

ABBYY® Mobile OCR Engine 4

USER'S GUIDE

Table of Contents

Introducing ABBYY Mobile OCR Engine 4.....	5
Guided Tour	6
How to Use the ABBYY Mobile OCR Engine Native Library	6
Recognizing Business Cards.....	9
Working with Languages.....	9
Working with Dictionaries.....	10
Recognizing with Custom Languages	11
Working with Regular Expressions.....	11
Recognizing in MICR Mode	13
Description of the ABBYY Mobile OCR Engine Native Sample	13
Description of the Demo Tool	14
<i>Recognition Settings Dialog Box</i>	15
Tips for Taking Photos.....	17
Native Library API Reference.....	20
Types in ABBYY Mobile OCR Engine Native Library	20
Standard Return Codes of ABBYY Mobile OCR Engine Functions	21
List of the ABBYY Mobile OCR Engine Functions	21
<i>FineAllocMemory Function</i>	22
<i>FineAnalyzeTextAsBusinessCard Function</i>	23
<i>FineAreCjkLanguagesSupported Function</i>	23
<i>FineDeinitialize Function</i>	23
<i>FineExecutionLogFunction Function</i>	24
<i>FineExtractBarcodes Function</i>	24
<i>FineFreeMemory Function</i>	25
<i>FineGetLastErrorMessage Function</i>	25
<i>FineGetLicenseInfo Function</i>	26
<i>FineGetTextLines Function</i>	26
<i>FineGetVersionInfo Function</i>	27
<i>FineGetWordSuggest Function</i>	27
<i>FineInitialize Function</i>	28
<i>FineLoadImageFromFile Function</i>	28
<i>FineLoadImageFromInputStream Function</i>	29
<i>FinePrebuildWordsInfo Function</i>	30
<i>FinePreprocessImage Function</i>	30
<i>FineRecognizeBarcode Function</i>	31
<i>FineRecognizeBusinessCard Function</i>	32
<i>FineRecognizeImage Function</i>	34
<i>FineRecognizeRegion Function</i>	35
<i>FineSetLicenseInfo Function</i>	36
<i>FineSetRecognizerThreadsCount Function</i>	37
<i>Callback Functions</i>	37
<i>TFinePrebuiltDataCallbackFunction</i>	37
<i>TFineProgressCallbackFunction</i>	38

<i>Custom Memory Management Functions</i>	39
<i>TFineAllocMemoryFunction Function</i>	39
<i>TFineFreeMemoryFunction Function</i>	39
Structures	40
<i>CFineAngle Structure</i>	40
<i>CFineBarcode Structure</i>	41
<i>CFineBcrField Structure</i>	41
<i>CFineBusinessCard Structure</i>	42
<i>CFineBusinessCard Structure</i>	42
<i>CFineImage Structure</i>	42
<i>CFineImageFile Structure</i>	43
<i>GetLength Method of CFineImageFile Structure</i>	43
<i>Read Method of CFineImageFile Structure</i>	44
<i>Seek Method of CFineImageFile Structure</i>	44
<i>CFineImageInputStream Structure</i>	45
<i>Read Method of CFineImageInputStream Structure</i>	45
<i>Skip Method of CFineImageInputStream Structure</i>	45
<i>CFineImageTransformationInfo Structure</i>	46
<i>CFineLayout Structure</i>	46
<i>CFineLicenseInfo Structure</i>	47
<i>CFinePrebuiltLayoutInfo Structure</i>	47
<i>CFinePrebuiltTextBlockInfo Structure</i>	48
<i>CFinePrebuiltTextLineInfo Structure</i>	48
<i>CFineRects Structure</i>	48
<i>CFineTextBlock Structure</i>	49
<i>CFineTextCharacter Structure</i>	49
<i>CFineTextLine Structure</i>	50
<i>CFineWarningDataWrongLanguages Structure</i>	50
<i>CFineWordInfo Structure</i>	51
<i>CFineWordSuggestion Structure</i>	51
<i>CFineWordVariant Structure</i>	52
Enumerations	52
<i>BIT_FLAG Macros</i>	53
<i>TFineCharacterAttributes</i>	53
<i>TBcrFieldType</i>	54
<i>TFineBarcodeOrientation</i>	55
<i>TFineBarcodeSupplement</i>	55
<i>TFineBarcodeType</i>	56
<i>TFineCharacterAttributes</i>	57
<i>TFineImageFileSeekPosition</i>	58
<i>TFineImageLoadingOptionsFlags</i>	58
<i>TFineImageProcessingOptionsFlags</i>	58
<i>TFinePrebuiltDataType</i>	60
<i>TFineRecognitionConfidenceLevel</i>	61
<i>TFineRecognitionMode</i>	61
<i>TFineRotationType</i>	62
<i>TFineSupportedCodepage</i>	62

<i>TFineCharacterAttributes</i>	63
<i>TFineTextCharacterQuality</i>	64
<i>TFineWarningCode</i>	64
<i>TFineWordAttributes</i>	65
<i>TFineWordVariantType</i>	65
<i>TLanguageID</i>	65
Licensing	68
Copyright and Trademark Notices	68
Specifications	70
Recognition Languages in ABBYY Mobile OCR Engine	70
Supported Image Formats.....	72
Barcode Types.....	72
System Requirements	74
Backward Compatibility Issues of ABBYY Mobile OCR Engine	74
<i>Compatibility of ABBYY Mobile OCR Engine 4 release 14 with previous releases</i>	74
<i>Compatibility of ABBYY Mobile OCR Engine with version 3.0 and older</i>	78
ABBYY Mobile OCR Engine Distribution Kit.....	81
What's New in ABBYY Mobile OCR Engine 4 release 15	83
Contact ABBYY	84
How to Buy ABBYY Mobile OCR Engine 4	84
Technical Support	84

Introducing ABBYY Mobile OCR Engine 4

Use ABBYY Mobile OCR Engine 4 to develop fast, light and compact OCR applications for mobile devices. Based on ABBYY's world-famous cutting-edge OCR technologies, ABBYY Mobile OCR Engine 4 provides powerful algorithms for image processing and high-accuracy recognition and is optimized to provide high efficiency combined with low requirements for device resources:

- Upgraded memory management: optimal balance of speed and quality
- Compact code: it occupies from 8 MB of ROM and from 10 MB of RAM
- OCR technologies famous worldwide
- 62 recognition languages including Chinese, Japanese, and Korean
- Reliable development tool

ABBYY Mobile OCR Engine 4 is the ideal solution for developers of mobile and "light" applications that strive to increase the attractiveness of their products, upgrade and expand their functionality and application areas. ABBYY Mobile OCR Engine 4 is a leader among the technological products for developing applications for mobile devices.

Key features

- High quality and accuracy of recognition
- Recognition of multilingual documents
- Business card reading
- Barcode recognition
- Integration with popular mobile platforms
- Low resource requirements

This distribution provides you a possibility to test these features of ABBYY Mobile OCR Engine 4 using a special Demo Tool utility (TestShell.exe) and ABBYY Mobile OCR Engine 4 native library.

How to use this Help

In this Developer's Help you can find all the necessary information about testing ABBYY Mobile OCR Engine 4.

Guided Tour

See this section for quick-start instructions and articles about using different aspects of ABBYY Mobile OCR Engine.

Description of the Demo Tool

Here you can find detailed information about the Demo Tool utility (TestShell.exe) which demonstrates the work of the functions.

Native Library API Reference

This section contains the complete description of ABBYY Mobile OCR Engine native library API.

Licensing

The information about ABBYY Mobile OCR Engine license protection.

Specifications

The list of supported image formats, recognition languages, compatibility information, etc.

Contact ABBYY

Here you can find the contact information of the ABBYY offices and technical support service.

Guided Tour

This section contains information which will help you to start working with ABBYY Mobile OCR Engine and describes various special techniques:

- How to Use the Native Library
- Recognizing Business Cards
- Working with Languages
- Working with Dictionaries
- Recognizing with Custom Languages
- Working with Regular Expressions
- Recognizing in MICR Mode
- Description of the Native Sample
- Description of the Demo Tool
 - Recognition Settings Dialog Box
- Tips for Taking Photos

How to Use the ABBYY Mobile OCR Engine Native Library

The ABBYY Mobile OCR Engine native library may be used for testing. The ABBYY Mobile OCR Engine library supplied as DLL and static library and as a wrapper of the library for Android and iOS may be found in the appropriate distributions.

This section contains description of how to work with DLL and static library.

Go to the Native Library API Reference section for detailed description of the ABBYY Mobile OCR Engine native library functions and structures.

Loading the library

You start your work with ABBYY Mobile OCR Engine by loading the library with help of the **FineInitialize** function. It allows you to specify memory allocation/de-allocation functions and a logging function. Then you need to specify the license information (see the Licensing section for details).

Note: If you use a trial license, the word "ABBYY" will appear in every 20th line in the recognized text and in every third recognized business card.

1. [optional] Implement custom memory management functions and a logging function if you wish to use them.
2. Call the **FineInitialize** function, passing these functions as input parameters. If you do not wish to use custom functions, pass 0 for the corresponding parameters.
3. Load the license file into memory.
4. Create a **CFineLicenseInfo** structure. Assign the pointer to the loaded license data to the **LicenseData** field and the size of loaded data to the **DataLength** field. Assign the name of your application to the **ApplicationID** field. It must correspond to the name of the application that is specified in the license file.
5. Call the **FineSetLicenseInfo** function, passing to it the constant pointer to the **CFineLicenseInfo** structure you just configured.

6. After specifying the license you can work with the library until the **FineDeinitialize** function is called.

Sample code of library initialization

```
// Error handling is omitted

// Custom memory allocation function
static void* allocFunction( int size )
{
    void* res = malloc( size );
    return res;
}
// Custom memory release function
static void freeFunction( void* ptr )
{
    free( ptr );
}

// Initialize the library with custom memory management functions and without logging
const TFineErrorCode initializeLibError = FineInitialize( allocFunction, freeFunction,
0 );
...
void* loadedLicenseData = 0;
int licenseDataSize = 0;
wchar* appID; // set to your application ID
// Load the license file into memory at the loadedLicenseData address
...
CFineLicenseInfo licenseInfo;
licenseInfo.LicenseData = static_cast<BYTE*>( loadedLicenseData );
licenseInfo.DataLength = licenseDataSize;
licenseInfo.ApplicationId = appID;

const TFineErrorCode errorCode = FineSetLicenseInfo( &licenseInfo );
...
// Work with the library until the FineDeinitialize call
```

Important! All functions of the ABBYY Mobile OCR Engine library should only be called from the thread in which the library was initialized. You cannot initialize the library in several threads simultaneously without deinitialization.

Opening and processing the images

Below is a description of a typical procedure performed by ABBYY Mobile OCR Engine:

1. Load the image for recognition. You can open the image file with the help of the **FineLoadImageFromFile** function or load it from the input stream using the **FineLoadImageFromInputStream** function. These functions convert the image in JPEG or PNG format into **CFineImage** internal format of ABBYY Mobile OCR Engine. If you need to load an image in any other format, you must load it into memory and convert it into **CFineImage** format on your side. The functions which perform recognition work with the image in this format.
2. Recognize the image. We will use the **FineRecognizeImage** function as an example. Configure the parameters in the following way:
 - *languages*: Recognition languages, passed as an array of the **TLanguageID** constants. We do not recommend setting more than two recognition languages at once.
 - *patterns*: You need to load the pattern file which includes the description of the languages you need into memory and pass its address as **TFinePatternsPtr**. To find out which predefined pattern file you need for your languages, consult the

PatternsFilesInfo.txt file which you will find in the **data/Patterns** folder of the distribution package.

- *cjkPatterns*: If you need to recognize CJK languages, load the pattern files for those languages into memory in the same way and create a zero-terminated array of **TFinePatternsPtr** variables with the addresses of loaded patterns. Otherwise, pass an array which contains only one zero pointer.
 - *dictionaries*: Load the dictionary files for the languages you use. Attaching the dictionary will improve recognition quality. See Recognition Languages in ABBYY Mobile OCR Engine for information on which languages have built-in dictionary support. Create a zero-terminated array of **TFineDictionaryPtr** variables with the addresses of loaded dictionaries. If you do not use dictionaries, pass an array which contains only one zero pointer (do not pass zero instead of array).
 - *image*: Pass the **CFineImage** structure you obtained in the first step.
 - *imageProcessingOptions*: Select the image processing options and pass the OR combination of appropriate **TFineImageProcessingOptionsFlags** constants. For default option, pass 0.
 - *recMode*: Choose the recognition mode and specify the appropriate **TFineRecognitionMode** constant.
 - *confidenceLevel*: Choose the level of marking characters as uncertain (**TFineRecognitionConfidenceLevel**).
 - *layoutBuff*: The result of processing. Create a **CFineLayout*** pointer variable which will receive the recognition results.
 - *rotation*: Create a **TFineRotationType** variable which will receive the information about image rotation. You can pass 0 as this parameter if you do not use the recognized text coordinates.
 - *progressCallback*: Implement the **TFineProgressCallbackFunction** function if you need to receive information about the operation progress or be able to cancel the operation. You can pass zero pointer if you do not use this callback.
 - *prebuiltDataCallback*: Implement the **TFinePrebuiltDataCallbackFunction** function if you need to receive information about the image rotation and detected text blocks before the processing is completed, for example, to display the text blocks while the recognition is going on. You can pass zero pointer if you do not use this callback.
3. The **CFineLayout** structure you receive after the operation is completed contains all recognized text and its coordinates. Iterate through the blocks, the text lines within them, and the characters within text lines; save the text in the format you need, search it for keywords or work with it in any other way.

Note: By default, recognition operations will be performed in parallel, using up to 4 threads. You can change this limitation by calling the **FineSetRecognizerThreadsCount** after library initialization. Pass 1 for the *threadsCount* parameter to turn off multi-threading, or increase the number of threads if you need faster processing. However, the number of threads working simultaneously can never exceed the number of CPU cores the device provides.

See also

Description of the Native Sample

Recognizing Business Cards

Business cards contain business information about a company or a person. Business cards can include person name, company, telephone numbers, fax, e-mail, website addresses and similar information. You may need to capture this information from paper business cards and save it in digital form. It can be the address book of a mobile phone, e-mail client, or any other data storage system.

In ABBYY Mobile OCR Engine a business card is represented by a set of fields, which can be of different types such as name, phone number, e-mail address. Fields of some types can also contain several components, e.g. name field can contain "first name", "middle name", and "last name" components. To extract the information you need, you can implement a procedure iterating through fields and performing different actions depending on the field type.

General recommendations

Business card recognition quality will be significantly improved if you add the English language to the list of recognition languages and use the English.akw keywords dictionary, even if the business cards you recognize are not in English.

Not all languages are provided with the keywords dictionaries necessary for business card recognition. See Recognition Languages in ABBYY Mobile OCR Engine for a full list.

