			
Dutch	Dutch (Netherlands)	+	+	
DutchBelgian	Dutch (Belgium)	+	+	
English	English	+	+	
EskimoCyrillic	Eskimo (Cyrillic)			
EskimoLatin	Eskimo (Latin)			
Esperanto	Esperanto			
Estonian	Estonian	+	+	
Even	Even			+
Evenki	Evenki			+
Faeroese	Faeroese			
Fijian	Fijian			+
Finnish	Finnish	+	+	
French	French	+	+	
Frisian	Frisian			+
Friulian	Friulian			+
GaelicScottish	Scottish Gaelic			+
Gagauz	Gagauz			
Galician	Galician			+
Ganda	Ganda			+
German	German	+	+	
GermanNewSpelling	German (new spelling)	+	+	

GermanLuxembourg	German (Luxembourg)			+
Greek	Greek	+	+	
Guarani	Guarani			+
Hani	Hani			+
Hausa	Hausa			
Hawaiian	Hawaiian			+
Hebrew	Hebrew	+		
Hungarian	Hungarian	+	+	
Icelandic	Icelandic			
Ido	Ido			+
Indonesian	Indonesian	+	+	
Ingush	Ingush			
Interlingua	Interlingua			+
Irish	Irish			+
Italian	Italian	+	+	
Japanese	Japanese	+		
Kabardian	Kabardian			
Kalmyk	Kalmyk			
KarachayBalkar	Karachay-Balkar			+
Karakalpak	Karakalpak			
Kasub	Kasub			+
Kawa	Kawa			+
Kazakh	Kazakh			+
Khakas	Khakas			
Khanty	Khanty			
Kikuyu	Kikuyu			
Kirgiz	Kirghiz			+
Kongo	Kongo			+
Korean	Korean	+		
KoreanHangul	Korean (Hangul)	+		
Koryak	Koryak			
Kpelle	Kpelle			+
Kumyk	Kumyk			+
Kurdish	Kurdish			+
Lak	Lak			
Lappish	Sami (Lappish)			+
Latin	Latin			+
Latvian	Latvian	+	+	
LatvianGothic	Latvian language written in Gothic script			

Lezgin	Lezgin			
Lithuanian	Lithuanian	+	+	
Luba	Luba			+
Macedonian	Macedonian			
Malagasy	Malagasy			+
Malay	Malay			
Malinke	Malinke			+
Maltese	Maltese			
Mansi	Mansi			
Maori	Maori			+
Mari	Mari			
Maya	Maya			+
Miao	Miao			+
Minankabaw	Minangkabau			+
Mohawk	Mohawk			+
Mongol	Mongol			+
Mordvin	Mordvin			+
Nahuatl	Nahuatl			+
Nenets	Nenets			+
Nivkh	Nivkh			+
Nogay	Nogay			+
Norwegian	NorwegianNynorsk + NorwegianBokmal	+		
NorwegianBokmal	Norwegian (Bokmal)	+		
NorwegianNynorsk	Norwegian (Nynorsk)	+		
Nyanja	Nyanja			+
Occidental	Occidental			
Ojibway	Ojibway			+
OldEnglish	Old English	+		
OldFrench	Old French	+	+	
OldGerman	Old German	+	+	
OldItalian	Old Italian	+	+	
OldSpanish	Old Spanish	+	+	
Ossetic	Ossetian			
Papiamento	Papiamento			+
PidginEnglish	Tok Pisin			+
Polish	Polish	+	+	
PortugueseBrazilian	Portuguese (Brazil)	+		
PortugueseStandard	Portuguese (Portugal)	+		
Provencal	Provencal			

Quechua	Quechua			+
RhaetoRomanic	Rhaeto-Romanic			+
Romanian	Romanian	+	+	
RomanianMoldavia	Romanian (Moldavia)			+
Romany	Romany			+
Ruanda	Ruanda			+
Rundi	Rundi			+
RussianOldSpelling	Russian (old spelling)			
Russian	Russian	+	+	
Samoan	Samoan			+
Selkup	Selkup			+
SerbianCyrillic	Serbian (Cyrillic)			
SerbianLatin	Serbian (Latin)			+
Shona	Shona			
Sioux	Sioux (Dakota)			+
Slovak	Slovak	+	+	
Slovenian	Slovenian	+	+	
Somali	Somali			+
Sorbian	Sorbian			
Sotho	Sotho			+
Spanish	Spanish	+	+	
Sunda	Sunda			
Swahili	Swahili			+
Swazi	Swazi			+
Swedish	Swedish	+		
Tabassaran	Tabassaran			
Tagalog	Tagalog			+
Tahitian	Tahitian			+
Tajik	Tajik			
Tatar	Tatar	+		
Thai	Thai	+		
Tinpo	Jingpo			+
Tongan	Tongan			+
Tswana	Tswana			+
Tun	Tun			+
Turkish	Turkish	+	+	
Turkmen	Turkmen			
Tuvin	Tuvan			+
Udmurt	Udmurt			
UighurCyrillic	Uighur (Cyrillic)			

UighurLatin	Uighur (Latin)			+
Ukrainian	Ukrainian	+	+	
UzbekCyrillic	Uzbek (Cyrillic)			
UzbekLatin	Uzbek (Latin)			
Vietnamese	Vietnamese	+		
Visayan	Cebuano			+
Welsh	Welsh			
Wolof	Wolof			+
Xhosa	Xhosa			+
Yakut	Yakut			
Yiddish	Yiddish			
Zapotec	Zapotec			+
Zulu	Zulu			
Mixed *	Russian and English	+		
ChinesePRC+English	Chinese Simplified and English			
ChineseTaiwan+English	Chinese Traditional and English			
Japanese+English	Japanese and English	+		
Basic	Basic programming language			
C++	C/C++ programming language			
Cobol	Cobol programming language			
Fortran	Fortran programming language			
Java	Java programming language			
Pascal	Pascal programming language			
Chemistry	Simple chemical formulas			
E13B	For MICR (E-13B) text type			
CMC7	For MICR CMC-7 text type			
Digits	Numbers			+

* — The language is available only if Russian locale is selected on the user's computer.

See also

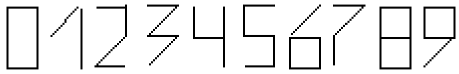
LanguageIdEnum
Working with Languages

Text Types

The ABBYY FineReader Engine 10 recognizes the following types of text:

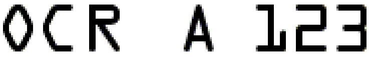
- Common typographic text
- Text typed on a typewriter

- Text printed on a dot-matrix printer
- Special set of characters including only digits written in ZIP-code style. They look as follows:

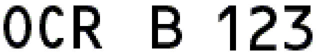


- Handprinted text. It may look as follows:

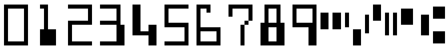
- Text in monospaced font designed specifically for Optical Character Recognition. It is largely used by banks, credit card companies and similar businesses. It may look as follows:



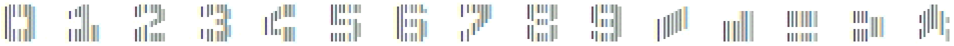
- Text printed in a font designed specifically for Optical Character Recognition. It may look as follows:



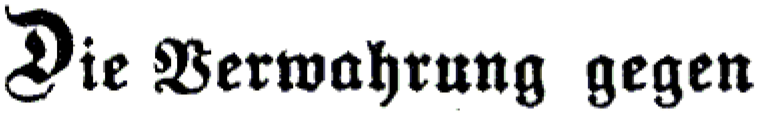
- Special numeric characters printed in magnetic ink. MICR (Magnetic Ink Character Recognition) characters are found in a variety of places, including personal checks. They may look as follows:



- Special MICR barcode font (CMC-7). It may look as follows:



- Text printed in Gothic type. It may look as follows:



For this text type, the ABBYY FineReader Engine currently supports only the "Fraktur" font.