Native library

The ABBYY Mobile OCR Engine native library provides the **FineRecognizeBusinessCard** function for recognizing an image as a business card. Follow these steps:

1. Load the image for recognition using the **FineLoadImageFromFile** or **FineLoadImageFromInputStream** functions. See How to Use the Native Library for details.
2. The parameters of the **FineRecognizeBusinessCard** function are the same as for the **FineRecognizeImage** function (described in How to Use the Native Library), with the following exceptions:
 - *languages*: We recommend including the English language (LID_English) in this parameter, even if your business cards are in some other language.
 - *keywords*: Load the keywords dictionaries for the languages you use. Add the English.akw dictionary, even if your business cards are not in English. Create a zero-terminated array of **TFineKeywordsPtr** variables with the addresses of loaded keywords dictionaries.
 - *businessCardBuffer*: The result is returned as **CFineBusinessCard**, not as **CFineLayout**. Create a **CFineBusinessCard*** pointer variable which will receive the recognition results.
3. Work with the result of recognition. The **CFineBusinessCard** structure contains an array of fields represented by **CFineBcrField** structure variables. Search through these fields for the information you need.
 Use the value of **CFineBcrField.Type** to check the type of field (represented by constants of **TBcrFieldType** enumeration).
 Use the value of **CFineBcrField.TextLines** to obtain the text of the business card field. It can contain several lines represented by **CFineTextLine** structures.

If your scenario requires recognizing different kinds of documents and detecting business cards among them, you can also use the **FineAnalyzeTextAsBusinessCard** function, which looks for business card fields in the text that has already been recognized.

See also

How to Use the Native Library

Working with Languages

One of the main recognition parameters is the language which is used during recognition. It is very important for good quality recognition results to set up the languages of a document correctly.

Any function you use for recognition takes a set of languages as an input parameter. In general we recommend not to add too many languages to the set. But, if you are recognizing business cards, adding the English language is highly recommended, even if the business cards are in another language.

Predefined and custom languages

ABBYY Mobile OCR Engine includes a set of predefined recognition languages. See the list in Recognition Languages in ABBYY Mobile OCR Engine. Some of these languages have dictionary support. Attaching a dictionary to the recognition language will improve the results' quality, but it is not mandatory if you are using the native library. See Working with Dictionaries for details.

You can also create a custom language which will allow only the words conforming to a specified regular expression. This can be useful if you need to extract some specific data from the images, such as telephone numbers or e-mail addresses, which are easily described by means of a regular expression. See Recognizing with Custom Languages and Working with Regular Expressions.

Patterns

The description of a recognition language (i.e. its ID, set of characters, etc.) is stored in a *pattern file* with the extension *.rom.

Important! A pattern file must include all languages that you are going to use for recognition.

If you use only predefined languages, you may choose one of the pattern files which are included in the distribution pack. You can find these files in the **data\Patterns** folder of the ABBYY Mobile OCR Engine installation folder. This folder contains pattern files for all languages, certain pairs, and groups. You can find the list of available pattern files and the languages they correspond to in the **PatternsFilesInfo.txt** file in the same folder. If none of the pattern files suits you, or if you are going to use custom languages, you need to create your own pattern file. To create a custom language, please contact support.

See also

How to Use the Native Library

Working with Dictionaries

ABBYY Mobile OCR Engine allows you to attach dictionaries to a recognition language, which greatly improves recognition quality.

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 Recognition Languages in ABBYY Mobile OCR Engine). These dictionaries are stored in dictionary files (*.edc) which are located in the **data\Dictionaries** folder of the ABBYY Mobile OCR Engine installation folder.
- **Keywords dictionary.** This dictionary is needed for business card recognition (BCR) and is provided for some predefined languages (see Recognition Languages in ABBYY Mobile OCR Engine). This dictionary contains words that appear most often on business cards, for example, "Phone", "Address", and etc. These dictionaries are stored in *.akw file located in the **data\BcrData** folder.

In the ABBYY Mobile OCR Engine native library, to use recognition functions (**FineRecognizeImage**, **FineRecognizeRegion**, and **FineRecognizeBusinessCard**), you must set up the list of dictionaries (the *dictionaries* parameter) and the list of keywords dictionaries (the *keywords* parameter in the **FineRecognizeBusinessCard** and **FineAnalyzeTextAsBusinessCard** functions).

Any predefined language (e.g. English) can be used without dictionary support. This is generally done to save memory, but the quality of recognition will deteriorate.

To use a predefined language without dictionary support, you need to pass as the *dictionaries* parameter an array which contains only one pointer to zero.

However, note that you cannot use business card recognition functions without at least one keywords dictionary. We strongly recommend adding the English keywords dictionary for recognition of business cards in any language. The results' quality will almost always improve.

See also

How to Use the ABBYY Mobile OCR Engine Library
 ABBYY Mobile OCR Engine Distribution Kit

Recognizing with Custom Languages

To recognize image with a custom language, you need to create a pattern file that contains all recognition languages that you are going to use, and you also have to know the IDs of these recognition languages and the name of the pattern file.

If you need to create a custom language, please contact support.

Why create a custom language

Here are some examples of scenarios in which a custom language improves recognition:

- If you need to extract some specific data from the images, such as telephone numbers or e-mail addresses, a custom language which will allow only words conforming to a specified regular expression can be created. Limiting the alphabet to exactly the set of symbols that occur can also be helpful. To explore the ABBYY Mobile OCR Engine regular expression alphabet see Working with Regular Expressions.
- If you recognize texts which use unusual vocabulary, e.g. contain many technical terms, they may not be recognized well using the inbuilt dictionary. In this case a custom language can be created which would be a copy of English, but work with a user-defined dictionary, so that the terms are recognized better.

Native library

If you use the **FineRecognizeImage**, **FineRecognizeRegion**, and **FineRecognizeBusinessCard** functions, which work with an image, you need to set up a list of recognition languages (the *languages* parameter) and a pattern file (the *patterns* parameter).

Note: The pattern file with custom language should be the first in the *patterns* parameter.

Each language has a unique ID (type of **TLanguageID**). If you attempt to use a language without or with a wrong pattern file, the recognition function will return the `FEC_InvalidArgument` error code.

See also

Working with Regular Expressions

Working with Regular Expressions

The ABBYY Mobile OCR Engine regular expression alphabet is described in the following table:

Item name	Conventional regular expression sign	Usage examples and explanations
Any character	.	<i>c.t</i> — denotes words like "cat", "cot"
Character from a character range	[]	<i>[b-d]ell</i> — denotes words like "bell", "cell", "dell" <i>[ty]ell</i> — denotes words "tell" and "yell".
Character out of a character	[^]	<i>[^y]ell</i> — denotes words like "dell", "cell", "tell", but forbids "yell" <i>[^n-s]ell</i> — denotes words like "bell", "cell", but forbids "nell", "oell", "pell", "qell", "rell" and "sell"

range		
Or		<i>c(a/u)t</i> — denotes words "cat" and "cut"
0 or more occurrences in a row	*	<i>10*</i> — denotes numbers 1, 10, 100, 1000 etc.
1 or more occurrences in a row	+	<i>10+</i> — allows numbers 10, 100, 1000 etc., but forbids 1.
Letter or digit	[0-9a-zA-Z]	<i>[0-9a-zA-Z]</i> — allows a single character; <i>[0-9a-zA-Z]+</i> — 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	
System character	@	
Word from dictionary	@(Dictionary)	<p>The Dictionary parameter sets the path to the user dictionary from which words must be taken. Backslashes in the path must be doubled. For example: <i>@(D: MyFolder MyDictionary.amd)</i>.</p> <p>Note: Some programming languages (such as C++) require you to escape backslashes in string literals. In this case you will need two escaped backslashes, which will result in a quadrupled backslash. The example above will look like this in C++:</p> <pre>L"@ (D: \\ \\ \\ MyFolder \\ \\ \\ MyDictionary.amd) "</pre>

Notes:

- 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 are 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, 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 parentheses. For example, *(a/b)+/c* denotes c and any combinations like abbbbaabbb, 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])/([12][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)[0-9][0-9])/([0-9][0-9])/((20)[0-9][0-9])/([0-9][0-9])*.

What is left is to combine all this together and separate the numbers by 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]))([12][0-9]))((30)(31))|. ((([0][1-9]))([10])(11)(12))|. (((19)[0-9][0-9]))([0-9][0-9]))((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

Recognizing with Custom Languages

Recognizing in MICR Mode

ABBYY Mobile OCR Engine supports Magnetic Ink Character Recognition (MICR) mode. A custom "MICR" language is defined for recognition of images in MICR Mode, and it is included in the **Micr.rom** pattern file which can be found in the **data\Patterns** folder of the distribution package. The MICR language has the "0123456789ABCD" alphabet and language ID equal to 1024.

Important: Only the MICR E13B characters are recognized in the MICR mode, all other fonts are ignored.

Native library

To enable MICR mode in the ABBYY Mobile OCR Engine native library, call a recognition method, e.g.,

FineRecognizeImage, with the following values:

1. *languages*: an array containing one constant, the MICR recognition language ID, which is 1024. In MICR mode you cannot add any more languages to the list.
2. *patterns*: the address of the **Micr.rom** file loaded into memory.
3. *imageProcessingOptions*: FIPO_MicrMode constant. You can pass an OR combination with some other constants, which contain the required settings as to geometry correction and other image transformations.

See also

How to Use the Native Library

Description of the ABBYY Mobile OCR Engine Native Sample

ABBYY Mobile OCR Engine includes a code sample written in C++ which illustrates the work with the native library. The code sample is located in the **\Sample\Generic\Sources\src** subfolder of the ABBYY Mobile OCR Engine folder.

The sample loads PNG image files from the **\Samples\Generic\SampleImages** folder with the help of the **FineLoadImageFromFile** function and recognizes them using the **FineRecognizeBusinessCard**, **FineRecognizeImage**, and **FineRecognizeBarcode** functions.

Note: If you use a trial license, the word "ABBYY" will appear in every 20th line in the recognized text and in every third recognized business card.

See also

How to Use the Native Library

Native Library API Reference

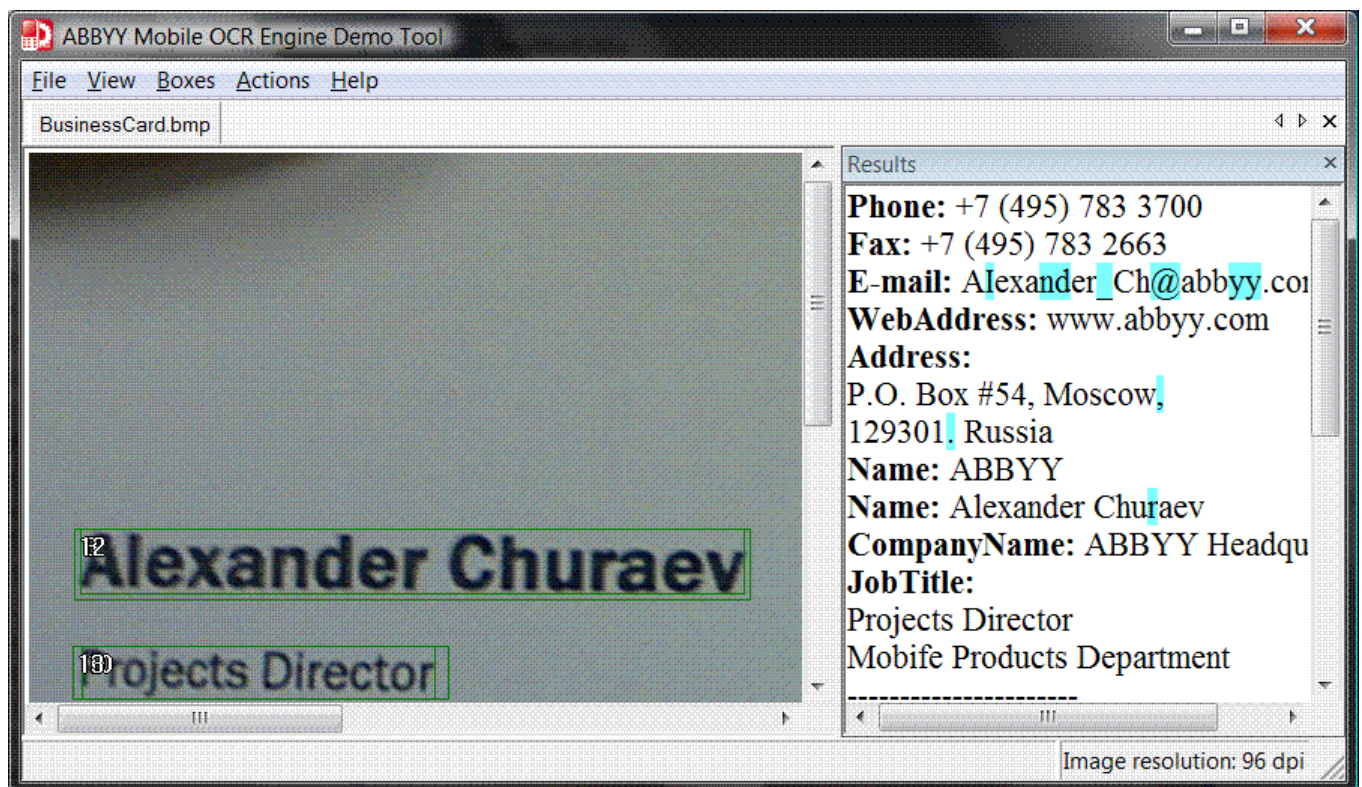
Description of the Demo Tool

ABBYY Mobile OCR Engine distribution kit includes a special Demo Tool utility (TestShell.exe) which demonstrates the work of the functions. This utility is located in the **\Tools.Windows** subfolder of the ABBYY Mobile OCR Engine folder.

The Demo Tool also performs the following functions:

- selecting input parameters of the functions (for example, the optimal memory size, which is necessary to recognize a specific image);
- viewing the results of recognition of various images;
- saving recognition results.

Main Window



Menu Bar

File Menu

In the **File** menu you can open an image (File>Open Image...)

Boxes Menu

The **Boxes** menu is designed for working with blocks, which have been detected during analysis or recognition. You can renumber, delete, and/or move blocks, change block types (Text or Picture), and save the current arrangements of blocks to a file or load it from a file.

Action Menu

The following items of the **Action** menu allow you to call the correspond ABBYY Mobile OCR Engine functions:

- Service functions:
 - You can select the Auto-initialize item and the library will be initialized automatically or select the **FineInitialize** and **FineDeinitialize** functions to initialize and deinitialize the library
 - Get last error message (the **FineGetLastErrorMessage** function)

- Get shell key license information (the **FineGetLicenseInfo** function)
- Get version (the **FineGetVersionInfo** function)
- Preprocess image (the **FinePreprocessImage** function)
- Preprocess words (the **FinePrebuildWordsInfo** function)
- Find all lines (the **FineGetTextLines** function)
- Find and recognize barcodes (the **FineExtractBarcodes** function)
- Recognize active block (the **FineRecognizeRegion** function)
- Recognize all blocks (the **FineRecognizeRegion** function)
- Recognize all blocks with words info (the **FineRecognizeRegion** function)
- Recognize image (the **FineRecognizeImage** function)
- Recognize business card (the **FineRecognizeBusinessCard** function)
- Generate word suggestions (the **FineGetWordSuggest** function)
- Recognize barcode (the **FineRecognizeBarcode** function)

By selecting the **Save Recognition Results...** item you can save the recognition results.

The **Recognition settings...** item opens a dialog box that allows you to set up image preprocessing and recognition parameters and select the optimal memory size which is necessary to recognize a specific image.

The **Barcode recognition setting...** item opens the corresponding dialog box that allows you to specify barcode type, orientation, and the other parameters.

See also

Recognition Settings Dialog Box

Recognition Settings Dialog Box

This dialog box allows you to set up image preprocessing and recognition parameters and select the optimal memory size which is necessary to recognize a specific image.

Option name	Option description
Language database path	A path to the language database (the textlang.dat file)
Recognition languages	Specifies the recognition languages. See Recognition Languages in ABBYY Mobile OCR Engine.
Analysis languages	Specifies languages which are used during image analysis.
Full dictionaries	The recognition languages which have full dictionary support are marked in Recognition Languages in ABBYY Mobile OCR Engine.
Image preprocessing options group	
Disable skew	Specifies whether the skew should be corrected.
Has CJK	Specifies whether the input image has Asian characters.
Find All Text	Specifies whether the program should find all text on the image.
Is European with some CJK	Specifies whether the input image contains European-language and some CJK text.
Detect page orientation	Specifies whether the program should detect page orientation.
Prohibit vertical CJK text	If this option is selected, the program will recognize only a horizontal CJK text on image, all vertical CJK text will be ignored.
MICR text type	Specifies whether the MICR E13B font must be recognized.
Confidence level	Specifies the recognition confidence level. If the level 0 is set, none of the uncertain characters are marked. If the level 4 is set, all suspicious characters are marked as uncertain. Level 3 is set as default.

Recognition mode	<p>Specifies the recognition mode:</p> <ul style="list-style-type: none"> • Full Full recognition mode • Fast This mode provides 25% faster recognition speed for European languages
Adjust image resolution	Allows you to adjust image resolution.
RAM size	Specifies the optimal memory size, which is necessary to recognize a specific image.

See also

Description of the Demo Tool

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

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.



Positioning the Camera

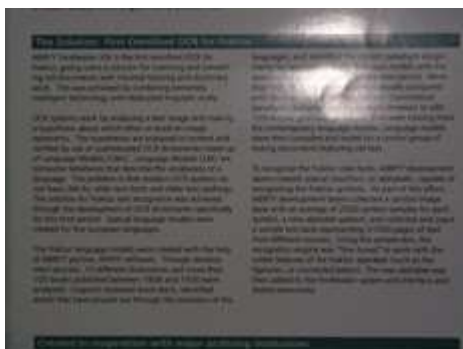
If possible, use a tripod. Position the lens parallel to the plane of the document and point it toward the center of the text.

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

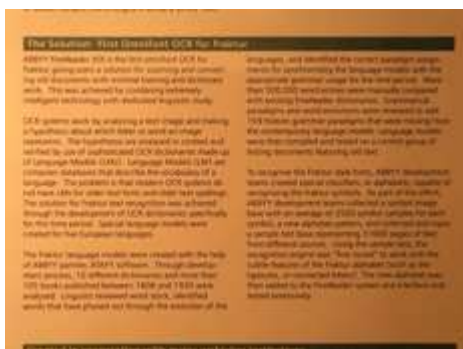
Whenever possible, turn off the flash to avoid glare and sharp shadows on the page. In poor lighting conditions, try using the flash from a distance of about 50 cm, or, preferably, use additional lighting.

Important! Using the flash when photographing documents printed on glossy paper causes the worst glare.



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.



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.

Native Library API Reference

This section contains the description of ABBYY Mobile OCR Engine native library:

- Using Types
- Return Codes
- Functions
 - Callback Functions
 - Custom Memory Management Functions
- Enumerations
- Structures

See also

How to Use the Native Library

Types in ABBYY Mobile OCR Engine Native Library

ABBYY Mobile OCR Engine native library functions use the following types:

Type	Description
BYTE	Byte (8 bits). <code>typedef unsigned char BYTE;</code>
DWORD	32-bit unsigned integer. <code>typedef unsigned long DWORD;</code>
RECT	This structure defines the coordinates of the upper-left and lower-right corners of a rectangle. <pre>typedef struct tagRECT { long left; long top; long right; long bottom; } RECT, *PRECT;</pre>
TFineDictionaryPtr ¹	ABBYY Mobile OCR Engine dictionaries. <code>typedef void* TFineDictionaryPtr;</code>
TFineImageLoadingOptions	The image processing options. <code>typedef DWORD TFineImageLoadingOptions;</code>
TFineImageProcessingOptions	The image processing options. <code>typedef DWORD TFineImageProcessingOptions;</code>
TFinePatternsPtr ¹	ABBYY Mobile OCR Engine patterns. <code>typedef void* TFinePatternsPtr;</code>
TFineKeywordsPtr ¹	ABBYY Mobile OCR Engine keywords dictionaries. <code>typedef void* TFineKeywordsPtr;</code>
WCHAR	Unicode character. <code>typedef WORD WCHAR;</code>
WORD	16-bit unsigned integer. <code>typedef unsigned short WORD;</code>

¹ — It is very important for ARM processors that absolute addresses corresponding to variables of these types are 4-byte aligned.

Standard Return Codes of ABBYY Mobile OCR Engine Functions

Below is the list of the standard return codes of the ABBYY Mobile OCR Engine functions.

```
typedef enum tagTFineErrorCode {
    FEC_NoError = 0,
    FEC_NotInitialized = 1,
    FEC_LicenseError = 2,
    FEC_InvalidArgument = 3,
    FEC_NotEnoughMemory = 5,
    FEC_InternalFailure = 6,
    FEC_TerminatedByCallback = 7,
    FEC_AlreadyInitialized = 8,
} TFineErrorCode;
```

Elements

Name	Description
FEC_NoError	The function is completed successfully.
FEC_NotInitialized	The library has not been initialized.
FEC_LicenseError	Unacceptable license information is used or the functionality is not available under the license.
FEC_InvalidArgument	One or more arguments are invalid. Use the FineGetLastErrorMessage function for diagnostics.
FEC_NotEnoughMemory	Not enough memory to perform the operation.
FEC_InternalFailure	An unspecified internal error.
FEC_TerminatedByCallback	The operation was terminated by the user via a callback function.
FEC_AlreadyInitialized	The library has already been initialized.

List of the ABBYY Mobile OCR Engine Functions

Function	Description
FineAllocMemory	Allocates memory.
FineAnalyzeImage	Analyzes the image and finds the text blocks on it.
FineAnalyzeTextAsBusinessCard	Detects business card fields in text lines returned from the FineRecognizeImage function.
FineAreCjkLanguagesSupported	Returns a non-zero value if the library supports the CJK language recognition.
FineDeinitialize	Deinitializes the ABBYY Mobile OCR Engine library.
FineExecutionLogFunction	Delivers to the client the information about execution.
FineExtractBarcodes	Finds and recognizes all barcodes on the image.
FineFreeMemory	Releases memory allocated for output buffer.
FineGetLastErrorMessage	Returns the last error message.
FineGetLicenseInfo	Returns information about the current license.
FineGetTextLines	Detects text lines on the image.

FineGetVersionInfo	Returns the version of the library.
FineGetWordSuggest	Generates a list of suggestions for the selected word from the specified dictionary.
FineInitialize	Initializes the ABBYY Mobile OCR Engine library.
FineLoadImageFromFile	Loads an image from a file.
FineLoadImageFromInputStream	Loads an image from the input stream.
FinePrebuildWordsInfo	Returns the document layout information, including rectangles of words, without the text recognition.
FinePreprocessImage	Binarizes an image.
FineRecognizeBarcode	Recognizes barcodes.
FineRecognizeBlocks	Recognizes a set of blocks on the image. Layout analysis is not performed.
FineRecognizeBusinessCard	Recognizes all text lines on the image and analyzes the image as a business card in one step.
FineRecognizeImage	Recognizes all text lines on the image.
FineRecognizeRegion	Recognizes all text lines in the specified region.
FineSetLicenseInfo	Sets the ABBYY Mobile OCR Engine the license information.
FineSetRecognizerThreadsCount	Limits the number of threads that can be used for multi-threaded processing.
<i>Callback Functions</i>	
TFinePrebuiltDataCallbackFunction	Delivers to the client the prebuilt information about the document layout, text blocks and lines before the text recognition.
TFineProgressCallbackFunction	Delivers to the client the information about the approximate percentage of analysis or recognition.
<i>Custom Memory Management Functions</i>	
TFineAllocMemoryFunction	Implemented on the client side. Custom memory allocation function.
TFineFreeMemoryFunction	Implemented on the client side. Custom memory release function.

FineAllocMemory Function

This function allocates memory. If during library initialization you specified a custom function for memory allocation, that function will be used.

C Syntax

```
TFineErrorCode FineAllocMemory(
    int    size,
    void** ptr
);
```

Parameters

size

[in] The size of memory buffer that needs to be allocated.

ptr

[out] A pointer to allocated memory.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineFreeMemory

FineInitialize

FineAnalyzeTextAsBusinessCard Function

This function finds business card fields in text lines returned from the **FineRecognizeImage** function.

C Syntax

```
TFineErrorCode FineAnalyzeTextAsBusinessCard(
    const TFineKeywordsPtr      keywords[],
    const CFineLayout*          layoutBuff,
    CFineBusinessCard**         businessCardBuffer,
    TFineProgressCallbackFunction progressCallback
);
```

Parameters

keywords[]

[in] The zero-terminated list of keywords dictionaries as a **TFineKeywordsPtr** variable. For the best result of business card recognition, add the English language keywords dictionary to the list, regardless of the language of the business card.

layoutBuff

[in] A reference to a **CFineLayout** variable which is output variable of the **FineRecognizeImage** function.

businessCardBuffer

[out] A pointer to pointer variable that receives the interface pointer of a **CFineBusinessCard** variable which represents a business card. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

FineAreCjkLanguagesSupported Function

This function returns a non-zero value if the library supports recognition of the CJK languages.

C Syntax

```
int FineAreCjkLanguagesSupported();
```

Return value

It returns a non-zero value if the library supports recognition of the CJK languages.

FineDeinitialize Function

This function deinitializes the ABBYY Mobile OCR Engine library.

C Syntax

```
TFineErrorCode FineDeinitialize();
```

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineInitialize

FineExecutionLogFunction Function

This function is implemented on the client side. It delivers to the client the information about execution. It is used as input parameter in the **FineInitialize** function.

C Syntax

```
void (*TFineExecutionLogFunction)( const WCHAR* str );
```

Parameters

str

[in] Log information.

See also

FineInitialize

FineExtractBarcodes Function

This function finds and recognizes all barcodes on the image.

C Syntax

```
TFineErrorCode FineExtractBarcodes(
    const CFineImage*          image,
    unsigned                   allowedTypes,
    unsigned                   allowedOrientations,
    unsigned                   allowedSupplements,
    int                        hasChecksum,
    int                        isCode39WithoutAsterisk,
    int                        isBinaryInterpretedAsText,
    TFineSupportedCodepage     defaultCodePage,
    CFineLayout**              layoutBuff,
    TFineProgressCallbackFunction progressCallback
);
```

Parameters

image

[in] The image to be recognized as a **CFineImage** variable.

allowedTypes

[in] The OR combination of the **TFineBarcodeType** constants that define acceptable barcode types.

allowedOrientations

[in] The OR combination of the **TFineBarcodeOrientation** constants that define the possible orientations of the barcode.

allowedSupplements

[in] The OR combination of the **TFineBarcodeSupplement** constants that define the possible supplements of the barcode. This parameter is ignored for barcodes without supplement. Set the parameter to FBS_Void if the barcode you recognize does not have a supplement.

hasChecksum

[in] Should not be zero if the barcode should be recognized as a barcode with checksum. It is valid for Code39, Interleaved25, Codabar, and Matrix25 barcodes. For these types of the barcodes, the last symbol of the barcode is considered as control sum of all barcode symbols, and is checked during the recognition.

isCode39WithoutAsterisk

[in] Should not be zero if the Code39 barcode has no start and stop symbol, the asterisk "*". It is valid for Code39 barcode. This parameter is ignored if the *allowedTypes* parameter set to more than one type.

isBinaryInterpretedAsText

[in] Should not be zero if byte data should be interpreted as text in the current code page. If this parameter is zero the data will be stored in hexadecimal format. This parameter is ignored if the *allowedTypes* parameter set to more than one type.

defaultCodePage

[in] A **TFineSupportedCodepage** constant that specifies a default code page. If barcode was created using code page that differs from the specification code page, that code page should be specified in this parameter. This parameter is ignored if the *allowedTypes* parameter set to more than one type.

layoutBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineLayout** variable that contains the recognition results. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

progressCallback

[in] The pointer to the **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

TFineBarcodeType

TFineBarcodeOrientation

TFineBarcodeSupplement

FineFreeMemory Function

This function releases memory allocated internally by the processing engine or by an explicit call to **FineAllocMemory**. If during library initialization you specified a custom function for memory release, that function will be used.

C Syntax

```
TFineErrorCode FineFreeMemory( void* ptr );
```

Parameters

ptr

[in] A pointer to memory which must be released.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineAllocMemory

FineInitialize

FineGetLastErrorMessage Function

This function returns the human-readable description of the last error that occurred in ABBYY Mobile OCR Engine library functions.

C Syntax

```
TFineErrorCode FineGetLastErrorMessage( const WCHAR** message );
```

Parameters

message

[out] A pointer to the string which receives the message.

Return value

This function returns `FEC_NotInitialized` if the ABBYY Mobile OCR Engine library has not been initialized. It can also return the other standard return values of ABBYY Mobile OCR Engine functions.

FineGetLicenseInfo Function

This function returns information about the current license.

Note: If you use a trial license, the word "ABBYY" will appear in each 20th line in the recognized text and in each third recognized business card.

C Syntax

```
TFineErrorCode FineGetLicenseInfo( WCHAR** licenseInfo );
```

Parameters

licenseInfo

[out] A string with license information. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

Return value

This function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineSetLicenseInfo

FineGetTextLines Function

This function detects text lines on the image.

C Syntax

```
TFineErrorCode FineGetTextLines(
    const CFineImage*          image,
    TFineImageProcessingOptions imageProcessingOptions,
    CFineRects**               linesBuff,
    TFineProgressCallbackFunction progressCallback,
);
```

Parameters

image

[in] The image to be recognized as a **CFineImage** variable.

imageProcessingOptions

[in] OR combination of the **TFineImageProcessingOptionsFlags** constants which define the image processing parameters.

linesBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineRects** variable which describes an array of rectangles. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

FineGetVersionInfo Function

This function returns the version of the library in following format: <major>.<minor>.<modification>.<build>. For example, 4.2.1.10.

C Syntax

```
void FineGetVersionInfo(
    int* major,
    int* minor,
    int* modification,
    int* build
);
```

Parameters

major

[out] A major version of the library.

minor

[out] A minor version of the library.

modification

[out] A modification version of the library.

build

[out] A build number of the library.

Return value

This function returns the standard return values of ABBYY Mobile OCR Engine functions.

FineGetWordSuggest Function

This function generates a list of suggestions for the selected word from the specified dictionary.