See also

IRecognizerParams::TextTypes
TextTypeEnum

Barcode Types

ABBYY FineReader Engine 10 recognizes the following types of barcodes:

Barcode Type	Description
Aztec	Aztec is a high density two-dimensional matrix style bar code symbology that can encode up to 3750 characters from the entire 256 byte ASCII character set. The symbol is built on a square grid with a bulls-eye pattern at its center.
Codabar	Codabar is a self-checking, variable length barcode that can encode 16 data characters. It is used primarily for numeric data, but also encodes six special characters. Codabar is useful for encoding dollar and mathematical figures because a decimal point, plus sign, and minus sign can be encoded.
Code 128	Code 128 is an alphanumeric, very high-density, compact, variable length barcode scheme that can encode the full 128 ASCII character set. Each character is represented by three bars and three spaces totaling 11 modules. Each bar or space is one, two, three, or four modules wide with the total number of modules representing bars an even number and the total number of modules representing a space an odd number. Three different start characters are used to select one of three character sets.
Code 39	Code 39, also referred to as Code 3 of 9, is an alphanumeric, self-checking, variable length barcode that uses five black bars and four spaces to define a character. Three of the elements are wide and six are narrow.

Code 93	Code 93 is a variable length bar code that encodes 47 characters. It is named Code 93 because every character is constructed from nine elements arranged into three bars with their adjacent spaces. Code 93 is a compressed version of Code 39 and was designed to complement Code 39.
Data Matrix	Data Matrix is a two-dimensional matrix barcode consisting of black and white modules arranged in either a square or rectangular pattern. Every Data Matrix is composed of two solid adjacent borders in an "L" shape and two other borders consisting of alternating dark and light modules. Within these borders are rows and columns of cells encoding information. A Data Matrix barcode can store up to 2335 alphanumeric characters.
EAN 8 and 13	The European Article Numbering (EAN) system is used for products that require a country origin. This is a fixed-length barcode used to encode either eight or thirteen characters. The first two characters identify the country of origin, the next characters are data characters, and the last character is the checksum. These barcodes may include an additional barcode to the right of the main barcode. This second barcode, which is usually not as tall as the primary barcode, is used to encode additional information for newspapers, books, and other periodicals. The supplemental barcode may either encoded 2 or 5 digits of information.
IATA 2 of 5	IATA 2 of 5 is a barcode standard designed by the IATA (International Air Transport Association). This standard is used for all boarding passes.
Industrial 2 of 5	Industrial 2 of 5 is numeric-only barcode that has been in use a long time. Unlike Interleaved 2 of 5, all of the information is encoded in the bars; the spaces are fixed width and are used only to separate the bars. The code is self-checking and does not include a checksum.
Interleaved 2 of 5	Interleaved 2 of 5 is a variable length (must be a multiple of two), high-density, self-checking, numeric barcode that uses five black bars and five white bars to define a character. Two digits are encoded in every character; one in the black bars and one in the white bars. Two of the black bars and two of the white bars are wide. The other bars are narrow.
Matrix 2 of 5	Standard 2 of 5 is self-checking numeric-only barcode. Unlike Interleaved 2 of 5, all of the information is encoded in the bars; the spaces are fixed width and are used only to separate the bars. Matrix 2 of 5 is used primarily for warehouse sorting, photo finishing, and airline ticket marking.
Patch	A pattern of horizontal black bars separated by spaces. Typically, a patch code is placed near the top center of a paper document to be scanned and used as a document separator.
PDF417	PDF417 is a variable length, two-dimensional (2D), stacked symbology that can store up to 1,850 printable ASCII characters or 1,100 binary characters per symbol. PDF417 is designed with selectable levels of error correction. Its high data capacity can be helpful in applications where a large amount of data must travel with a labeled document or item.
PostNet	The Postnet (Postal Numeric Encoding Technique) is a fixed length symbology (5, 6, 9, or 11 characters) which uses constant bar and space width. Information is encoded by varying the bar height between the two values. Postnet barcodes are placed on the lower right of envelopes or postcards, and are used to expedite the processing of mail with automatic equipment and provide reduced postage rates.
QR Code	QR Code is a two-dimensional matrix barcode. The barcode has 3 large squares (registration marks) in the corners which define the top of the barcode. The black and white squares in the area between the registration marks are the encoded data and error correction keys. QR Codes can encode over 4000 ASCII characters.
UCC-128	This type of barcode is a 19 digit barcode with a 20th check digit. For a total of 20 digits. It typically is used for carton identification. Both for internal carton numbering and also for using the UCC-128 barcode on your cartons being shipped out to your customers.
UPC-A	The UPC-A (Universal Product Code) barcode is 12 digits long, including its checksum. Each digit is represented by a seven-bit sequence, encoded by a series of alternating bars and spaces. UPC-A is used for marking products which are sold at retail in the USA.
UPC-E	The UPC-E barcode is a shortened version of UPC-A barcode. It compresses the data characters and the checksum into six characters. This bar code is ideal for small packages because it is the smallest bar code.

See also

BarcodeTypeEnum
BarcodeParams

Export Formats

The ABBYY FineReader Engine allows export recognized text in the following formats:

- RTF/DOC/DOCX
- XLS/XLSX
- PDF
- PDF/A
- HTML
- PPTX
- TXT/CSV
- XML*

* – XML file format contains recognized text which structure is described with the help of XML.

See also

FileExportFormatEnum

What's New in ABBYY FineReader Engine 10

Here you can find the list of new features in ABBYY FineReader Engine 10.

New and improved language recognition

- Arabic language recognition
- Improved CJK languages recognition:
  - Chinese Simplified (PRC)
  - Chinese Traditional (Taiwan)
  - Japanese
  - Korean
- Improved Thai, Vietnamese, and Hebrew recognition
- Improved recognition of Old European languages (Fraktur font)

See the list of predefined languages in ABBYY FineReader Engine.

Speed improvements

- Normal recognition mode has become faster
- Multi-core support improvements (the MultiProcessingParams object)

FineReader Engine usage scenarios and profiles

- Document conversion scenarios:
  - Document archiving
  - Book archiving
  - Document conversion for content reuse
- Scenarios for data capture:
  - Text extraction

  - Field-level recognition
  - Barcode recognition

- For each usage scenario, the best settings are provided with the predefined profiles

Recognition improvements

- Improved OCR of low resolution documents (the IRecognizerParams::LowResolutionMode property)
- ICR improvements for European languages (English, French, German)
- Improved barcode recognition

PDF conversion improvements

- PDF (PDF/A) export may be adjusted much easier by setting only a few parameters (the new PDFExportParams object)
- Improved PDF MRC export

Image preprocessing improvements

- Improved image binarization (more text can be found on low-contrast images and images with complicated backgrounds)
- Image color filtering (IImageDocument::RemoveColorObjects)
- Improved image preprocessing for images received from a digital camera:
  - Automatic correction of 3D perspective distortions (IFRPage::RemoveGeometricalDistortions, IPageProcessingParams::RemoveGeometricalDistortions, IDocumentAnalyzer::RemoveGeometricalDistortions)
  - Blur correction (IImageDocument::RemoveCameraBlur)
  - ISO noise reduction (IImageDocument::RemoveCameraNoise)

Adaptive Document Recognition Technology (ADRT) improvements

- Processing picture captions
- Constructing a document map and table of contents
- New API for the results of document structure synthesis

New messages languages

- Brazilian
- Korean
- Chinese (RPC)
- Chinese (Taiwan)
- Danish

See the list of supported interface languages.

Please visit our website at www.abbyy.com for the most up-to-date information about ABBYY FineReader Engine and other ABBYY products.

See also

ABBYY FineReader Engine 10 and 9.0 compatibility

ABBYY FineReader Engine 10 and 9.0 compatibility

ABBYY FineReader Engine 10 is not binary compatible with ABBYY FineReader Engine 9.0. Applications that were compiled using ABBYY FineReader Engine 9.0 should be recompiled using ABBYY FineReader Engine 10 headers and library. Some changes of the source code may be necessary because of the ABBYY FineReader Engine API improvements.

Below is the full list of changes.