C Syntax

```
TFineErrorCode FineGetWordSuggest(
    const TFineDictionaryPtr dictionary,
    const WCHAR word[],
    int wordLength,
    int stringAssurance
    CFineWordSuggestion** suggestionBuff
);
```

Parameters

dictionary

[in] The address of the specified dictionary as a **TFineDictionaryPtr** variable.

word[]

[in] The word suggestion for which will be generated.

wordLength

[in] The number of characters in the word suggestion for which will be generated.

stringAssurance

[in] The average confidence of characters in the word. It must in the range from 0 to 100. The more this parameter, the less suggestions will be created.

suggestionBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineWordSuggestion** variable which represents an array of word suggestions. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

Return value

It returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

Working with Languages
Working with Dictionaries

FineInitialize Function

This function initializes the ABBYY Mobile OCR Engine library. It allows you to specify client-implemented functions for memory allocation and release and a logging function.

Important! All functions of the ABBYY Mobile OCR Engine library should be called only from the thread in which the library was initialized. You can not initialize the library in several threads simultaneously or parallel without deinitialization.

C Syntax

```
TFineErrorCode FineInitialize(
    TFineAllocMemoryFunction  allocFunction,
    TFineFreeMemoryFunction   freeFunction,
    TFineExecutionLogFunction  executionLogFunction
);
```

Parameters

allocFunction

[in] A custom function for memory allocation **TFineAllocMemoryFunction**. All memory used by the library will be allocated through this function. This parameter is optional. If this parameter is zero, memory will be allocated in standard way.

freeFunction

[in] A custom function for memory release **TFineFreeMemoryFunction**. This function is used to release memory which was allocated by the function specified in the *allocFunction* parameter. This parameter is optional. If the *allocFunction* parameter is zero, this parameter must be zero too.

executionLogFunction

[in] A custom function **FineExecutionLogFunction** which receives the logging information (errors, warnings and tips which occur during the execution). This parameter is optional. If this parameter is zero, the logging will be disabled.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineDeinitialize

Memory Management Functions
How to Use the Native Library

FineLoadImageFromFile Function

This function loads an image from a file.

C Syntax

```
TFineErrorCode FineLoadImageFromFile(
    CFineImageFile*          imageFile,
    TFineImageLoadingOptions  imageLoadingOptions,
    const RECT*              cropRect,
    CFineImage**             imageBuff
);
```

Parameters

imageFile

[in] A pointer to the **CFineImageFile** object for loading an image from a file.

imageLoadingOptions

[in] The OR combination of the **TFineImageLoadingOptionsFlags** constants which define the image loading parameters.

cropRect

[in] A pointer to a rectangle which defines the image crop. If it is zero, the image is loaded without crop.

imageBuff

[out] A pointer to pointer variable that receives the interface pointer to a **CFineImage** variable which stores the loaded image. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineFreeMemory

FineLoadImageFromInputStream Function

This function loads an image from the input stream.

C Syntax

```
TFineErrorCode FineLoadImageFromInputStream(
    CFineImageInputStream*  imageInputStream,
    TFineImageLoadingOptions imageLoadingOptions,
    const RECT*             cropRect,
    CFineImage**            imageBuff
);
```

Parameters

imageInputStream

[in] A pointer to the **CFineImageInputStream** object for loading data from the input stream.

imageLoadingOptions

[in] OR combination of the **TFineImageLoadingOptionsFlags** constants which define the image loading parameters.

cropRect

[in] A pointer to a rectangle to crop. If it is zero, the image is loaded without crop.

imageBuff

[out] A pointer to pointer variable that receives the interface pointer to a **CFineImage** variable which stores the loaded image. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineFreeMemory

FinePrebuildWordsInfo Function

This function returns the document layout information, including rectangles of words, without the text recognition. It does not support CJK languages. If any CJK language is included in the *languages* parameter, the function returns an error.

C Syntax

```
TFineErrorCode FinePrebuildWordsInfo(
    const TLanguageID      languages[],
    const TFinePatternsPtr  patterns,
    const CFineImage*       image,
    TFineImageProcessingOptions  imageProcessingOptions,
    CFinePrebuiltLayoutInfo** finePrebuiltLayoutInfo,
    TFineRotationType*      rotation,
    TFineProgressCallbackFunction  progressCallback
);
```

Parameters

languages[]

[in] The list of language IDs as an array of the **TLanguageID** constants terminated by LID_Undefined.

Note: It is better not to add to the list more than two recognition languages.

patterns

[in] The address of patterns as a **TFinePatternsPtr** variable.

image

[in] The image to be recognized as a **CFineImage** variable.

imageProcessingOptions

[in] OR combination of the **TFineImageProcessingOptionsFlags** constants that define the image processing parameters.

finePrebuiltLayoutInfo

[out] A pointer to the **CFinePrebuiltLayoutInfo** pointer with the output results. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

rotation

[out] The **TFineRotationType** constant that specifies the rotation angle of an input image before recognition if the FIPO_DetectPageOrientation flag is set in the *imageProcessingOptions* parameter, otherwise, it contains 0.

Note: The recognized text coordinates correspond to a rotated image.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

Working with Languages

Working with Dictionaries

FinePreprocessImage Function

This function binarizes an image and significantly reduces the size of the image. It can also perform skew correction, detect orientation etc., depending on the value of *imageProcessingOptions* parameter.

C Syntax

```
TFineErrorCode FinePreprocessImage(
    const CFineImage*       image,
    TFineImageProcessingOptions  imageProcessingOptions,
    CFineImage**            preprocessedImageBuff,
```

```
CFineImageTransformationInfo** transformationInfo,
TFineProgressCallbackFunction progressCallback
);
```

Parameters

image

[in] The image to be binarized as a **CFineImage** variable.

imageProcessingOptions

[in] OR combination of the **TFineImageProcessingOptionsFlags** constants which define the image processing parameters.

preprocessedImageBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineImage** variable which describes the resultant image. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

transformationInfo

[out] A pointer to pointer variable that receives the interface pointer of a **CFineImageTransformationInfo** variable which stores information about input image transformation. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineFreeMemory

FineRecognizeBarcode Function

This function recognizes an image of a barcode.

Consider also using the **FineExtractBarcodes** function, which can recognize more than one barcode on the image.

C Syntax

```
TFineErrorCode FineRecognizeBarcode(
    const CFineImage*          image,
    DWORD                     allowedTypes,
    DWORD                     allowedOrientations,
    DWORD                     allowedSupplements,
    int                       hasChecksum,
    int                       isCode39WithoutAsterisk,
    int                       isBinaryInterpretedAsText,
    TFineSupportedCodepage    defaultCodePage,
    WCHAR                     unknownLetter,
    CFineBarcode**            resultBuff,
    TFineProgressCallbackFunction progressCallback
);
```

Parameters

image

[in] The image to be recognized as a **CFineImage** variable.

allowedTypes

[in] The OR combination of the **TFineBarcodeType** constants that define acceptable barcode types.

allowedOrientations

[in] The OR combination of the **TFineBarcodeOrientation** constants that define the possible orientations of the barcode.

allowedSupplements

[in] The OR combination of the **TFineBarcodeSupplement** constants that define the possible supplements of the barcode. This parameter is ignored for barcodes without supplement. Set the parameter to FBS_Void if the barcode you recognize does not have a supplement.

hasChecksum

[in] Should not be zero if the barcode should be recognized as a barcode with checksum. It is valid for Code39, Interleaved25, Codabar, and Matrix25 barcodes. For these types of the barcodes, the last symbol of the barcode is considered as control sum of all barcode symbols, and is checked during the recognition.

isCode39WithoutAsterisk

[in] Should not be zero if the Code39 barcode has no start and stop symbol, the asterisk "*". It is valid for Code39 barcode. This parameter is ignored if the *allowedTypes* parameter set to more than one type.

isBinaryInterpretedAsText

[in] Should not be zero if byte data should be interpreted as text in the current code page. If this parameter is zero the data will be stored in hexadecimal format. This parameter is ignored if the *allowedTypes* parameter set to more than one type.

defaultCodePage

[in] A **TFineSupportedCodepage** constant that specifies a default code page. If barcode was created using code page that differs from the specification code page, that code page should be specified in this parameter. This parameter is ignored if the *allowedTypes* parameter set to more than one type.

unknownLetter

[in] A character that is written instead of unrecognized symbol or binary zero. Also it indicates unsuccessful recognition result.

resultBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineBarcode** variable that contains the recognition results. This pointer must be released afterwards with the help of the **FineFreeMemory** function. If the function fails to recognize the image as a barcode, this parameter contains a character specified in the *unknownLetter* parameter with FTCQ_Min quality and have FBT_Unrecognized type in the Type property.

progressCallback

[in] The pointer to the **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

TFineBarcodeType

TFineBarcodeOrientation

TFineBarcodeSupplement

FineRecognizeBusinessCard Function

This function recognizes all text lines on the image and analyzes the image as a business card in one step.

C Syntax

```
TFineErrorCode FineRecognizeBusinessCard(
    const TLanguageID          languages[],
    const TFinePatternsPtr      patterns,
    const TFinePatternsPtr*     cjkPatterns,
    const TFineDictionaryPtr    dictionaries[],
    const TFineKeywordsPtr      keywords[],
    const CFineImage*           image,
```

```

TFineImageProcessingOptions    imageProcessingOptions,
TFineRecognitionMode          recMode,
TFineRecognitionConfidenceLevel confidenceLevel,
CFineBusinessCard**          businessCardBuffer,
TFineRotationType*            rotation,
TFineProgressCallbackFunction progressCallback,
TFinePrebuiltDataCallbackFunction prebuiltDataCallback
);

```

Parameters

languages[]

[in] The list of language IDs as an array of the **TLanguageID** constants terminated by LID_Undefined. We do not recommend adding more than two recognition languages to the list.

For the best result of business card recognition, add the English language to the list of the recognition languages, regardless of the language of the business card.

patterns

[in] The address of patterns as a **TFinePatternsPtr** variable.

cjkPatterns

[in] The zero-terminated list of pointers to the patterns for CJK languages.

dictionaries[]

[in] The zero-terminated list of dictionaries as an array of the **TFineDictionaryPtr** variables.

keywords[]

[in] The zero-terminated list of keywords dictionaries as an array of the **TFineKeywordsPtr** variables. For the best result of business card recognition, add the English language keywords dictionary to the list, regardless of the language of the business card.

image

[in] The image to be recognized as a **CFineImage** variable.

imageProcessingOptions

[in] OR combination of the **TFineImageProcessingOptionsFlags** constants which define the image processing parameters.

recMode

[in] The **TFineRecognitionMode** constant which sets the recognition mode.

confidenceLevel

[in] The **TFineRecognitionConfidenceLevel** constant which sets the recognition confidence level.

businessCardBuffer

[out] A pointer to pointer variable that receives the interface pointer of a **CFineBusinessCard** variable which represents a business card. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

rotation

[out] The **TFineRotationType** constant which specifies the rotation angle of an input image before recognition if the FIPO_DetectPageOrientation flag is set in the *imageProcessingOptions* parameter, otherwise, it contains 0.

Note: The recognized text coordinates correspond to a rotated image.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

prebuiltDataCallback

[in] The **TFinePrebuiltDataCallbackFunction** callback function that delivers the information about the document layout, text blocks and lines before the text recognition. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

Recognizing Business Cards
Working with Languages
Working with Dictionaries

FineRecognizeImage Function

This function recognizes all text lines on the image.

C Syntax

```
TFineErrorCode FineRecognizeImage(
    const TLanguageID          languages[],
    const TFinePatternsPtr     patterns,
    const TFinePatternsPtr*    cjkPatterns,
    const TFineDictionaryPtr   dictionaries[],
    const CFineImage*          image,
    TFineImageProcessingOptions imageProcessingOptions,
    TFineRecognitionMode       recMode,
    TFineRecognitionConfidenceLevel confidenceLevel,
    CFineLayout**              layoutBuff,
    TFineRotationType*         rotation,
    TFineProgressCallbackFunction progressCallback,
    TFinePrebuiltDataCallbackFunction prebuiltDataCallback
);
```

Parameters

languages[]

[in] The list of language IDs as an array of the **TLanguageID** constants terminated by LID_Undefined. We do not recommend adding more than two recognition languages to the list.

patterns

[in] The address of patterns as a **TFinePatternsPtr** variable.

cjkPatterns

[in] The zero-terminated list of pointers to the patterns for CJK languages.

dictionaries[]

[in] The zero-terminated list of dictionaries as an array of **TFineDictionaryPtr** variables.

image

[in] The image to be recognized as a **CFineImage** variable.

imageProcessingOptions

[in] OR combination of the **TFineImageProcessingOptionsFlags** constants that define the image processing parameters.

recMode

[in] The **TFineRecognitionMode** constant that sets the recognition mode.

confidenceLevel

[in] The **TFineRecognitionConfidenceLevel** constant that sets the recognition confidence level.

layoutBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineLayout** variable that describes the recognized text. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

rotation

[out] The **TFineRotationType** constant that specifies the rotation angle of an input image before recognition if the FIPO_DetectPageOrientation flag is set in the *imageProcessingOptions* parameter, otherwise, it contains 0.

Note: The recognized text coordinates correspond to a rotated image.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

prebuiltDataCallback

[in] The **TFinePrebuiltDataCallbackFunction** callback function that delivers the information about the document layout, text blocks and lines before the text recognition. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

Working with Languages

Working with Dictionaries

FineRecognizeRegion Function

This function recognizes all text lines in the specified region. A region is an array of bounding rectangles. Each rectangle represents one text line. This function is useful when you need to recognize a specific business card field with the help of a custom dictionary.

C Syntax

```
TFineErrorCode FineRecognizeRegion(
    const TLanguageID          languages[],
    const TFinePatternsPtr     patterns,
    const TFinePatternsPtr*    cjkPatterns,
    const TFineDictionaryPtr   dictionaries[],
    const CFineImage*          image,
    int                        regionLength,
    const RECT*                regionRects,
    TFineImageProcessingOptions imageProcessingOptions,
    TFineRecognitionMode       recMode,
    TFineRecognitionConfidenceLevel confidenceLevel,
    CFineLayout**              layoutBuff,
    TFineRotationType*         rotation,
    TFineProgressCallbackFunction progressCallback,
    TFinePrebuiltDataCallbackFunction prebuiltDataCallback
);
```

Parameters

languages[]

[in] The list of language IDs as an array of the **TLanguageID** constants terminated by LID_Undefined. We do not recommend adding more than two recognition languages to the list.

patterns

[in] The address of patterns as a **TFinePatternsPtr** variable.

cjkPatterns

[in] The zero-terminated list of pointers to the patterns for CJK languages.

dictionaries[]

[in] The zero-terminated list of dictionaries as an array of the **TFineDictionaryPtr** variables.

image

[in] The image to be recognized as a **CFineImage** variable.

regionLength

[in] The number of rectangles of the specified region.

regionRects

[in] The array of rectangles of the specified region.

imageProcessingOptions

[in] OR combination of the **TFineImageProcessingOptionsFlags** constants which define the image processing parameters.

recMode

[in] The **TFineRecognitionMode** constant which sets the recognition mode.

confidenceLevel

[in] The **TFineRecognitionConfidenceLevel** constant which sets the recognition confidence level.

layoutBuff

[out] A pointer to pointer variable that receives the interface pointer of a **CFineLayout** variable which describes the recognized text. This pointer must be released afterwards with the help of the **FineFreeMemory** function.

rotation

[out] The **TFineRotationType** constant which specifies the rotation angle of an input image before recognition if the FIPO_DetectPageOrientation flag is set in the *imageProcessingOptions* parameter, otherwise, it contains 0.