Layout and blocks

Object/Enumeration	Property/Method/Constant	What has happened?	Comment
Layout	**BlackSeparators**	The type of the property has been changed.	In this version separators are marked as blocks during page processing. Therefore, the type of the property is **LayoutBlocks**.
	LoadBlocks	Removed	This functionality is no longer supported.
	TextAsString		Text of barcode blocks is not included into the output text line.
	Resolution	Removed	Layout resolution is equal to the resolution of the black-and-white plane of the image for which the **Layout** object is defined. To view the resolution of the image, you can use the **XResolution** and **YResolution** properties of the corresponding **Image** object.
Block	**BarcodeBlockProperties, CheckmarkBlockProperties, CheckmarkGroupProperties, PictureBlockProperties, TableBlockProperties, TextBlockProperties**	Removed	In this version all the block type interfaces are derived from the **IBlock** interface and inherit all its properties. The following methods of the **Block** object provide access to extended attributes of blocks of these types: **GetAsBarcodeBlock**, **GetAsCheckmarkBlock**, **GetAsCheckmarkGroup**, **GetAsRasterPictureBlock**, **GetAsTableBlock**, **GetAsTextBlock**.
	RecognitionStatus	Removed	This functionality is no longer supported.
BlocksCollection		Removed	The same functionality is provided via the **LayoutBlocks** object with the following exceptions: • The **Add** and **Insert** methods for the collection received using the **ILayout::Blocks** property cannot be called. To add or insert a block into the collection, use the **AddBlock** or **InsertBlock** methods of the corresponding **Layout** object.
AutoAnalysisBlockProperties		Removed	This type of block is no longer supported. To analyze an image zone, you can use the **AnalyzeRegion** method of the **FRPage** or **DocumentAnalyzer** object.
BarcodeBlockProperties		Removed	The same functionality is provided via the **BarcodeBlock** object with the following exceptions: • **BarcodeOrientation** — the barcode orientation is defined

			with the help of the **ImageProcessingParams** property. • **Text** — the property returns a string instead of the **Text** object. The text is also accessible via the **BarcodeText** property. • **BarcodeSupplementType** — the property has been renamed. The new name of the property is **SupplementType**.
CheckmarkBlockProperties		Removed	Use the **CheckmarkBlock** object instead.
CheckmarkGroupProperties		Removed	The same functionality is provided via the **CheckmarkGroup** object with the following exceptions: • **MaximumCheckedInGroup** — the default value of this property has been changed. • **Add** — the method has been removed. Use the **AddCheckmark** method instead. • **Insert** — the method has been removed. Use the **InsertCheckmark** method instead.
PictureBlockProperties		Removed	The same functionality is provided via the **RasterPictureBlock** and **VectorPictureBlock** objects with the following exceptions: • **DescriptionText** — use the **IBlock::Description** property instead. • **ImageEnhancerValues** — the properties of this subobject are provided via the **RasterPictureBlock** object. • **IsEmbeddedInText** — use the **ITextPicture::IsInlinePicture** property instead.
TableBlockProperties		Removed	Use the **TableBlock** object instead.
TextBlockProperties		Removed	The same functionality is provided via the **TextBlock** object with the following exceptions: • **Text** — the property is read-only.
BlockTypeEnum	BT_Picture	Removed	Two types of picture blocks are supported in this version: BT_RasterPicture and BT_VectorPicture.
	BT_AutoAnalysis	Removed	This type of block is no longer supported.
BlackSeparator		Removed	The same functionality is provided via the

			SeparatorBlock object with the following exceptions: • **Direction** — the property is no longer supported as separators can be slanting. • **Type** — use the **SeparatorType** property instead. • **Left**, **Top**, **Right**, **Bottom** — the coordinates of the start and end points of the separator is provided instead.
BlackSeparators		Removed	The same functionality is provided via the **SeparatorGroup** object.
BlackSeparatorDirectionEnum		Removed	These constants are no longer in use.
BlackSeparatorTypeEnum		Removed	These constants are no longer in use.
TableCell	**Text, ImagePreprocessingParams, RecognizerParams, ContainsPicture**	Removed	Use the **ITableCell::Block** property instead. Properties and methods of the **Block** sub-object of the **TableCell** object provides similar functionality.
TableSeparatorTypeEnum		The names and number of enumeration constants have been changed.	Use TST_Invisible instead of TST_White and TST_Explicit instead of TST_Black.
TableSeparator	**Position,** **Type**	The properties are no longer read-only.	

Processing parameters

Object/Enumeration	**Property/Method/Constant**	**What has happened?**	**Comment**
PageProcessingParams	**TableAnalysisParams**	Removed	The corresponding properties are available through the **IPageAnalysisParams::TableAnalysisParams** subobject.
	PageSynthesisParams	Removed	The **PageSynthesisParams** object is no longer supported. The same functionality is provided via the **SynthesisParamsForPage** and **SynthesisParamsForDocument** objects.
PageAnalysisParams	**DetectOrientation**	Removed	Use the DetectOrientation and OrientationDetectionParams properties of the PageProcessingParams object.
	DetectBarcodes, DetectInvertedImage,	Removed	Use the corresponding properties of the **PageProcessingParams** object.
	DetectInvertedTexture	Removed	Use the **RemoveTexture** property of the **ObjectsExtractionParams**

			object instead.
	DetectMatrixPrinter, DetectPorousText, FastObjectsExtraction, FlexiFormsDA, FullTextIndexDA, ProhibitColorImage, RemoveTexture	Removed	Use the corresponding properties of the **ObjectsExtractionParams** object instead.
	ProhibitClockwiseRotation, ProhibitCounterclockwiseRotation, ProhibitUpsidedownRotation	Removed	The corresponding properties are available through the **OrientationDetectionParams** subobject of the **PageAnalysisParams** object.
PageSynthesisParams		Removed	The same functionality is provided via the **SynthesisParamsForPage** and **SynthesisParamsForDocument** objects.
SynthesisParamsForDocument	**DetectCaptions, DetectColumns, DetectFootnotes, DetectRunningTitles**	Removed	The corresponding properties are available through the **DocumentStructureDetectionParams** subobject of the **SynthesisParamsForDocument** object.
	DetectBold, DetectDropCaps, DetectFontSize, DetectItalic, DetectSerifs, DetectSmallCaps, DetectSubscriptsSuperscripts, DetectUnderlineStrikeout, MonospaceDetectionMode	Removed	The corresponding properties are available through the **FontFormattingDetectionParams** subobject of the **SynthesisParamsForDocument** object.
	DontReplaceBullets, UseVisualOrderForBidirectionalText	Removed	These properties are no longer supported.
	DetectScaleSpacing	Removed	Use the DetectScaling and DetectSpacing properties of the SynthesisParamsForDocument object instead.
SynthesisParamsForPage	**DoNotExtractSeparators**	Removed	This property is no longer supported.
BarcodeAnalysisParams		Removed	The same functionality is provided via the **BarcodeParams** and **ObjectsExtractionParams** objects.
BarcodeParams	**IsEAN13InterpretedAsUPCA**	Removed	The UPC-A barcode type can be specified explicitly: use the BT_UPCA enumeration constant in the value of the **Type** property of the **BarcodeParams** object.
TableAnalysisParams	**RectangularTables**	Removed	This property is no longer supported.
ImageProcessingParams	**BlackGarbageSize** **WhiteGarbageSize**	Removed	These properties are no longer supported.
	ProhibitCorrectLocalSkew	Removed	Use the **SkewCorrectionMode** property of the **TextBlockAnalysisParams**

			object instead.
	RemoveGarbage, **RemoveTexture**	Removed	Use the corresponding properties of the **ObjectsExtractionParams** object instead.
	AutodetectInversion	Removed	Use the **AutodetectInversion** property of the **TextBlockAnalysisParams** object instead.

Text-related objects

Object/Enumeration	**Property/Method/Constant**	**What has happened?**	**Comment**
Text	**BackgroundColor**	Removed	Currently the background color is an attribute of a paragraph or a block. Use the corresponding properties of the **ParagraphParams** and **Block** objects.
	IsMirrored	Removed	Use the **IsVerticalMirrored** property of the **TextOrientation** subobject of the **Text** object.
	IsPlain, **Append,** **CopyFrom**	Removed	The functionality is no longer supported.
	DiscardExtendedRecAttributes	Removed	The **ExtendedRecAttributes** object is no longer in use.
	DiscardRectangles	Removed	Use the properties and methods of the **CharParams** object to edit characters rectangles.
	TextOrientation	The type of the property has been changed.	This property provides access to the **TextOrientation** object.
	Insert, **InsertParagraphBreak**	Removed	Use the **InsertText** and **InsertParagraphBreak** methods of the **Paragraph** object respectively.
	SeparateHorz, **SeparateVert**	Removed	The functionality is no longer supported.
TextOrientationEnum		Removed	These constants are no longer in use.
Paragraph	**Bookmark**	Input parameter has been changed.	The property receives as an input parameter the index of the bookmark in the internal collection of the paragraph's bookmarks instead of its position inside the paragraph.
	GetBookmarkRange	Input parameter has been changed.	The method receives as an input parameter the name of the bookmark instead of its position inside the paragraph.
	Params	Renamed	The new name of the property is **ExtendedParams**.
	TabLeaderInfo	Removed	Use the **TabPositions** property instead.
	ParentText	Removed	These properties are no longer supported.
	Id	Removed	Use the **IPageElement::Id** instead.
	SetCharParams	The number of input	The method takes an OR combination of the **StyleParamsEnum** constants as