Note: The recognized text coordinates correspond to a rotated image.

progressCallback

[in] The **TFineProgressCallbackFunction** callback function that delivers the progress information. It can be 0.

prebuiltDataCallback

[in] The **TFinePrebuiltDataCallbackFunction** callback function that delivers the information about the document layout, text blocks and lines before the text recognition. It can be 0.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

Working with Languages

Working with Dictionaries

FineSetLicenseInfo Function

This function sets the license information. The ABBYY Mobile OCR Engine license file has to be loaded to the memory and set with the help of this function.

Note: If you use a trial license, the word "ABBYY" will appear in each 20th line in the recognized text and in each third recognized business card.

C Syntax

```
TFineErrorCode FineSetLicenseInfo( const CFineLicenseInfo* licenseInfo );
```

Parameters

licenseInfo

[in] A constant pointer to the **CFineLicenseInfo** variable containing the license information.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineGetLicenseInfo

How to Use the Native Library

FineSetRecognizerThreadsCount Function

This function is used to limit the number of threads that can be started simultaneously for multi-threaded processing with ABBYY Mobile OCR Engine.

By default, all recognition operations are performed in parallel, using up to 4 threads. Call this function after the library is initialized if you need to change this limitation.

Note: For multi-threaded processing the Pthreads-win32 library is used. This library is included in the distribution as a separate dynamic link library (pthreadVC2.dll).

C Syntax

```
TFineErrorCode FineSetRecognizerThreadsCount( int threadsCount );
```

Parameters

threadsCount

[in] The maximum number of threads that can be run at once.

Return value

The function returns the standard return values of ABBYY Mobile OCR Engine functions.

See also

FineInitialize

How to Use the Native Library

Callback Functions

This section contains:

- [TFinePrebuiltDataCallbackFunction](#)
- [TFineProgressCallbackFunction](#)

TFinePrebuiltDataCallbackFunction

A pointer to the callback function that should be implemented on the client side. It delivers to the client the prebuilt information about the document layout, text blocks and lines before the text recognition.

C Syntax

```
typedef void (*TFinePrebuiltDataCallbackFunction) (
    TFinePrebuiltDataType dataType,
    void* data
);
```

Parameters

dataType

[in] A **TFinePrebuiltDataType** constant that specifies the pointer type to which the *data* parameter should be cast.

data

[in] A pointer to a structure with prebuilt data of the type that is specified in the *dataType* parameter.

Return value

This function returns zero to break recognition process.

See also

FineAnalyzeImage

FineRecognizeBlocks

FineRecognizeImage

FineRecognizeRegion

FineRecognizeBusinessCard

TFineProgressCallbackFunction

This is a callback function that should be implemented on the client side. It delivers to the client the information about the approximate percentage of analysis or recognition and warnings or errors that have occurred during processing.

C Syntax

```
int (*TFineProgressCallbackFunction) (
    int    processedPercentage,
    DWORD  warning,
    void*   warningData
);
```

Parameters

processedPercentage

[in] The percentage of the current work which has already been done. It is in the range from 0 to 100.

warning

[in] The constant of the **TFineWarningCode** enumeration which describes the warning which has occurred during processing.

warningData

[in] A pointer to data structure with the details of the warning. Its contents depend on the type of the warning. See the description of the **TFineWarningCode** constants for details.

Return value

If this function returns zero, the recognition is cancelled.

Sample

Here is a sample implementation of the callback function:

```
int TFineProgressCallbackFunction( int processedPercentage, DWORD warning, void*
warningData )
{
    fprintf( TraceFile, "%d%% of the work is done.\n", processedPercentage );
    if( warning == FWC_ProbablyBadImage) {
        fprintf( TraceFile, "The image quality is too low.\n" );
    }
    if( processedPercentage < 50 && ( warning == FWC_SlowRecognition ) != 0 ) {
        return 0;
    } else {
        return 1;
    }
}
```

See also

FineAnalyzeImage

FinePrebuildWordsInfo

FineRecognizeBlocks

FineRecognizeImage

FineRecognizeRegion

FineRecognizeBusinessCard

Custom Memory Management Functions

ABBYY Mobile OCR Engine provides the **FineAllocMemory** and **FineFreeMemory** functions to deal with memory management. However, if you need to use special allocation-deallocation algorithms, you can implement your own functions:

- `TFineAllocMemoryFunction`
- `TFineFreeMemoryFunction`

These pointers to functions allow you to use your own algorithms for memory allocation and release. When initializing the library, pass the pointers to the functions you implemented as input parameters to the **FineInitialize** function, and the internal memory operations will be executed using these functions.

See also

FineInitialize

TFineAllocMemoryFunction Function

This function is implemented on the client side and allows you to use your preferred memory management algorithm. It is used to allocate memory.

C Syntax

```
void* (*TFineAllocMemoryFunction)( int size );
```

Parameters

size

[in] The size of memory buffer that needs to be allocated.

Return value

The function returns the address to the allocated memory buffer.

See also

TFineFreeMemoryFunction

FineAllocMemory

TFineFreeMemoryFunction Function

This function is implemented on the client side and allows you to use your preferred memory management algorithm. It releases memory allocated by the call to **TFineAllocMemoryFunction**.

C Syntax

```
void (*TFineFreeMemoryFunction)( void* ptr );
```

Parameters

ptr

[in] A pointer to memory which must be released.

Return value

The function does not have a return value.

See also

TFineAllocMemoryFunction

FineFreeMemory

Structures

This section contains the description of ABBYY Mobile OCR Engine structures:

- CFineAngle
- CFineBarcode
- CFineBcrField
- CFineBcrFieldComponent
- CFineBusinessCard
- CFineImage
- CFineImageFile
- CFineImageInputStream
- CFineImageTransformationInfo
- CFineLayout
- CFineLicenseInfo
- CFinePrebuiltLayoutInfo
- CFinePrebuiltTextBlockInfo
- CFinePrebuiltTextLineInfo
- CFineRects
- CFineTextBlock
- CFineTextCharacter
- CFineTextLine
- CFineWarningDataWrongLanguages
- CFineWordInfo
- CFineWordSuggestion
- CFineWordVariant

CFineAngle Structure

This structure describes the value of the angle tangent.

C Syntax

```
typedef struct tagCFineAngle{
    int    Numerator;
    int    Denominator;
} CFineAngle;
```

Fields

Name	Type	Description
Denominator	int	Stores the denominator of the angle tangent.
Numerator	int	Stores the numerator of the angle tangent.

See also

CFineImageTransformationInfo

CFineBarcode Structure

This structure describes the result of the barcode recognition.

C Syntax

```
typedef struct tagCFineBarcode {
    TFineBarcodeType Type;
    CFineTextLine TextLine;
} CFineBarcode;
```

Fields

Name	Type	Description
TextLine	CFineTextLine	Stores the recognized text of the barcode.
Type	TFineBarcodeType	Stores the barcode type.

Output parameter

This structure is the output parameter of the **FineRecognizeBarcode** function.

See also

CFineLayout

CFineBcrField Structure

This structure represents a business card field.

C Syntax

```
typedef struct tagCFineBcrField {
    TBcrFieldType Type;
    int TextLinesCount;
    CFineTextLine* TextLines;
    int ComponentsCount;
    CFineBcrFieldComponent* Components;
} CFineBcrField;
```

Fields

Name	Type	Description
Components	CFineBcrFieldComponent*	Stores the field components. The field components are stored in the order of appearance on the business card.
ComponentsCount	int	Stores the number of field components. It is zero if the field is not divided.
TextLines	CFineTextLine*	Stores recognized text in the card field. It may contain several lines.
TextLinesCount	int	Stores the number of text lines in the card field.
Type	TBcrFieldType	Stores the type of a business card field.

See also

CFineBusinessCard

CFineBusinessCard Structure

This structure represents a business card.

C Syntax

```
typedef struct tagCFineBusinessCard {
    CFineBcrField* Fields;
    int FieldsCount;
} CFineBusinessCard;
```

Fields

Name	Type	Description
Fields	CFineBcrField*	Stores the array of business card fields.
FieldsCount	int	Stores the number of fields.

Output parameter

This structure is the output parameter of the **FineAnalyzeTextAsBusinessCard**, **FineRecognizeBusinessCard** functions.

CFineBusinessCard Structure

This structure represents a business card.

C Syntax

```
typedef struct tagCFineBusinessCard {
    CFineBcrField* Fields;
    int FieldsCount;
} CFineBusinessCard;
```

Fields

Name	Type	Description
Fields	CFineBcrField*	Stores the array of business card fields.
FieldsCount	int	Stores the number of fields.

Output parameter

This structure is the output parameter of the **FineAnalyzeTextAsBusinessCard**, **FineRecognizeBusinessCard** functions.

CFineImage Structure

This structure represents the image. ABBYY Mobile OCR Engine supports the following formats:

- *black and white* – 1 bit-per-pixel (bpp) image, where 0 is black and 1 is white;
- *grey* – 8 bpp image, where 0 is black and 255 is white;
- *color* – 24 bpp image. The byte order is BGR, therefore (0,0,0) is black and (255,255,255) is white.

C Syntax

```
typedef struct tagCFineImage {
    int ImageWidth;
    int ImageHeight;
    int ImageByteWidth;
    int BitsPerPixel;
    int Resolution;
    BYTE* Image;
} CFineImage;
```

Fields

Name	Type	Description
BitsPerPixel	int	Stores the number of bits used for one pixel. It should be 1 for black and white image, 8 for gray, 24 for color.
Image	BYTE*	Stores the image bitmap line-by-line, from top to bottom. Each line starts at the BYTE boundary.
ImageByteWidth	int	Stores the number of bytes occupied by each raster line. The value of this field should be at least the smallest integer greater than or equal to $(\text{ImageWidth} * \text{BitsPerPixel}/8)$.
ImageHeight	int	Stores the height of the image in pixels.
ImageWidth	int	Stores the width of the image in pixels.
Resolution	int	Stores the horizontal and vertical resolution in dpi.

Input parameter

This structure is passed as an input parameter to the **FineAnalyzeImage**, **FineGetTextLines**, **FinePrebuildWordsInfo**, **FinePreprocessImage**, **FineRecognizeBarcode**, **FineRecognizeBlocks**, **FineRecognizeBusinessCard**, **FineRecognizeImage**, **FineRecognizeRegion** functions.

Output parameter

This structure is the output parameter of the **FineLoadImageFromFile**, **FineLoadImageFromInputStream** function.

CFineImageFile Structure

This structure contains methods for loading an image from a file.

C Syntax

```
typedef struct tagCFineImageFile {
    int (*Read)( void* file, BYTE* buffer, int size );
    int (*Seek)( void* file, int offset, TFineImageFileSeekPosition from );
    int (*GetLength)( void* file );
} CFineImageFile;
```

Methods

Name	Description
GetLength	Returns the file size in bytes.
Read	Reads an image from a file.
Seek	Sets the position indicator associated with the file to a new position.

Input parameter

This structure is passed as an input parameter to the **FineLoadImageFromFile** function.

GetLength Method of CFineImageFile Structure

This method returns the size of the image file in bytes.

C Syntax

```
int (*GetLength)( void* file );
```

Parameters

file

A pointer to the **CFineImageFile** object.

Return value

It returns the file size in bytes.

See also

CFineImageFile

Read Method of CFineImageFile Structure

This method loads an image from a file into memory.

C Syntax

```
int (*Read) (
    void* file,
    BYTE* buffer,
    int size
);
```

Parameters

file

A pointer to the **CFineImageFile** object.

buffer

A pointer to the area of memory which receives the data from the file.

size

The size of the data that will be stored in the *buffer* parameter.

Return value

It returns the number of the loaded bytes. The return value can be less than the *size* parameter. If the value is less or equal to zero, no data have been loaded.

See also

CFineImageFile

Seek Method of CFineImageFile Structure

This method sets the position indicator associated with the file to a new position.

C Syntax

```
int (*Seek) (
    void* file,
    int offset,
    TFineImageFileSeekPosition from
);
```

Parameters

file

A pointer to the **CFineImageFile** object.

offset

A value to which the position indicator should be set or incremented.

from

A constant from the **TFineImageFileSeekPosition** enumeration that specifies whether the position indicator should be set to or incremented by the offset parameter.

Return value

It returns zero if a new position is set successfully; otherwise, non-zero value that is interpreted as an error.

See also

CFineImageFile

CFineImageInputStream Structure

This structure contains methods for loading an image from the input stream.

C Syntax

```
typedef struct tagCFineImageInputStream {
    int (*Read)( void* inputStream, BYTE* buffer, int size );
    int (*Skip)( void* inputStream, int size );
} CFineImageInputStream;
```

Methods

Name	Description
Read	Reads an image from the input stream.
Skip	Skips the specified number of bytes from the input stream.

Input parameter

This structure is passed as an input parameter to the **FineLoadImageFromInputStream** function.

Read Method of CFineImageInputStream Structure

This method loads an image from the input stream.

C Syntax

```
int (*Read)(
    void* inputStream,
    BYTE* buffer,
    int size
);
```

Parameters

inputStream

A pointer to the **CFineImageInputStream** object.

buffer

Data from the input stream is written to this parameter.

size

The size of the loading data that are stored in the *buffer* parameter.

Return value

It returns the number of the loaded bytes. The return value can be less than the *size* parameter. If the value is less or equal to zero, no data have been loaded.

See also

CFineImageInputStream

Skip Method of CFineImageInputStream Structure

This method skips the specified number of bytes from the input stream.

Note: A pointer to the **Skip** function can be zero, in this case, the **Read** function is used.

C Syntax

```
int (*Skip)(
    void* inputStream,
```

```
int size
);
```

Parameters

inputStream

A pointer to the **CFineImageInputStream** object.

size

The size of the skipped data in bytes.

Return value

It returns the number of the skipped bytes that should be equal to the *size* parameter. If the value does not equal to the *size* parameter, it is interpreted as a reading error.

See also

CFineImageInputStream

CFineImageTransformationInfo Structure

This structure stores information about the image transformation.

C Syntax

```
typedef struct tagCFineImageTransformationInfo {
    CFineAngle SkewAngle;
} CFineImageTransformationInfo;
```

Fields

Name	Type	Description
SkewAngle	CFineAngle	Stores the value of the skew angle tangent.

Output parameter

This structure is the output parameter of the **FinePreprocessImage** function.

CFineLayout Structure

This structure describes the recognition result of an image. It can contain a set of text blocks with the recognized text and a set of barcode blocks with the recognized barcodes.