		parameters has been changed.	one of the input parameters.
	Left **Top** **Right** **Bottom**		The coordinates of the paragraph borders are not available for the paragraphs of barcodes.
	SetRect	Removed	This functionality is no longer supported.
	ImageEnhancerValues	Removed	Image enhancement is no longer supported. To access the properties of an inline picture: 1. Use the **IParagraph:: InlinePictureID** property to receive the ID of the **PageElement** object which describes the embedded image. 2. Find the corresponding **PageElement** object by its ID. 3. Receive its **TextPicture** object using the **GetAsPicture** method and work with its properties.
ParagraphParams	**ChangeParagraphTabInfo, GetParagraphTabInfoCopy**	Removed	Use the TabPositions, TabPosition objects and the IParagrapgh::TabPositions property instead.
	HasUncertainAlignment, Width	Removed	These properties are obsolete.
	UserProperty	Removed	This functionality is no longer supported.
ParagraphLine	**BaseLine**		The property becomes read-only.
	Left **Top** **Right** **Bottom**		The coordinates of the line borders are not available for the paragraphs of barcodes.
	SetRect	Removed	This functionality is no longer supported.
Paragraphs	**Find**	Renamed	Use the **GetIndex** method instead.
ParagraphTabInfo		Removed	The same functionality is provided via the **TabPosition** object.
CharParams	**IsHidden**	Removed	This property is no longer supported.
	IsStartStopSymbol	Removed	Use the **IsStartStopSymbol** property of the **BarcodeSymbol** object instead.
	ExtendedRecAttributes	Removed	This object is no longer in use. Similar properties are available via the **CharacterRecîgnitionVariant** and **WordRecognitionVariant** object.
	CharacterHeight, HasUncertainHeight	Removed	These properties are no longer supported.
ExtendedRecAttributes		Removed	Use the CharacterRecîgnitionVariant,

			WordRecognitionVariant, and CharParams objects instead: • **CharConfidence**, **SerifProbability** — use the corresponding properties of the **CharacterRecognitionVariant** object. • **IsWordFromDictionary**, **MeanStrokeWidth** — use the corresponding properties of the **WordRecognitionVariant** object. • **IsWordIdentifier**, **IsWordNormal**, **IsWordNumeric** — use the **ModelType** property of the **WordRecognitionVariant** object: o IsWordNormal set to TRUE is equal to the ModelType property set to WMT_MonoLingualWord | WMT_RegExpWord o IsWordNumeric set to TRUE — to WMT_Number | WMT_NumberWithQualifier | WMT_RomanNumber | WMT_PhoneNumber | WMT_UrlOrEmail o IsWordNormal set to TRUE — to WMT_BilingualComposit | WMT_Acronym | NumberWithQualifier | WMT_WordNumberComposite | WMT_BilingualWordNumberComposite | WMT_RomanNumber | WMT_MixedFormDictionaryWord | WMT_PhoneNumber | WMT_Punctuation | WMT_FileNumber | WMT_UrlOrEmail

			• **WordPenalty** — use the **WordConfidence** property of the **WordRecognitionVariant** object. • **IsWordStart** — use the corresponding property of the **CharParams** object.
CFL_ prefixed flags		The corresponding module has been renamed.	The new name of the module is CharacterFlags.
	CFL_Bold, CFL_Italic, CFL_Underlined, CFL_Strikeout, CFL_SmallCaps CFL_FontSize, CFL_FontName, CFL_Scale, CFL_Spacing, CFL_Color, CFL_BaseLine	Removed	Use the corresponding constants of the **StyleParamsEnum** enumeration.
	CFL_Hidden, CFL_UncertainCharHeight, CFL_CharacterHeight, CFL_ExtRecAttributes, CFL_Rectangle, CFL_IsStartStopSymbol	Removed	These constants are no longer in use.
PlainText	**SaveToTextFile**	The number of input parameters has been changed.	The following input parameters have been added: encoding type and code page of the output file.

Language-related objects

Object/Enumeration	**Property/Method/Constant**	**What has happened?**	**Comment**
StandardDictionaryDescription	**CanUseTrigramms**	Renamed	The new name of the property is **CanUseTrigrams**.
EnumDictionaryWords	**Next**		The confidence of the word is an output parameter and is no longer an input parameter.

Export

Object/Enumeration	**Property/Method/Constant**	**What has happened?**	**Comment**
DBFExportParams		Removed	Export to DBF format is no longer supported.
HTMLExportParams	**Author,** **Keywords,** **Subject,** **Title**	Removed	Use the corresponding properties of the **DocumentContentInfo** subobject of the **FRDocument** or **DocumentInfo** object. In order these properties are written into HTML file, set the values of the **WriteAuthor**, **WriteKeywords**, **WriteSubject**, **WriteTitle** properties to TRUE.
	CodePageType	Removed	This property is no longer supported.
	Quality	Renamed	The new name of the property is

			PictureJpegQuality.
	SeparatePages	Removed	This property is no longer supported. Output HTML document can be split into files (use the **SplitDocumentToFiles** property).
	UseUnicode	Removed	This property is no longer supported. Similar functionality is provided via the **EncodingType** property.
HTMLSynthesisModeEnum	HSM_PageLayout	Removed	Page structure is no longer retained in the output HTML document. Logical structure of the document can be saved using the HSM_FlexibleLayout constant.
HTMLFormatModeEnum	HFM_TwoFormats32_40	Removed	This format is no longer supported.
TextExportParams	**CodePageType**	Removed	This property is no longer supported.
	TXTIsCSV **CSVTablesOnly**	Removed	To export to CSV format, use the **ExportFormat** property: • select the TEF_CSVFullLayout constant as the value of this property to retain full layout in the output CSV file • set the property to TEF_CSVTablesOnly to export recognized text from table blocks only
RTFExportParams	**Author,** **Keywords,** **Subject,** **Title**	Removed	Use the corresponding properties of the **DocumentContentInfo** subobject of the **FRDocument** or **DocumentInfo** object. In order these properties are written into RTF/DOC/DOCX file, set the values of the **WriteAuthor**, **WriteKeywords**, **WriteSubject**, **WriteTitle** properties to TRUE.
	FormatWord95	Removed	This property is no longer supported.
	PictureFormat		The EPF_BmpColor, EPF_BmpGray, EPF_BmpBlackWhite constants cannot be used as the values of this property as the Word95 format is no longer supported.
	Quality	Renamed	The new name of the property is **PictureJpegQuality**.
	EnhanceImages	Removed	Image enhancement is no longer supported.
	WriteWordXML, **WriteCustomXMLTags**	Removed	Export to WordXML format is no longer supported.
PDFAExportParams		Renamed	The new name of the object is **PDFAExportParamsOld**. This object is obsolete. We recommend you to use the new **PDFExportParams** object instead. The **PDFAExportParamsOld** object provides the same functionality with the following exceptions: • **Author**, **Creator**, **Keywords**, **Producer**, **Subject**, **Title** — the properties have been removed. Use the corresponding properties of the **DocumentContentInfo** subobject of the **FRDocument** or **DocumentInfo** object. In order these properties are written into PDF/A file, set the values of the **WriteAuthor**, **WriteKeywords**, **WriteSubject**, **WriteTitle** properties to