C Syntax

```
typedef struct tagCFineLayout {
    CFineTextBlock* TextBlocks;
    int TextBlocksCount;
    CFineBarcode* BarcodeBlocks;
    int BarcodeBlocksCount;
} CFineLayout;
```

Fields

Name	Type	Description
BarcodeBlocks	CFineBarcode*	The pointer to the array of barcode blocks found on the image.
BarcodeBlocksCount	int	The number of barcode blocks.
TextBlocks	CFineTextBlock*	The pointer to the array of text blocks found on the image.
TextBlocksCount	int	The number of text blocks.

Input parameter

This structure is passed as an input parameter to the **FineAnalyzeTextAsBusinessCard** function.

Output parameter

This structure is the output parameter of the **FineAnalyzeImage**, **FineExtractBarcodes**, **FineRecognizeImage**, **FineRecognizeRegion** functions.

CFineLicenseInfo Structure

This structure defines license information.

C Syntax

```
typedef struct tagCFineLicenseInfo {
    BYTE*      LicenseData;
    DWORD      DataLength;
    const WCHAR* ApplicationId;
} CFineLicenseInfo;
```

Fields

Name	Type	Description
ApplicationId	WCHAR*	Stores a string with application identification. Important! The ApplicationId must correspond to the application ID in the license file. If you do not know your application ID, contact your sales manager.
DataLength	DWORD	Stores the length of the data loaded from the license file.
LicenseData	BYTE*	Stores a pointer to the memory buffer that contains data loaded from the license file.

Input parameter

This structure is passed as an input parameter to the **FineSetLicenseInfo** function.

CFinePrebuiltLayoutInfo Structure

This structure stores information about the document layout. The information is available before the text recognition.

C Syntax

```
typedef struct tagCFinePrebuiltLayoutInfo {
    CFinePrebuiltTextBlockInfo* TextBlocks;
    int TextBlocksCount;
} CFinePrebuiltLayoutInfo;
```

Fields

Name	Type	Description
TextBlocks	CFinePrebuiltTextBlockInfo*	Stores a pointer to an array of the CFinePrebuiltTextBlockInfo objects with information about the text blocks.
TextBlocksCount	int	Stores the number of elements in the TextBlocks field.

Input parameter

This structure is passed as an input parameter to the **TFinePrebuiltDataCallbackFunction** if the *dataType* argument is set to **FPDT_WordsInfo**.

Output parameter

This structure is the output parameter of the **FinePrebuildWordsInfo** function.

CFinePrebuiltTextBlockInfo Structure

This structure stores information about the text block. The information is available before the text recognition.

C Syntax

```
typedef struct tagCFinePrebuiltTextBlockInfo {
    RECT*           RegionRects;
    int             RegionRectsCount;
    CFinePrebuiltTextLineInfo* Lines;
    int             LinesCount;
} CFinePrebuiltTextBlockInfo;
```

Fields

Name	Type	Description
Lines	CFinePrebuiltTextLineInfo*	Stores a pointer to an array of the CFinePrebuiltTextLineInfo objects with information about the text lines in the text block.
LinesCount	int	Stores the number of elements in the Lines field.
RegionRects	RECT*	Stores a pointer to an array of the rectangles that describe an image region with the text block.
RegionRectsCount	int	Stores the number of elements in the RegionRects field.

See also

CFinePrebuiltLayoutInfo

CFinePrebuiltTextLineInfo Structure

This structure stores information about the text line. The information is available before the text recognition.

C Syntax

```
typedef struct tagCFinePrebuiltTextLineInfo {
    RECT Rect;
    RECT* WordsRects;
    int WordsRectsCount;
} CFinePrebuiltTextLineInfo;
```

Fields

Name	Type	Description
Rect	RECT	Stores the bounding rectangle of the line.
WordsRects	RECT*	Stores a pointer to an array of the bounding rectangles of words in the line.
WordsRectsCount	int	Stores the number of elements in the WordsRects field.

See also

CFinePrebuiltTextBlockInfo

CFineRects Structure

This structure describes an array of rectangles.

C Syntax

```
typedef struct tagCFineRects {
    RECT* Rects;
```

```
int    RectsCount;
} CFineRects;
```

Fields

Name	Type	Description
Rects	RECT*	Stores the pointer to an array of rectangles.
RectsCount	int	Stores the number of rectangles in array.

Output parameter

This structure is the output parameter of the **FineGetTextLines** function.

CFineTextBlock Structure

This structure describes a block of the recognized text.

C Syntax

```
typedef struct tagCFineTextBlock {
    CFineTextLine*  Lines;
    int             LinesCount;
    RECT*           RegionRects;
    int             RegionRectsCount;
    DWORD           Attributes
} CFineTextBlock;
```

Fields

Name	Type	Description
Attributes	DWORD	Stores the attributes of the text block as the OR combination of the TFineTextBlockAttributes constants.
Lines	CFineTextLine*	Stores the pointer to an array of text lines.
LinesCount	int	Stores the number of text lines.
RegionRects	RECT*	Stores the pointer to an array of rectangles which describes the region of text 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).
RegionRectsCount	int	Stores the number of region rectangles.

Input parameter

An array of objects of this type is passed as an input parameter to the **FineRecognizeBlocks** function.

See also

CFineLayout

CFineTextCharacter Structure

This structure describes a character in the recognized text.

C Syntax

```
typedef struct tagCFineTextCharacter {
    WCHAR Unicode;
    WORD  SmallLetterHeight;
    RECT  Rect;
    DWORD Attributes;
    BYTE  Quality;
} CFineTextCharacter;
```

Fields

Name	Type	Description
Attributes	DWORD	The OR combination of the TFineCharacterAttributes constants which specifies the attributes detected for this character.
Rect	RECT	Stores the bounding rectangle of a character or ligature.
SmallLetterHeight	WORD	Stores the height of a small letter for the detected font.
Quality	BYTE	Stores the character recognition quality. The value of this field must be in the range from FTCQ_Min to FTCQ_Max.
Unicode	WCHAR	Stores the character code in the Unicode standard.

CFineTextLine Structure

This structure describes a line of the recognized text.

C Syntax

```
typedef struct tagCFineTextLine {
    CFineTextCharacter* Chars;
    int CharCount;
    CFineWordInfo* Words;
    int WordsCount;
    RECT Rect;
    int BaseLine;
} CFineTextLine;
```

Fields

Name	Type	Description
BaseLine	int	Stores the coordinate of the base line.
Chars	CFineTextCharacter*	Stores a pointer to characters buffer.
CharCount	int	Stores the number of characters in the line.
Rect	RECT	Stores the bounding rectangle of the line. Note: For barcode recognition this field is empty.
Words	CFineWordInfo*	Stores a pointer to an array of the CFineWordInfo objects with the word information structures. This field is available if the FIPO_BuildWordsInfo flag is passed to the recognition function.
WordsCount	int	Stores the number of elements in the Words field.

See also

CFineBarcode

CFineWordInfo

CFineTextCharacter

CFineWarningDataWrongLanguages Structure

This structure defines an array of recommended recognition languages. A pointer to this structure will be stored in the *warningData* parameter of the **TFineProgressCallbackFunction** when the warning which occurred is FWC_ProbablyWrongLanguages.

C Syntax

```
typedef struct tagCFineWarningDataWrongLanguages {
    TLanguageID* RecommendedLanguages;
    int RecommendedLanguagesCount;
} CFineWarningDataWrongLanguages;
```

Fields

Name	Type	Description
RecommendedLanguages	TLanguageID*	Stores a pointer to an array of recommended recognition language IDs.
RecommendedLanguagesCount	int	Stores the number of recommended languages.

Output parameter

This structure is the output parameter of the **TFineProgressCallbackFunction** when the warning which occurred is **FWC_ProbablyWrongLanguages**.

CFineWordInfo Structure

This structure represents information related to a part of text after splitting the text into words.

C Syntax

```
typedef struct tagCFineWordInfo {
    CFineWordVariant* Variants;
    int VariantsCount;
    DWORD Attributes;
    RECT Rect;
    DWORD SmallLetterHeight;
} CFineWordInfo;
```

Fields

Name	Type	Description
Attributes	DWORD	The OR combination of the TFineWordAttributes constants which specifies to which word model this part of text conforms.
Rect	RECT	The bounding rectangle of the part of the text. Note: If the Attributes field has the FWA_HyphenatedWord flag, this field should be ignored.
SmallLetterHeight	DWORD	The medium height of small letters of the word.
Variants	CFineWordVariant*	An array of the word and its derivatives. This array contains at least one word that is the recognized word as is.
VariantsCount	int	The number of elements in the Variants field.

See also

CFineTextLine

CFineWordVariant

CFineWordSuggestion Structure

This structure defines an array of word suggestions.

C Syntax

```
typedef struct tagCFineWordSuggestion {
    WCHAR** Words;
    int WordsCount;
} CFineWordSuggestion;
```

Fields

Name	Type	Description
Words	WCHAR**	Stores a pointer to an array of words. A word is Unicode and zero-terminated.

WordsCount	int	Stores the number of words.
-------------------	------------	-----------------------------

Output parameter

This structure is the output parameter of the **FineGetWordSuggest** function.

CFineWordVariant Structure

This structure defines the recognized word or its derivatives (primary form and corrections of the word).

C Syntax

```
typedef struct tagCFineWordVariant {
    WCHAR*          Chars;
    int             CharCount;
    TLanguageID*    WordLanguages;
    int             WordLanguagesCount;
    TFineWordVariantType Type;
} CFineWordVariant;
```

Fields

Name	Type	Description
Chars	WCHAR*	The string containing the word variant.
CharCount	int	The length of the word variant.
Type	TFineWordVariantType	The type of the word variant.
WordLanguages	TLanguageID*	The languages to which the word belongs.
WordLanguagesCount	int	The number of elements in the WordLanguages field.

See also

CFineWordInfo

Enumerations

This section contains:

- BIT_FLAG Macros
- TBcrComponentType
- TBcrFieldType
- TFineBarcodeOrientation
- TFineBarcodeSupplement
- TFineBarcodeType
- TFineCharacterAttributes
- TFineErrorCode
- TFineImageFileSeekPosition
- TFineImageLoadingOptionsFlags
- TFineImageProcessingOptionsFlags
- TFinePrebuiltDataType

- `TFineRecognitionConfidenceLevel`
- `TFineRecognitionMode`
- `TFineRotationType`
- `TFineSupportedCodepage`
- `TFineTextBlockAttributes`
- `TFineTextCharacterQuality`
- `TFineWarningCode`
- `TFineWordAttributes`
- `TFineWordVariantType`
- `TLanguageID`

BIT_FLAG Macros

This macros returns a number with one non-zero bit in the `n` position.

C Syntax

```
BIT_FLAG( n ) ( 1 << ( n ) )
```

See also

TFineCharacterAttributes

TFineWarningCode

TFineImageProcessingOptionsFlags

TFineCharacterAttributes

TFineCharacterAttributes enumeration constants are used as the mask in the **CFineTextCharacter** structure.

The mask is an OR combination of these flags' values which define character attributes. These constants are defined using the `BIT_FLAG` macros.

```
typedef enum tagTFineCharacterAttributes {
    FCA_Italic          = BIT_FLAG( 0 ),
    FCA_Bold            = BIT_FLAG( 1 ),
    FCA_Underlined      = BIT_FLAG( 2 ),
    FCA_Strikethrough   = BIT_FLAG( 3 ),
    FCA_Smallcaps       = BIT_FLAG( 4 ),
    FCA_Superscript     = BIT_FLAG( 5 ),
    FCA_Uncertain       = BIT_FLAG( 16 ),
    FCA_BarcodeBinaryDataHexed = BIT_FLAG( 17 ),
    FCA_BarcodeBinaryZero   = BIT_FLAG( 18 ),
    FCA_BarcodeStartStopSymbol = BIT_FLAG( 19 ),
} TFineCharacterAttributes;
```

Elements

Name	Description
<code>FCA_Italic</code>	Specifies whether the character is italic.
<code>FCA_Bold</code>	Specifies whether the character is bold.
<code>FCA_Underlined</code>	Specifies whether the character is underlined.
<code>FCA_Strikethrough</code>	Specifies whether the character is strikeout.

FCA_Smallcaps	Specifies whether the character has the "small caps" style. This means that the small characters are displayed as small capitals.
FCA_Superscript	Specifies whether the character is superscript.
FCA_Uncertain	Specifies whether the character has been recognized uncertainly. The confidence level at which characters are marked as uncertain must be set during recognition as a TFineRecognitionConfidenceLevel constant.
FCA_BarcodeBinaryDataHexed	Specifies a binary symbol that is written in hexadecimal format.
FCA_BarcodeBinaryZero	Specifies a zero binary symbol replaced by character that is specified in the FineRecognizeBarcode function as <i>unknownLetter</i> for correct representation.
FCA_BarcodeStartStopSymbol	Specifies the start and stop symbols. This flag is valid for Code39 and Codabar barcodes.

See also

CFineTextCharacter

TBcrFieldType

TBcrFieldType enumeration constants are used to describe different types of business card fields.

```
typedef enum tagTBcrFieldType {
    BFT_Phone,
    BFT_Fax,
    BFT_Mobile,
    BFT_Email,
    BFT_Web,
    BFT_Address,
    BFT_Name,
    BFT_Company,
    BFT_Job,
    BFT_Text,
    BFT_Count
} TBcrFieldType;
```

Elements

Name	Description
BFT_Phone	A phone number.
BFT_Fax	A fax number.
BFT_Mobile	A cell phone number.
BFT_Email	An e-mail address.
BFT_Web	A web address.
BFT_Address	A post address.
BFT_Name	A full name.
BFT_Company	A company name.
BFT_Job	A job title.
BFT_Text	The recognized text.
BFT_Count	The auxiliary constant which stores the number of constants in the enumeration.

See also

CFineBcrField

TFineBarcodeOrientation

TFineBarcodeOrientation enumeration constants are used to set the barcode orientation in the **FineRecognizeBarcode** function. The constants are defined using the BIT_FLAG macros or as a combination.

```
typedef enum tagTFineBarcodeOrientation {
    FBO_LeftToRight = BIT_FLAG( 0 ),
    FBO_DownToTop    = BIT_FLAG( 1 ),
    FBO_RightToLeft  = BIT_FLAG( 2 ),
    FBO_TopToDown    = BIT_FLAG( 3 ),
    FBO_AutoDetect   = FBO_LeftToRight | FBO_DownToTop | FBO_RightToLeft | FBO_TopToDown
} TFineBarcodeOrientation;
```

Flag

Name	Description
FBO_LeftToRight	Barcode is oriented from left to right.
FBO_DownToTop	Barcode is oriented from down to top.
FBO_RightToLeft	Barcode is oriented from right to left.
FBO_TopToDown	Barcode is oriented from top to down.
FBO_AutoDetect	The barcode orientation will be detected automatically.

See also

FineRecognizeBarcode

TFineBarcodeSupplement

TFineBarcodeSupplement enumeration constants are used to set the barcode supplement in the **FineRecognizeBarcode** function. The constants are defined using the BIT_FLAG macros or as a combination.

```
typedef enum tagTFineBarcodeSupplement {
    FBS_Void    = BIT_FLAG( 0 ),
    FBS_2Digit  = BIT_FLAG( 1 ),
    FBS_5Digit  = BIT_FLAG( 2 ),
    FBS_AutoDetect = FBS_Void | FBS_2Digit | FBS_5Digit,
    FBS_AnySupplement = FBS_2Digit | FBS_5Digit
} TFineBarcodeSupplement;
```

Flag

Name	Description
FBS_Void	The empty supplement.
FBS_2Digit	The 2-digit supplement.
FBS_5Digit	The 5-digit supplement.
FBS_AutoDetect	Forces ABBYY Mobile OCR Engine to automatically detect the barcode type during recognition.
FBS_AnySupplement	Combination of all non-empty supplements.