			TRUE. • **PictureResolution** — the default value has been changed. The new default value is 150 dpi. • **ExportMode** — the default value has been changed. The new default value is PEM_ImageOnText dpi.
PDFExportParams		Renamed	The new name of the object is **PDFExportParamsOld**. This object is obsolete. We recommend you to use the new **PDFExportParams** object instead. The **PDFExportParamsOld** object provides the same functionality with the following exceptions: • **Author**, **Creator**, **Keywords**, **Producer**, **Subject**, **Title** — the properties have been removed. Use the corresponding properties of the **DocumentContentInfo** subobject of the **FRDocument** or **DocumentInfo** object. In order these properties are written into PDF file, set the values of the **WriteAuthor**, **WriteKeywords**, **WriteSubject**, **WriteTitle** properties to TRUE. • **ReplaceUncertainWordsWithImage** — the default value has been changed. The new default value is FALSE. • **PictureResolution** — the default value has been changed. The new default value is 150 dpi.
PDFMRCParams	**MRCEnabled**	Removed	Use the **EnableMRC** property of the **PDFExportParams** or **PDFAExportParams** object instead.
	BackgroundDownSampling, **BackgroundFormat,** **BackgroundQuality,** **ColorMaskDownSampling,** **MonochromeText,** **TextMaskQuality**		The default values have been changed.
PDFExportModeEnum	PEM_TextOnly	Renamed	The new name of the constant is PEM_TextWithPictures.
XLExportParams	**Author**, **Keywords**, **Subject**, **Title**	Removed	Use the corresponding properties of the **DocumentContentInfo** subobject of the **FRDocument** or **DocumentInfo** object. In order these properties are written into XLS/XLSX file, set the values of the **WriteAuthor**, **WriteKeywords**, **WriteSubject**, **WriteTitle** properties to TRUE.
PPTExportParams			Export to PPT format is no longer supported. These parameters are used for export to PPTX.
	Author, **Keywords**, **Subject**, **Title**	Removed	Use the corresponding properties of the **DocumentContentInfo** subobject of the **FRDocument** or **DocumentInfo** object. In order these properties are written into PPTX file, set the

			values of the **WriteAuthor**, **WriteKeywords**, **WriteSubject**, **WriteTitle** properties to TRUE.
	Quality	Renamed	The new name of the property is **PictureJpegQuality**.
	WrapTextInBlock		The value of this property is no longer ignored when exporting to PPTX format.
FileExportFormatEnum	FEF_PPT	Removed	Export to PPT format is no longer supported. For export to PPTX use the FEF_PPTX constant.
CodePageTypeEnum		Removed	These constants are no longer in use.
IRecognizedPages	**Count**	Removed	Use the **PageIds** property instead.
	ImageDocument, **Layout**, **ReleasePage**	Input parameter has been changed.	These properties and method receive as an input parameter the page ID instead of a page number.
Exporter	**ExportPages**, **ExportPagesEx**	The number of input and output parameters has been changed.	The parameter, which defines a mode of export, has been removed, therefore the exported file cannot be put into the clipboard. These methods have two output parameters, which provide the full paths to the additional files and the additional directories that were generated during export.

Image-related objects

Image document internal format has been changed. It is a folder with files.

Object/Enumeration	Property/Method/Constant	What has happened?	Comment
ImageDocument		Internal representation has been changed.	An open image, so-called "image in internal format", is represented by a folder with files. Therefore, all the methods, which work with images in internal format (e.g. **IEngine::OpenImage**), work with folders.
	SaveImage	Removed	Use the **SaveTo** method instead.
	SaveImageDocToMemory	Renamed	Use the **SaveToMemory** method instead.
	WriteRectImage	Renamed	The new name of the method is **SaveImageRegionTo**. The method saves the parts of an image into a folder on disk. The saved image is in the ABBYY FineReader Engine internal format.
	SaveModified		The method does not overwrite the source image file. It saves the current image document state on disk.
	IsSkewCorrect	Renamed	The new name of the property is **IsSkewCorrected**.
	IsReadOnly	Removed	This property is obsolete. All modification methods work correctly with all image documents.
	IsInternalFormat	Removed	This property is obsolete. Any image document represents an image in internal format.

PrepareImageMode	**CalcSkewByBlackSquares**, **CorrectSkew**	Removed	These properties are obsolete. Use the **CorrectSkewMode** property instead.
	RemoveGarbage, **SmoothColorImage**	Removed	Similar functionality is provided via the corresponding methods of the **ImageDocument** object.
	ColorJpegQuality **GrayJpegQuality**	Removed	These properties are obsolete.
ImageCompressionEnum		The enumeration constants have been changed.	In this version ZIP compression is used.
ImageModification			This object's methods work with regions instead of rectangles. All the methods were renamed.
EnhancedImage, **ImageEnhancerValues**		Removed	Image enhancement is no longer supported.
StraightenLinesParams		Removed	This object is no longer in use. Use the **RemoveGeometricalDistortions** method of the **DocumentAnalyzer** or **FRPage** object, the **RemoveGeometricalDistortions** property of the **PageProcessingParams** object instead.

Document-related objects

Object/Enumeration	**Property/Method/Constant**	**What has happened?**	**Comment**
FRDocument	**Recognize**, **RecognizePages**	The number of input parameters has been changed.	The **ObjectsExtractionParams** object is new in the set of input parameters.
	AutoFlush	Removed	Use the IFRDocument::PageFlushingPolicy property instead.
FRPage	**AnalyzeAndRecognizeBlocks**, **AnalyzeBlock**	Removed	These methods were generally used to perform layout analysis inside blocks of the autoanalysis type. As this type of block is no longer supported the methods have been removed. You can use the **AnalyzeRegion** method, if layout analysis must be performed in an image zone, and then call the **RecognizeBlocks** method.
	Recognize **RecognizeBlocks**	The number of input parameters has been changed.	The **ObjectsExtractionParams** object is new in the set of input parameters.
DocumentContentInfo	**DocumentInformationDictionary**		The property does not return a copy of the object any more, it returns a constant object. To change the document information dictionary, you must first receive an intermediate **DocumentInformationDictionary** object with the help of the

			IEngine::CreateDocumentInformationDictionary method, change the necessary parameters, and then assign this object to the property.
	Creator	The default value has been changed.	The new default value is "ABBYY FineReader Engine 10".
DocumentInfo			This object is created using the special **IEngine::CreateDocumentInfo** method. The **IEngine::PrepareImage** and **IEngine::PrepareAndOpenImage** methods do not create this object, but take a reference to this object as an input parameter.
PageSplitDirectionEnum	PSD_NoneSplit	Renamed	The new name of the constant is PSD_NoSplit.

Engine object

The following methods and properties of the **Engine** object have been changed:

Property/Method	**What has happened?**	**Comment**
MaxMemoryImageByteSize	Removed	This property is obsolete.
MessagesLanguage		The ML_Portuguese and ML_Latvian constants have been removed from the **MessagesLanguageEnum** enumeration. These messages languages are no longer supported.
MultiProcessingMode, **RecognitionProcessesCount**	Removed	Use the corresponding properties of the **MultiProcessingParams** subobject of the **Engine** object.
CreateBarcodeAnalysisParams, **CreateBlocksCollection**, **CreateParagraphTabInfo**, **CreateStraightenLinesParams**	Removed	Corresponding objects have been removed or have no effect on the operation of ABBYY FineReader Engine.
CreateBlock	Removed	Use the **AddBlock** or **InsertBlock** method of the **Layout** object, to create a new block and add or insert it into the desired layout.
CreateText	Removed	Currently the **Text** object cannot be created.
CreateLicense	Removed	The collection of available (activated) licenses you can receive using the **Licenses** property of the **Engine** object.
CreateDocumentInformationDictionaryItem	Removed	Use the IDocumentInformationDictionary::CreateDocumentInformationDictionaryItem method instead.
OpenImage		Image in internal format is represented by a folder with files. Therefore, this method takes as an input parameter a path to a folder.
	The number of input parameters has been changed.	You do not need to pass the **DocumentInfo** object as an input parameter.
OpenBitmapImage, **OpenDib**, **OpenMemoryImage**		The resulting image document is not read-only. All modification methods work correctly with it.
PrepareDib, **PrepareBitmap**, **PrepareMemoryImage**		Image in internal format is represented by a folder with files. Therefore, these methods take as an input parameter a path to a folder for prepared images.
PrepareImage		Image in internal format is represented by a folder with files.

		Therefore, this method returns as the output parameter a set of paths to folders.
		The **DocumentInfo** object is an input parameter and is no longer an output parameter. The **DocumentInfo** object can be created using the **CreateDocumentInfo** method of the **Engine** object.
PrepareAndOpenImage		The **DocumentInfo** object is an input parameter and is no longer an output parameter. The **DocumentInfo** object can be created using the **CreateDocumentInfo** method of the **Engine** object.
AnalyzeAndRecognizeBlocks, **AnalyzeRegion**, **AnalyzeTable**, **ExtractBarcodes**, **FindPageSplitPosition**, **RecognizeBlocks**	Removed	Use the corresponding methods of the **DocumentAnalyzer** or **FRPage** object instead.
AnalyzeAndRecognizePage, **RecognizeImageDocumentAsPlainText**, **RecognizeImageAsPlainText**	The number of input parameters has been changed.	The **SynthesisParamsForPage** object is new in the set of input parameters.
RecognizePage	The input parameters have been changed.	The **SynthesisParamsForPage** and **ObjectsExtractionParams** objects are new in the set of input parameters.
RecognizePages	The input parameters have been changed.	The **SynthesisParamsForPage** and **ObjectsExtractionParams** objects are new in the set of input parameters. The **PageProcessingParams** object is no longer used as the input parameter.
RecognizeImageFile	The number of input parameters has been changed.	The **SynthesisParamsForPage** and **SynthesisParamsForDocument** objects are new in the set of input parameters.
ExportPage	The number of input parameters has been changed.	The **DocumentInfo** object is new in the set of input parameters.
CleanDocumentAnalyzer	Removed	This method is obsolete. To release recognition session, use the **IFRPage::CleanRecognizerSession** method. To release the whole document, simply release all the references to the **FRDocument** and **DocumentInfo** objects.
StraightenLines	Removed	Use the RemoveGeometricalDistortions method of the DocumentAnalyzer, FRPage, or PageAnalysisParams object instead.
PerformEnhancement, **EnhanceImageBlocks**	Removed	Image enhancement is no longer supported.