See also

FineRecognizeBarcode

TFineBarcodeType

TFineBarcodeType enumeration constants are used to set the barcode type in the **FineRecognizeBarcode** function. The constants are defined using the **BIT_FLAG** macros or as a combination.

```
typedef enum tagTFineBarcodeType {
    FBT_Unrecognized = 0,
    FBT_Code39 = BIT_FLAG( 0 ),
    FBT_Interleaved25 = BIT_FLAG( 1 ),
    FBT_Ean13 = BIT_FLAG( 2 ),
    FBT_Code128 = BIT_FLAG( 3 ),
    FBT_Ean8 = BIT_FLAG( 4 ),
    FBT_Pdf417 = BIT_FLAG( 5 ),
    FBT_Codabar = BIT_FLAG( 6 ),
    FBT_Upce = BIT_FLAG( 7 ),
    FBT_Industrial25 = BIT_FLAG( 8 ),
    FBT_Iata25 = BIT_FLAG( 9 ),
    FBT_Matrix25 = BIT_FLAG( 10 ),
    FBT_Code93 = BIT_FLAG( 11 ),
    FBT_Postnet = BIT_FLAG( 12 ),
    FBT_Ucc128 = BIT_FLAG( 13 ),
    FBT_Patch = BIT_FLAG( 14 ),
    FBT_Aztec = BIT_FLAG( 15 ),
    FBT_Datamatrix = BIT_FLAG( 16 ),
    FBT_Qrcode = BIT_FLAG( 17 ),
    FBT_Upca = BIT_FLAG( 18 ),
    FBT_Maxicode = BIT_FLAG( 19 ),
    FBT_Any1D = FBT_Code39 | FBT_Interleaved25 |
        FBT_Ean13 | FBT_Code128 | FBT_Ean8 | FBT_Codabar |
        FBT_Upce | FBT_Industrial25 | FBT_Iata25 |
        FBT_Matrix25 | FBT_Code93 | FBT_Ucc128 |
        FBT_Patch | FBT_Postnet | FBT_Upca,
    FBT_Square2D = FBT_Aztec | FBT_Datamatrix | FBT_Qrcode | FBT_Maxicode,
    FBT_Any1DWithSupplement = FBT_Ean13 | FBT_Ean8 | FBT_Upce | FBT_Upca
} TFineBarcodeType;
```

Flag

Name	Description
FBT_Unrecognized	Denotes unrecognized type of barcode. It is used as the return value if ABBYY Mobile OCR Engine has failed to detect the type of barcode.
FBT_Code39	Barcode in Code 39 standard.
FBT_Interleaved25	Barcode in Interleaved 2 of 5 standard.
FBT_Ean13	Barcode in EAN-13 standard.
FBT_Code128	Barcode in Code 128 standard.
FBT_Ean8	Barcode in EAN-8 standard.
FBT_Pdf417	Barcode in PDF417 standard.
FBT_Codabar	Barcode in Codabar standard.
FBT_Upce	Barcode in UPC-E standard.
FBT_Industrial25	Barcode in Industrial 2 of 5 standard.
FBT_Iata25	Barcode in IATA 2 of 5 standard.
FBT_Matrix25	Barcode in Matrix 2 of 5 standard.
FBT_Code93	Barcode in Code 93 standard.

FBT_Postnet	Barcode in Postnet standard.
FBT_Ucc128	Barcode in GS1-128 standard. The former name was UCC-128.
FBT_Patch	Barcode in Patch standard.
FBT_Aztec	Barcode in Aztec standard.
FBT_Datamatrix	Barcode in Data Matrix standard.
FBT_Qrcode	Barcode in QR Code standard.
FBT_Upca	Barcode in UPC-A standard.
FBT_Maxicode	Barcode in MaxiCode standard.
FBT_Any1D	Combination of all one-dimensional barcodes.
FBT_Square2D	Combination of all two-dimensional barcodes.
FBT_Any1DWithSupplement	Combination of all one-dimensional barcodes that can have a supplement.

See also

CFineBarcode

FineRecognizeBarcode

TFineCharacterAttributes

TFineCharacterAttributes enumeration constants are used as the mask in the **CFineTextCharacter** structure. The mask is an OR combination of these flags' values which define character attributes. These constants are defined using the BIT_FLAG macros.

```
typedef enum tagTFineCharacterAttributes {
    FCA_Italic                = BIT_FLAG( 0 ),
    FCA_Bold                  = BIT_FLAG( 1 ),
    FCA_Underlined            = BIT_FLAG( 2 ),
    FCA_Strikethrough         = BIT_FLAG( 3 ),
    FCA_Smallcaps             = BIT_FLAG( 4 ),
    FCA_Superscript           = BIT_FLAG( 5 ),
    FCA_Uncertain             = BIT_FLAG( 16 ),
    FCA_BarcodeBinaryDataHexed = BIT_FLAG( 17 ),
    FCA_BarcodeBinaryZero     = BIT_FLAG( 18 ),
    FCA_BarcodeStartStopSymbol = BIT_FLAG( 19 ),
} TFineCharacterAttributes;
```

Elements

Name	Description
FCA_Italic	Specifies whether the character is italic.
FCA_Bold	Specifies whether the character is bold.
FCA_Underlined	Specifies whether the character is underlined.
FCA_Strikethrough	Specifies whether the character is strikeout.
FCA_Smallcaps	Specifies whether the character has the "small caps" style. This means that the small characters are displayed as small capitals.
FCA_Superscript	Specifies whether the character is superscript.
FCA_Uncertain	Specifies whether the character has been recognized uncertainly. The confidence level at which characters are marked as uncertain must be set during recognition as a TFineRecognitionConfidenceLevel constant.
FCA_BarcodeBinaryDataHexed	Specifies a binary symbol that is written in hexadecimal format.
FCA_BarcodeBinaryZero	Specifies a zero binary symbol replaced by character that is specified in

	the FineRecognizeBarcode function as <i>unknownLetter</i> for correct representation.
FCA_BarcodeStartStopSymbol	Specifies the start and stop symbols. This flag is valid for Code39 and Codabar barcodes.

See also

CFineTextCharacter

TFineImageFileSeekPosition

TFineImageFileSeekPosition enumeration constants are used in the *from* parameter of the **Seek** method of the **CFineImageFile** structure.

```
typedef enum tagTFineImageFileSeekPosition {
    FIFSP_Begin,
    FIFSP_Current,
} TFineImageFileSeekPosition;
```

Flag

Name	Description
FIFSP_Begin	Specifies that the <i>from</i> parameter should equal the <i>offset</i> parameter.
FIFSP_Current	Specifies that the <i>from</i> parameter should be incremented to the <i>offset</i> parameter.

See also

CFineImageFile::Seek

TFineImageLoadingOptionsFlags

TFineImageLoadingOptionsFlags enumeration constants are used to set the input parameter of the **FineLoadImageFromFile** and **FineLoadImageFromInputStream** functions. Some of the constants are defined using the BIT_FLAG macros.

```
typedef enum tagTFineImageLoadingOptionsFlags{
    FILO_Default = 0,
    FILO_ApplyExifOrientation = BIT_FLAG( 0 ),
    FILO_CropUsingRelativeCoordinates = BIT_FLAG( 1 )
} TFineImageLoadingOptionsFlags;
```

Flag

Name	Description
FILO_Default	Specifies that an image is loaded as is, ignoring any metadata, and the crop rectangle coordinates are in pixels.
FILO_ApplyExifOrientation	Specifies that the EXIF orientation information is applied to the image.
FILO_CropUsingRelativeCoordinates	Specifies that the crop rectangle coordinates are in the ten thousandth of the original image size.

See also

FineLoadImageFromFile

FineLoadImageFromInputStream

TFineImageProcessingOptionsFlags

TFineImageProcessingOptionsFlags enumeration constants are used to set the input parameter of the **FinePreprocessImage** function. Some of the constants are defined using the BIT_FLAG macros.

```
typedef enum tagTFineImageProcessingOptionsFlags{
    FIPO_Default = 0,
    FIPO_DisableDeskew = BIT_FLAG( 0 ),
    FIPO_DisableImageGeometricTransform = FIPO_DisableDeskew,
    FIPO_DetectPageOrientation = BIT_FLAG( 1 ),
    FIPO_HasCjk = BIT_FLAG( 4 ),
    FIPO_FindAllText = BIT_FLAG( 5 ),
    FIPO_IsEuropeanWithSomeCjk = BIT_FLAG( 6 ),
    FIPO_ProhibitVerticalCjkText = BIT_FLAG( 7 ),
    FIPO_MicrMode = BIT_FLAG( 8 ),
    FIPO_BuildWordsInfo = BIT_FLAG( 9 ),
    FIPO_PrebuildWordsInfo = BIT_FLAG( 10 ),
    FIPO_UseOldBinarization = BIT_FLAG( 11 )
} TFineImageProcessingOptionsFlags;
```

Flag

Name	Description
FIPO_BuildWordsInfo	If this flag is set, then, after recognition, the CFineTextLine::WordInfo field stores an array of CFineWordInfo structures with the CFineTextLine::WordInfoCount elements.
FIPO_Default	If this constant is set, input image will be processed with default parameters.
FIPO_DetectPageOrientation	Specifies whether the page orientation should be detected. The portrait or landscape page orientation will be detected. If this flag is set, the <i>rotation</i> parameter of the FineRecognizeImage , FineRecognizeRegion and FineRecognizeBusinessCard functions returns the rotation angle multiple of 90 degrees. These functions return the recognized text coordinates corresponding to a rotated image. Note: Setting this flag decreases recognition speed. Also detection of page orientation requires additional 0.5-1 MB RAM.
FIPO_DisableDeskew	Turns off automatic skew correction. Note: If you do not use this option, skew correction is performed during image preprocessing. For adequate skew correction, the skew angle should not exceed 16 degrees.
FIPO_DisableImageGeometricTransform	Equal to FIPO_DisableDeskew.
FIPO_FindAllText	If this constant is set, the program will find all text on image. Pictures and garbage will be analyzed and recognized.
FIPO_HasCjk	Specifies whether the input image has Asian characters. This constant is automatically added into the input parameters in the recognition functions if the input list of the recognition languages of these functions contains a CJK language.
FIPO_IsEuropeanWithSomeCjk	Specifies whether the input image has text that is written in European and CJK languages. This constant is automatically added into the input parameters in the FineRecognizeBusinessCard function if the input list of the recognition languages of this function contains a CJK language. Note: Setting this constant increases recognition speed on images which contain text written in CJK and European languages. If text on the image is written only in a CJK language, it could decrease recognition quality.
FIPO_MicrMode	Specifies whether the MICR E13B font must be recognized. See

	<p>the Recognizing in MICR Mode section for details.</p> <p>Note: Only the MICR E13B characters are recognized in the FIPO_MicrMode mode, all other fonts are ignored.</p>
FIPO_PrebuildWordsInfo	<p>If this flag is set, the information about the document layout, text blocks and lines is prebuilt before the text recognition. The TFinePrebuiltDataCallbackFunction callback is called before the text recognition, and the <i>data</i> argument points to a CFinePrebuiltLayoutInfo structure.</p> <p>Note: This flag is ignored if the FIPO_HasCjk flag is set.</p>
FIPO_ProhibitVerticalCjkText	<p>If this constant is set, the program will recognize only the horizontal CJK text on image, all vertical CJK text will be ignored.</p>
FIPO_UseOldBinarization	<p>If this constant is set, fast binarization mechanism will not be used. Image binarization will be slower, but for CJK languages recognition quality may improve.</p>

See also

FineAnalyzeImage

FineGetTextLines

FinePrebuildWordsInfo

FinePreprocessImage

FineRecognizeBlocks

FineRecognizeBusinessCard

FineRecognizeImage

FineRecognizeRegion

TFinePrebuiltDataType

TFinePrebuiltDataType enumeration constants are used to specify the type of the data that are obtained before the text recognition. It is used in the **TFinePrebuiltDataCallbackFunction** callback to specify the pointer type to which the *data* argument should be cast.

```
typedef enum tagTFinePrebuiltDataType {
    FPDT_RotationType = 0,
    FPDT_WordsInfo    = 1
} TFinePrebuiltDataType;
```

Elements

Name	Description
FPDT_RotationType	<p>If the <i>dataType</i> argument of the TFinePrebuiltDataCallbackFunction callback function is set to FPDT_RotationType, the <i>data</i> argument of that function should be cast to the (TFineRotationType*) pointer type. If the FIPO_DetectPageOrientation flag in the image processing options of the recognition function is set, then the callback function with this data type delivers the detected rotation type.</p> <p>Note: If both FIPO_DetectPageOrientation and FIPO_PrebuildWordsInfo are set in the image processing options of the recognition function, the callback function with the FPDT_WordsInfo data type is called after the callback function with the FPDT_RotationType data type.</p>
FPDT_WordsInfo	<p>If the <i>dataType</i> argument of the TFinePrebuiltDataCallbackFunction callback function is set to FPDT_WordsInfo, the <i>data</i> argument of the that function should be cast to the (CFinePrebuiltLayoutInfo*) pointer type. If the FIPO_PrebuildWordsInfo flag is set in the image processing options of the recognition function, then the callback function with this data type delivers the prebuilt information about the document layout, including the approximate positions of the words.</p>

Note: If both FIPO_DetectPageOrientation and FIPO_PrebuildWordsInfo are set in the image processing options of the recognition function, the callback function with the FPDT_WordsInfo data type is called after the callback function with the FPDT_RotationType data type.

See also

TFinePrebuiltDataCallbackFunction

TFineRecognitionConfidenceLevel

TFineRecognitionConfidenceLevel enumeration constants are used to set the level of recognition confidence at which the recognized characters will receive the FCA_Uncertain attribute.

```
typedef enum tagTFineRecognitionConfidenceLevel {
    FRCL_Level0 = 0,
    FRCL_Level1 = 1,
    FRCL_Level2 = 2,
    FRCL_Level3 = 3,
    FRCL_Level4 = 4
} TFineRecognitionConfidenceLevel;
```

Elements

Name	Description
FRCL_Level0	No characters are marked as uncertain.
FRCL_Level1	Only very uncertainly recognized characters are marked.
FRCL_Level2	Medium marking level.
FRCL_Level3	Standard uncertain characters marking level.
FRCL_Level4	All suspicious characters are marked.

See also

FineAnalyzeImage

FineRecognizeBlocks

FineRecognizeImage

FineRecognizeRegion

FineRecognizeBusinessCard

CFineTextCharacter

TFineRecognitionMode

TFineRecognitionMode enumeration constants are used to set the recognition mode.

```
typedef enum tagTFineRecognitionMode{
    FRM_Fast = 0,
    FRM_Full = 1
} TFineRecognitionMode;
```

Elements

Name	Description
FRM_Fast	This mode provides 25% faster recognition speed for European languages.
FRM_Full	The full recognition mode.