__Document Analyzer__

Object/Enumeration	**Property/Method/Constant**	**What has happened?**	**Comment**
DocumentAnalyzer	**AnalyzeAndRecognizeBlocks**	Removed	This method was generally used to perform layout analysis

			inside blocks of the autoanalysis type. As this type of block is no longer supported, the method has been removed. You can use the **AnalyzeRegion** method, if layout analysis must be performed in an image zone, and then call the **RecognizeBlocks** method.
	AnalyzeAndRecognizePage, AnalyzeAndRecognizePages, RecognizeImageDocumentAsPlainText	The number of input parameters has been changed.	The **SynthesisParamsForPage** object is new in the set of input parameters.
	ExtractBarcodes	The input parameters have been changed.	The **BarcodeParams** object is used as an input parameter instead of the **BarcodeAnalysisParams** object. The **ObjectsExtractionParams** object is new in the set of input parameters.
	RecognizeBlocks	The input parameters have been changed.	The **SynthesisParamsForPage** and **LayoutBlocks** objects are used as input parameters instead of the **PageSynthesisParams** and **BlocksCollection** objects, respectively. The **ObjectsExtractionParams** object is new in the set of input parameters.
	RecognizePage	The input parameters have been changed.	The **SynthesisParamsForPage** object is used as input parameter instead of the **PageSynthesisParams** object. The **ObjectsExtractionParams** object is new in the set of input parameters.
	RecognizePages	The input parameters have been changed.	The **SynthesisParamsForPage** and **ObjectsExtractionParams** objects are new in the set of input parameters. The **PageProcessingParams** object is no longer used as the input parameter.
	StraightenLines	Removed	Use the RemoveGeometricalDistortions method of the DocumentAnalyzer object instead.
	PerformEnhancement, EnhanceImageBlocks	Removed	Image enhancement is no longer supported.

IDocumentAnalyzerEvents	ReportPercentage, ReportRecognizerTip, ReportRecognizedRect	Removed	Use the OnProgress, OnRecognizerTip, OnRegionProcessed methods of the IDocumentAnalyzerEvents object instead.

Scanning

Object/Enumeration	Property/Method/Constant	What has happened?	Comment
ScanSourceSettings	PaperBottom, PaperRight	The default values have been changed.	In this version the scanning area rectangle is not set (all the properties **PaperBottom**, **PaperLeft**, **PaperRight**, **PaperTop** are set to 0 by default). In this case, the scanning area will be selected by the scanner. In most cases it will be the whole available scanning area.

License-related objects

The collection of available (activated) licenses you can receive using the **Licenses** property of the **Engine** object. The **IEngine::CreateLicense** method is no longer supported.

Object/Enumeration	Property/Method/Constant	What has happened?	Comment
License	**IsActivated**	Removed	Only activated licenses are available.
	IsAbsoluteTimeLimitationUsed **IsRelativeTimeLimitationUsed** **RelativeDays**	Removed	For activated licenses use the **ExpirationDate** method of the **License** object.
	CounterMeasureUnit	Removed	A license may have several counters with different measuring units.
	LimitationPeriod **RemainingUnits** **UnitsPerPeriod**	Removed	Use the **VolumeRefreshingPeriod**, **VolumeRemaining**, **Volume** properties instead, respectively. These properties take as the input parameter a **LicenseCounterTypeEnum** constant as a license may have several counters with different measuring units.
AEM_ prefixed flags		The corresponding module has been renamed.	The new name of the module is AvailableEngineModulesFlags.
	AEM_2DBarcodePDF417 AEM_FullTextSeachDA	Renamed	These flags have been renamed to AEM_PDF417 and AEM_FullTextIndexDA, respectively.
	AEM_CJK AEM_Thai AEM_Vietnames AEM_Hebrew AEM_FineReaderXIX AEM_LanguageDatabase	Removed	The corresponding **ALS_** prefixed flags are used.
AEF_ prefixed flags		The corresponding module has been	The new name of the module is AvailableExportFormatesFlags.

		renamed.	
	AEF_DBF	Removed	Export to DBF format is no longer supported.
ATT_ prefixed flags		The corresponding module has been renamed.	The new name of the module is AvailableTextTypesFlags.
AVC_ prefixed flags		The corresponding module has been renamed.	The new name of the module is AvailableVisualComponentsFlags.
LicenseLimitationPeriodEnum		Renamed	The new name is **VolumeRefreshingPeriodEnum**. It provides the same functionality with the following exception: • LLP_Hour — removed. This volume refreshing period is no longer supported.
LicenseCounterMeasuringUnitEnum		Renamed	The new name is **LicenseCounterTypeEnum**.

See also

Specifications
What's New in ABBYY FineReader Engine 10

Version History

Below you can find features overview from version 5.0 to 9.0.

What's New in ABBYY FineReader Engine 5.0 (Released: 05/2001)	
• Recognition quality improved by 1.5-2 times compared to 4.0 version • Saves in HTML and PDF format with full page layout retention • Full text color retention • Recognition of subscript characters and simple chemical formulas • Vertical text recognition and recognition of pictures "embedded" in table cells • Dual page splitting	• 176 recognition languages, including programming languages Basic, C/C++, COBOL, Fortran, JAVA, Pascal, and new language dictionaries • Component Object Model (COM) API accessible from any development environment supporting COM interface (Visual Basic, C/C++ etc.) • API to create user languages and dictionaries • Tools for training of user patterns for machine print characters via FineReader training dialog • New HTML Help with context-sensitive topics accessible directly from VB Object Browser
What's New in ABBYY FineReader Engine 6.0 (Released: 08/2002)	
• Improved algorithm for the recognition of poor print quality documents. The improved algorithm incorporates a new adaptive image binarization method and a new method of background removal, and is particularly effective in the case of images scanned in "gray" mode.	• New recognition fonts are supported: OCR-A, OCR-B and MICR (E13B). • Fast mode available in all FineReader 6.0 Engine versions except the FineReader 6.0 Engine Standard. This mode provides faster recognition with worse image quality

- New PDF saving mode — "Image only"
- Save text alignment in Excel format
- Save nonrectangular pictures in RTF format, recreate bullets and numbering
- 177 recognition languages

- New features in ASCII version: the ability to preprocess image files, to recognize multipage image files, to work with memory images
- New Licence Manager utility

What's New in ABBYY FineReader Engine 7.0/7.1 (Released: 07/2004)

- Recognition quality improved by approximately 25%
- Opening and processing of PDF files
- New recognition languages: Traditional Chinese, Simplified Chinese and Japanese languages
- Old European languages have been added: Old English, Old French, Old German, Old Italian, and Old Spanish
- Recognition of Fraktur/Black Letter fonts
- Support for JPEG2000 part 1
- Opening a selected page from a multipage TIFF or PDF file
- New method for analysis and recognition of barcodes
- Support for new types of 1D barcodes: CODABAR without checksum, UCC Code 128, Industrial 2of5, IATA 2of5, Matrix 2of5, Code 93, UPC-A, and UPC-E
- Microsoft Word XML and ASCII XML output
- Export to MS PowerPoint
- Improved DA for invoices; detection of page orientation; 1D barcode detection, including detection of barcodes at any angle

- Improved detection and analysis of tables, particularly of tables without printed grid lines and tables with color rows and columns
- Improved adaptive binarization and background filtering
- New dictionaries added: law and medical dictionaries for the languages English and German
- Saving recognition results as linearized PDF files: the user will see the first pages of a PDF before the entire file has been downloaded
- Improved saving of edited texts in PDF format
- Numerous improvements of export to HTML and RTF formats
- Network runtime licences available
- Support for form and semi-structured document processing with support for ABBYY FormReader and FlexiCapture
- New recognition languages for ICR: Hungarian, Greek, and Croatian
- Arabic ICR digits
- Fast Mode for ICR

What's New in ABBYY FineReader Engine 8.0/8.1 (Released: 09/2005)

- Voting API support
- Field-level recognition enhancements: fast mode for ICR, better text extraction from underlined fields, text block despeckling, better results on fields with spaces, dictionary with space-containing words
- PDF/A Support
- Up to 30 percent accuracy improvement on low resolution documents and faxes
- Up to 40 percent accuracy improvement on documents captured by using a digital camera
- Ability to straighten text lines on images taken by digital cameras

- Support for New Barcode Type – EAN 13 Supplemental
- CMC7 Text Type Support
- Additional Support for external dictionaries
- Forms and semi-structured documents processing improvements
- Ability to load Engine's subsystems on demand or preliminarily
- Ability to get all possible hypotheses for recognized words and characters
- Ability to trace Engine's calls in a log file

- New input image formats (GIF and DjVu)
- Balanced Processing Mode for OCR
- New Document Analysis for Full Text Indexing
- Improved PDF processing and creation, up to 2 times faster processing, accuracy improvement, enhanced security options, tagged PDF files, control of PDF page sizes
- "On the fly" core recognition tuning
- New Language for OCR: Thai
- New Language for OCR: Hebrew
- Expanded Asian Language Support for PDF and RTF Export
- Saving External Data in Engine Profiles

What's New in ABBYY FineReader Engine 9.0 (Released: 10/2008)

- **Adaptive Document Recognition Technology (ADRT):**
  Documents generated by ADRT have consistent formatting across all pages of a document since they are processed as a unit.
- **Multi-Page processing through new Document specific API:**
  The new API objects allow you to set up the parameters of page and document synthesis separately.
- **Multi CPU / Multi Core Recognition Architecture:**
  Utilises all CPU cores during analysis and recognition of multi page documents.
- **New 2D barcode types: Aztec 2D, Data Matrix 2D, QR Code 2D**
- **New Image preprocessing capabilities:**
  Detection of an image rotation up to 20 degrees, deskew by horizontal and vertical pairs of black squares, lines and lines of text
- **Visual Components – Scan Interface, Document Viewer, Image Viewer, Text Editor, Text Validator:**
  Developers can give users direct but controlled access to recognition results and functions for validation or checking of documents.
- **Improved Asian Language OCR Support for Chinese, Japanese and Korean**
- **XML-based Office 2007 File Formats: DOCX, XLSX,**
  Export recognised documents to the new, open, interoperable, robust XML based formats that were introduced in Microsoft Office 2007
- **MRC (Mixed Raster Content) Compression for PDF and**
  PDF/A:
  MRC compression achieves significantly better file compression without visible degradation of document representation. Significant reduced file size, up to 10 times smaller compared to JPEG compression.
- **Licensing: Extended CPU Core Support**
  New licensing scheme allows an unlimited number of cores with page limited licences.
- **Licensing: CPU core based licences**
  New offer of licences without a page counter, Pricing is based on the maximum number of CPU cores that can be used instead
- **Licensing: Maximum Speed Limitation**
- **Data capture functionality**, which was previously offered as the FormReader batch and FlexiLayout processing add-ons will soon be available through a separate Engine SDK – ABBYY FlexiCapture Engine. Please contact your ABBYY Sales representative for more information.

System Requirements

ABBYY FineReader Engine 10 Requirements

- PC with x86-compatible processor (1 GHz or higher).
- Operating System: Microsoft Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and 64-bit versions of Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP.
- Memory:
  - for processing one-page documents — minimum 400 MB RAM, recommended 1 GB RAM;
  - for processing multi-page documents — minimum 1 GB RAM, recommended 1,5 GB RAM.

- Hard disk space: 800 MB for library installation and 100 MB for program operation plus additional 15Mb for every processing page of a multi-page document.
- 100% TWAIN-compatible scanner, digital camera, or fax modem — for scanning only.
- Video card and monitor (min. resolution 1024*768 — for pattern training, dictionary editing, scanning with a GUI displayed)
- Keyboard, mouse or other input device
- The following registry branches should be accessible from the workstation:
  - "HKEY_CURRENT_USER\Software\ABBYY\SDK\10\FineReader Engine" — full control
  - "HKEY_CURRENT_USER\Software\ABBYY\SDK\10" — full control for installation only
  - "HKEY_LOCAL_MACHINE\Software\ABBYY\SDK\10" — full control for installation only
- The following folders should be accessible from the workstation:
  - Folder with ABBYY FineReader Engine binary files — access for reading
  - %TEMP% folder — full control access
  - %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\Licenses — full control access
  - %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\FineReader Engine — full control access
- The following components should be installed:
  - Microsoft Internet Explorer 5.0 or higher
  - If your application uses pattern training, dictionary editing, scanning with a GUI displayed, Windows Common Controls must have version 5.80 or later and Rich Edit Control must have version 3.0 or later

ABBYY SDK 10 License Server Requirements

- PC with x86-compatible processor (1 GHz or higher).
- Operating System: Microsoft Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, Windows 2000, and 64-bit versions of Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP.
- 25 MB of free hard-disk space
- The folder %ALLUSERSPROFILE%\Application Data\ABBYY\SDK\10\Licenses must have full control access

Frequently Asked Questions

Licensing and distribution

Is there a special installation program for distribution ABBYY FineReader Engine on a workstation?

ABBYY FineReader Engine 10 does not have special installation program for distribution. See Distribution of Applications Which Use the ABBYY FineReader Engine Library.

What license is required for compiling an application?

Your application must be compiled with a Developer License rather than a Runtime License. See Distribution of Applications Which Use the ABBYY FineReader Engine Library.

What license is required for activating ABBYY FineReader Engine on a workstation?

You should activate a Runtime License on the workstation. See Activating the ABBYY FineReader Engine Library with the Runtime License.

What ABBYY FineReader Engine files should be copied on a workstation?

See Installing the ABBYY FineReader Engine Library in Manual Mode.

Which folders should be accessible from the workstation?

The following folders should be accessible from the workstation:

See Installing the ABBYY FineReader Engine Library.

The application is run on a workstation with an activated Runtime License. The message saying "ABBYY FineReader Engine is not licensed." is displayed. What should I do?

Please, make sure that the Runtime Licenses corresponds to the Developer License. If the licenses do not match, the application will not work.
The **GetEngineObject** function requires a developer serial number to work. Make sure that the serial number used by the **GetEngineObject** function is the developer serial number.

See also Licensing, Distribution of Applications Which Use the ABBYY FineReader Engine Library.

Image-related questions

How can I remove background noise from each block separately?

Use the methods of the **ImageDocument** object, which improves image quality. These methods allow you to select an image region to work with.

Why does the OpenImage method not open an image file?

This method allows you to open images in ABBYY FineReader Engine internal format. Images in other formats cannot be opened using this method.

What's the difference between the ChangeResolution method of the ImageDocument object and the OverwriteResolution property of the PrepareImageMode object?

The **ChangeResolution** method changes the resolution of an already opened image. If the **OverwriteResolution** property of the **PrepareImageMode** object is set to TRUE, upon opening the image the program will use the resolution set in the **XResolutionToOverwrite** and **YResolutionToOverwrite** properties for image preprocessing (i.e. for binarization, deskewing, etc.).

See also ImageDocument.

How to open one page of a multi-page file?

You can use **PrepareImage** method of **Engine** object to open one page of a multi-page file:

```
Method PrepareImage(
  fileName          As String,
  destinationFolder As String,
  prepareMode       As PrepareImageMode,
  pageNumber        As Long,
  passwordCallback  As ImagePasswordCallback,
  documentInfo      As DocumentInfo
) As StringsCollection
```

The fourth parameter is *pageNumber*. This parameter contains the number of page to process (zero-based). This parameter is optional and may be -1, in which case all pages of the image file are extracted.

Using the Engine object

What should I do if I have got problems creating the Engine object in C#.NET?

You must make sure to specify [STAThread] (single-thread apartment model) as an attribute on your app's main function:

```
[STAThread]
public static void Main()
{
  ...
}
```

What should I do if the Engine object cannot be deinitialized in Delphi?

See Using ABBYY FineReader Engine in Delphi

How to work with read-only object properties in raw C++?

Certain ABBYY FineReader Engine objects (for example, **ILayout::Blocks**) have read-only object properties. Such properties cannot be changed directly in raw C++. If you want to change such a property, you need to pass a reference to the property object to a new variable, and then use this variable to change it. Below you can see a C++ sample for the **ILayout::Blocks** property which is represented by a read-only collection:

```
ILayout* pLayout = 0;
ILayoutBlocks* pLayoutBlocks = 0;
long blockIndex;
...
// The pLayoutBlocks variable receives a reference to the blocks collection from Layout
pLayout->get_Blocks( &pLayoutBlocks );
// Remove an element from the blocks collection
pLayoutBlocks->Remove( blockIndex );
```

Is it possible to run and use Engine object in several threads?

No, it is impossible. The **Engine** object is singleton, so only one object of this type may be created in a single instance of the application that uses ABBYY FineReader Engine. The methods of all ABBYY FineReader Engine objects should be called only from the thread in which **Engine** object was created.

See also the description of the **GetEngineObject** function and the **Engine** object.

Is it possible to create and run the Engine object on a multi-processor system?

Yes, it is possible. Please, see the description of the **MultiProcessingParams** object.

What should I do if the "Engine deinitialization failed" exception is thrown during deinitialization of the Engine object?

This exception is thrown if not all the objects which were created and used by the application have been deleted before the deinitialization of the **Engine** object. If all the objects have been deleted the exception may be caused by the scavenger operation.

If the application is developed in Visual Basic .Net:

In this environment, all objects with the *Nothing* value are not deleted, they are only marked for deletion. The exact moment when the garbage collector deletes the object is not known. Therefore, you should call the following methods before deinitializing the **Engine** object so that the garbage collector deletes the object:

```
   GC.Collect()
   GC.WaitForPendingFinalizers()
```

If the application is developed in Delphi:

See the Using ABBYY FineReader Engine in Delphi section.

You can use the **StartLogging** method of the **Engine** object to get the list of objects that have not been deleted.

How can I create a log file to keep track of all errors, warnings and method calls of ABBYY FineReader Engine?

To do this, you need to call the **StartLogging** method of the **Engine** object. As input parameters, specify the log file name and the Boolean variable which determines whether method call messages should be logged or not. Once you have called this method, all messages will be logged. To stop logging, call the **StopLogging** method of the **Engine** object.

Recognition-related questions

What should I do if I get an access violation error when working with recognition results?

Make sure that the **FRDocument** object has not been released before the method which leads to the error is called. Pointers to child object's interfaces are valid until the **FRDocument** object exists. An attempt to access a child object after its parent object has been destroyed may result in error. Please, see for details Working with Properties.

What recognition language is used by default?

English is the default recognition language. If you want to change the default recognition language, you must use the **SetPredefinedTextLanguage** method of the **RecognizerParams** object.

How can I improve the quality of recognition of blocks which contain different types of text?

If a block contains text of different types, ABBYY FineReader Engine will still treat it as text of the same type. To improve the quality of OCR, draw a separate block for text of each type.

See also Using Text Type Autodetection.

Why italic fonts and superscript/subscript are not recognized by autodetection?

If the **TextTypes** property of the **RecognizerParams** object contains any combination of TT_MATRIX, TT_TYPEWRITER, TT_OCR_A, and TT_OCR_B, then italic fonts and superscript/subscript will not be recognized, regardless of the values of the **ProhibitItalic**, **ProhibitSubscript** and **ProhibitSuperscript** properties of the **RecognizerParams** object.

See also Using Text Type Autodetection.

What is the difference between the CharConfidence and the IsSuspicious properties?

The **CharConfidence** property of the **PlainText** and the **CharacterRecognitionVariant** objects is the read-only long property which stores the value of character confidence. It is in the range from 0 to 100, and 255 means that confidence is undefined. It represents an estimate of recognition confidence of a character in percentage points. The greater its value, the greater the confidence. Character confidence can be undefined, for example, for characters which were added during text editing.

Recognition confidence of a character image is a numerical estimate of the similarity of this image and the "ideal" whose recognition confidence would be 100%. When recognizing a character, the program provides several recognition variants which are ranked by their confidence values. For example, an image of the letter "e" may be recognized

The sum total of the confidence values of all the recognition variants of a character need not be 100%. The hypothesis with a higher confidence rating is selected as the recognition result. But the choice also depends on the context (i.e. the word to which the character belongs) and the results of a differential comparison. For example, if the word with the “e” hypothesis is not a dictionary word while the word with the “c” hypothesis is a dictionary word, the latter will be selected as the recognition result, and its confidence rating will be 85%. The rest of the recognition variants can be obtained as hypotheses.

The **IsSuspicious** property of the **CharParams** object is the Boolean property. This property set to TRUE means that the character was recognized unreliably. This property is determined by an algorithm which takes into account a number of parameters, such as recognition confidence of a character, neighboring characters and their recognition confidence, hypotheses and their recognition confidence, the geometric parameters of a character, and context (i.e. the word to which a character belongs).

Other questions

Where do the scan log files locate?

There are two scan log files: scantwain.txt and scanwia.txt. They are stored in the %userprofile%\Local Settings\Application Data\ABBYY\ScanManager\11.00 folder.

How to change scanning settings?

You can use the **ScanSourceSettings** property of the **ScanManager** object to access to the **ScanSourceSettings** object. This object provides access to the scanning settings of a source.

See also Setting up Scanning Options.

Which PDF versions can recognized text be exported to?

A minimal version of the PDF file which matches the specified properties of the **PDFEncryptionInfo** object and the **IPDFExportParamsOld::WriteTaggedPDF** property is selected as the version of the PDF file.

See also ABBYY FineReader Engine 10 Modules, **PDFExportParamsOld**.

If you cannot find the answer to your question, please contact the ABBYY Technical Support.

Contact ABBYY

In this section you can find the contacts of ABBYY sales offices and technical support:

- How to Buy
- Technical Support

How to Buy ABBYY FineReader Engine 10

You can order ABBYY FineReader Engine by contacting our offices at the following addresses:

- ABBYY Russia: engine@abbyy.com
- ABBYY USA: sales@abbyyusa.com
- ABBYY Europe: engine_eu@abbyy.com
- ABBYY Ukraine: engine@abbyy.ua

You can purchase additional language support applications and fonts at www.paratype.com/shop.

Technical Support

If you have any questions regarding the use of ABBYY FineReader Engine 10, first of all consult the documentation provided with this product (this Developer's Help and the Readme file). Useful information can also be found in the technical support section of our Web site at www.abbyy.com.

If you cannot find the answer to your question, please contact the ABBYY office serving your region by e-mail. Please provide the following information when contacting technical support:

- your first and last name;
- the name of your organization;
- your phone number (or fax, or e-mail);
- the serial number of your ABBYY FineReader Engine 10 package;
- the protection type of your ABBYY FineReader Engine 10 package (software or hardware);
- the build number (to determine the build number, see the Introducing ABBYY FineReader Engine 10 page of this Help, or Properties in the FREngine.dll local menu);
- a description of the problem;
- a project that demonstrates the problem (with the necessary data files). This may be a slightly modified ABBYY FineReader Engine sample. We recommend that you compress the files using any popular archiving program (WinZIP, WinRAR, etc.);
- the name of your development tool;
- the type of your computer and processor;
- the version of your Windows operating system.

You can gather some of the above information automatically:

1. Run the **AInfo** utility (AInfo.exe) from the **<Installation folder>/Bin/Support** folder.
2. A dialog box will open displaying some of the above information. Save this information to a ZIP file.
   Note: No personal information or information about the user’s computer is collected. You can view all the saved information in the created archive.

You can also provide any additional information you consider important.

Support contacts:

Customers from USA, Canada, Japan, Mexico or other Central American countries, please contact:
ABBYY USA at dev_support@abbyyusa.com

Customers from Austria, Benelux, Denmark, France, Germany, Greece, Italy, Ireland, Norway, Portugal, Spain, Sweden, Switzerland, the United Kingdom or other Western European countries, please contact:
ABBYY Europe GmbH at TechSupport_eu@abbyy.com

Customers from Albania, Bosnia & Herzegovina, Bulgaria, Croatia, Czech Republic, Hungary, Israel, Macedonia, Moldova, Montenegro, Poland, Romania, Serbia, Slovakia, Slovenia, Turkey or Ukraine, please contact:
ABBYY Ukraine at engine_support@abbyy.ua

Customers from the countries not mentioned above, please contact:
ABBYY Russia at SDK_Support@abbyy.com