See also

FineAnalyzeImage
FineRecognizeBlocks
FineRecognizeImage
FineRecognizeRegion
FineRecognizeBusinessCard

TFineRotationType

TFineRotationType enumeration constants are used to specify the image rotation angle.

```
typedef enum tagTFineRotationType {
    FRT_NoRotation,
    FRT_Clockwise,
    FRT_UpsideDown,
    FRT_Counterclockwise
} TFineRotationType;
```

Elements

Name	Description
FRT_NoRotation	No rotation.
FRT_Clockwise	The image is rotated by 90 degrees clockwise.
FRT_UpsideDown	The image is rotated by 180 degrees.
FRT_Counterclockwise	The image is rotated by 90 degrees counterclockwise.

See also

FineAnalyzeImage
TFinePrebuiltDataCallbackFunction
FineRecognizeBlocks
FineRecognizeImage
FineRecognizeRegion
FineRecognizeBusinessCard

TFineSupportedCodepage

TFineSupportedCodepage enumeration constants are used to set the barcode code page.

```
typedef enum tagTFineSupportedCodepage {
    FSC_Arabic = 1256,
    FSC_ArabicIso = 28596,
    FSC_BalticIso = 28594,
    FSC_Cyrillic = 1251,
    FSC_CyrillicIso = 28595,
    FSC_CyrillicKoi8 = 20866,
    FSC_EasternEuropean = 1250,
    FSC_EasternEuropeanIso = 28592,
    FSC_GreekIso = 28597,
    FSC_HebrewIso = 28598,
    FSC_JapanSjis = 932,
    FSC_Latin = 1252,
    FSC_Latin5Iso = 28599,
    FSC_LatinIso = 28591,
    FSC_TurkishIso = 28593,
    FSC_UsMsdos = 437,
    FSC_Utf8 = 65001
}
```

```
} TFineSupportedCodepage;
```

Elements

Name	Description
FSC_Arabic	Arabic (1256)
FSC_ArabicIso	ISO Arabic (8859-6)
FSC_BalticIso	ISO Baltic (8859-4)
FSC_Cyrillic	Windows Cyrillic (1251)
FSC_CyrillicIso	ISO Cyrillic (8859-5)
FSC_CyrillicKoi8	KOI8 Cyrillic
FSC_EasternEuropean	Windows Central Europe (1250)
FSC_EasternEuropeanIso	ISO Central Europe (8859-2)
FSC_GreekIso	ISO Greek (8859-7)
FSC_HebrewIso	ISO Hebrew (8859-8)
FSC_JapanSjis	Japanese (Shift-JIS)
FSC_Latin	Windows Western Europe (1252)
FSC_Latin5Iso	ISO Turkish (8859-9)
FSC_LatinIso	ISO Latin 1 (8859-1)
FSC_TurkishIso	ISO Latin 3 (8859-3)
FSC_UsMsdos	DOS United States (437)
FSC_Utf8	Unicode UTF-8

See also

FineRecognizeBarcode

TFineCharacterAttributes

TFineCharacterAttributes enumeration constants are used as the mask in the **CFineTextCharacter** structure. The mask is an OR combination of these flags' values which define character attributes. These constants are defined using the BIT_FLAG macros.

```
typedef enum tagTFineCharacterAttributes {
    FCA_Italic                = BIT_FLAG( 0 ),
    FCA_Bold                  = BIT_FLAG( 1 ),
    FCA_Underlined            = BIT_FLAG( 2 ),
    FCA_Strikethrough         = BIT_FLAG( 3 ),
    FCA_Smallcaps             = BIT_FLAG( 4 ),
    FCA_Superscript           = BIT_FLAG( 5 ),
    FCA_Uncertain             = BIT_FLAG( 16 ),
    FCA_BarcodeBinaryDataHexed = BIT_FLAG( 17 ),
    FCA_BarcodeBinaryZero     = BIT_FLAG( 18 ),
    FCA_BarcodeStartStopSymbol = BIT_FLAG( 19 ),
} TFineCharacterAttributes;
```

Elements

Name	Description
FCA_Italic	Specifies whether the character is italic.
FCA_Bold	Specifies whether the character is bold.
FCA_Underlined	Specifies whether the character is underlined.
FCA_Strikethrough	Specifies whether the character is strikeout.

FCA_Smallcaps	Specifies whether the character has the "small caps" style. This means that the small characters are displayed as small capitals.
FCA_Superscript	Specifies whether the character is superscript.
FCA_Uncertain	Specifies whether the character has been recognized uncertainly. The confidence level at which characters are marked as uncertain must be set during recognition as a TFineRecognitionConfidenceLevel constant.
FCA_BarcodeBinaryDataHexed	Specifies a binary symbol that is written in hexadecimal format.
FCA_BarcodeBinaryZero	Specifies a zero binary symbol replaced by character that is specified in the FineRecognizeBarcode function as <i>unknownLetter</i> for correct representation.
FCA_BarcodeStartStopSymbol	Specifies the start and stop symbols. This flag is valid for Code39 and Codabar barcodes.

See also

CFineTextCharacter

TFineTextCharacterQuality

TFineTextCharacterQuality enumeration constants are used to set the character recognition quality.

```
typedef enum tagTFineTextCharacterQuality {
    FTCQ_Min = 0,
    FTCQ_Max = 100
} TFineTextCharacterQuality;
```

See also

CFineTextCharacter

TFineWarningCode

TFineWarningCode enumeration constants are used to describe codes of warnings which are passed through the callback functions.

```
typedef enum tagTFineWarningCode {
    FWC_NoWarning,
    FWC_SlowRecognition,
    FWC_ProbablyBadImage,
    FWC_ProbablyWrongLanguages,
    FWC_SureWrongLanguages
} TFineWarningCode;
```

Elements

Name	Description
FWC_NoWarning	No warning.
FWC_SlowRecognition	The recognition process takes a long time.
FWC_ProbablyBadImage	The image quality is low.
FWC_ProbablyWrongLanguages	The language of the document is likely to be different from the recognition language you specified. In this case a pointer to the CFineWarningDataWrongLanguages structure with recommended recognition languages will be stored in the <i>warningData</i> parameter of the TFineProgressCallbackFunction .
FWC_SureWrongLanguages	The wrong recognition language is specified.

See also

TFineProgressCallbackFunction

TFinePrebuiltDataCallbackFunction

TFineWordAttributes

TFineWordAttributes enumeration constants are used to specify the word attributes.

```
enum TFineWordAttributes {
    FWA_NotWord          = BIT_FLAG( 1 ),
    FWA_SplitWord        = BIT_FLAG( 2 ),
    FWA_HyphenatedWord   = BIT_FLAG( 3 )
};
```

Elements

Name	Description
FWA_NotWord	Non-word part of the text such as a space, punctuation, and so on. In this case, there should be the only one word variant with FWVT_Original type.
FWA_SplitWord	A part of a long word.
FWA_HyphenatedWord	The word contains two parts separated by a new line. The word with this flag is always the last one in the text line. You should ignore the rectangle of a word with this attribute.

See also

CFineWordInfo

TFineWordVariantType

TFineWordVariantType enumeration constants are used to specify the type of the recognized word variant.

```
typedef enum tagTFineWordVariantType {
    FWVT_Original,
    FWVT_PrimaryForm
} TFineWordVariantType;
```

Elements

Name	Description
FWVT_Original	The original word as it is recognized. This variant always exists.
FWVT_PrimaryForm	The primary word form.

See also

CFineWordVariant

TLanguageID

TLanguageID enumeration constants are used in ABBYY Mobile OCR Engine for internal representation of language ID. The name of every constant is constructed as LID_<internal language name>. See Recognition Languages in ABBYY Mobile OCR Engine for the list of languages and their internal names.

```
typedef enum tagTLanguageID {
    LID_Undefined,
    LID_Afrikaans,
    LID_Albanian,
    LID_Basque,
    LID_Breton,
```

```

LID_Bulgarian,
LID_Byelorussian,
LID_Catalan,
LID_Chechen,
LID_CrimeanTatar,
LID_Croatian,
LID_Czech,
LID_Danish,
LID_Dutch,
LID_DutchBelgian,
LID_English,
LID_Estonian,
LID_Fijian,
LID_Finnish,
LID_French,
LID_German,
LID_GermanNewSpelling,
LID_Greek,
LID_Hawaiian,
LID_Hungarian,
LID_Icelandic,
LID_Indonesian,
LID_Irish,
LID_Italian,
LID_Kabardian,
LID_Latin,
LID_Latvian,
LID_Lithuanian,
LID_Macedonian,
LID_Malay,
LID_Maori,
LID_Mixed,
LID_Moldavian,
LID_Mongol,
LID_Norwegian,
LID_NorwegianBokmal,
LID_NorwegianNynorsk,
LID_Ossetic,
LID_Polish,
LID_Portuguese,
LID_PortugueseBrazilian,
LID_Provencal,
LID_RhaetoRomanic,
LID_Romanian,
LID_Russian,
LID_Samoan,
LID_Serbian,
LID_Slovak,
LID_Slovenian,
LID_Spanish,
LID_Swahili,
LID_Swedish,
LID_Tagalog,
LID_Tatar,
LID_Turkish,

```

```
LID_Ukrainian,  
LID_Welsh,  
LID_Digits,  
LID_WestEuropean,  
LID_FirstCJKLanguageID,  
LID_ChineseSimplified = LID_FirstCJKLanguageID,  
LID_ChineseTraditional,  
LID_Japanese,  
LID_Korean,  
LID_KoreanHangul,  
LID_KoreanHanja,  
LID_LastCJKLanguageID = LID_KoreanHanja,  
LID_FirstUserLanguageID = 1024,  
LID_Max                 = 0xffff  
} TLanguageID;
```

Note: Count for user languages' IDs starts from LID_FirstUserLanguageID.

See also

Recognition Languages in ABBYY Mobile OCR Engine

FineAnalyzeImage

FineRecognizeImage

FineRecognizeRegion

Licensing

A special protection technology is used to protect ABBYY Mobile OCR Engine 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 Mobile OCR Engine has two types of licenses:

- **Developer License**
This license grants an SDK customer the right to use ABBYY Mobile OCR 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 Mobile OCR Engine functions inside or to use the developed applications internally.
- **Runtime License**
This license grants developers the right to distribute ABBYY Mobile OCR Engine functions inside developer's applications. Runtime licensing is regulated by Runtime License Agreement with ABBYY.

Note: If you use a trial license, the word "ABBYY" will appear in each 20th line in the recognized text and in each 3d recognized business card.

ABBYY Mobile OCR Engine license is stored in a license file (*.License). No operations with ABBYY Mobile OCR Engine may be performed until a valid license is loaded.

Loading the license in native library

To add license information to an application, do the following:

1. Load the license file into memory.
2. Assign the **LicenseData** field of the **CFineLicenseInfo** structure to a pointer to the memory buffer which contains loaded data.
3. Specify the **DataLength** and **ApplicationID** fields of the structure. The **ApplicationID** field must correspond to the application name in the license file.
4. Pass a constant pointer to the **CFineLicenseInfo** variable to the **FineSetLicenseInfo** function.

Use the **FineGetLicenseInfo** function to get information about the current license.

Copyright and Trademark Notices

© 2013 ABBYY Production LLC. All rights reserved

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.

All other trademarks are the property of their respective owners.

Working with JPEG image format:

This software is based in part on the work of the Independent JPEG Group.

Opening DjVu image format:

Portions of this computer program are copyright © 2008 Celartem, Inc. All rights reserved.

Portions of this computer program are copyright © 2011 Caminova, Inc. All rights reserved.

Portions of this computer program are copyright © 2013 Cuminas, Inc. All rights reserved.

DjVu is protected by U.S. Patent No. 6,058,214. Foreign Patents Pending.

Powered by AT&T Labs Technology.

Creating and manipulating processing threads:

Copyright © 2001,2006 Ross P. Johnson

All rights reserved.

Pthreads-win32 library is covered by the GNU Lesser General Public License. A copy of the License can be found under the filename COPYING.LIB

Specifications

This section contains the description of ABBYY Mobile OCR Engine general features and technical requirements:

- Recognition Languages
- Supported Image Formats
- Barcode Types
- System Requirements
- Backward Compatibility Issues
- ABBYY Mobile OCR Engine Distribution Kit
- What's New in ABBYY Mobile OCR Engine 4 release 14

Recognition Languages in ABBYY Mobile OCR Engine

Below is the list of internal names of the languages that are supported in ABBYY Mobile OCR Engine 4. Those languages which have full built-in dictionary support and keywords dictionary support for business card recognition (BCR) are marked in the table below. ABBYY Mobile OCR Engine 4 provides its own system dictionaries for the languages that have full built-in dictionary support.

Internal name	Recognition language	Full dictionary support available	Can be used for for BCR
Afrikaans	Afrikaans		
Albanian	Albanian		
Basque	Basque		
Byelorussian	Belarussian		
Breton	Breton		
Bulgarian	Bulgarian	+	
Catalan	Catalan		
Chechen	Chechen		
ChineseSimplified	Chinese Simplified		+
ChineseTraditional	Chinese Traditional		+
CrimeanTatar	Crimean Tatar		
Croatian	Croatian		
Czech	Czech	+	+
Danish	Danish	+	+
Digits	Digits		
Dutch	Dutch (Netherlands)	+	+
DutchBelgian	Dutch (Belgium)	+	+
English	English	+	+
Estonian	Estonian	+	+
Fijian	Fijian		
Finnish	Finnish	+	+
French	French	+	+

German	German	+	+
GermanNewSpelling	German (new spelling)	+	+
Greek	Greek	+	+
Hawaiian	Hawaiian		
Hungarian	Hungarian		
Icelandic	Icelandic		
Indonesian	Indonesian	+	+
Irish	Irish		
Italian	Italian	+	+
Japanese	Japanese		+
Kabardian	Kabardian		
Korean	Korean		+
KoreanHanja	Hanja		
KoreanHangul	Hangul		
Latin	Latin		
Latvian	Latvian		
Lithuanian	Lithuanian		
Macedonian	Macedonian		
Malay	Malay		
Maori	Maori		
Mixed	English + Russian	+	
Moldavian	Moldavian		
Mongol	Mongol		
Norwegian	Norwegian (Bokmal) + Norwegian (Nynorsk)	+	+
NorwegianBokmal	Norwegian (Bokmal)	+	+
NorwegianNynorsk	Norwegian (Nynorsk)	+	+
Ossetic	Ossetic		
Polish	Polish	+	+
Portuguese	Portuguese	+	+
PortugueseBrazilian	Portuguese (Brazil)	+	+
Provençal	Provençal		
RhaetoRomanic	Rhaeto-Romanic		
Romanian	Romanian		
Russian	Russian	+	+
Samoan	Samoan		
Serbian	Serbian		
Slovak	Slovak		
Slovenian	Slovenian		
Spanish	Spanish	+	+
Swahili	Swahili		
Swedish	Swedish	+	+
Tagalog	Tagalog		

Tatar	Tatar		
Turkish	Turkish	+	+
Ukrainian	Ukrainian	+	+
Welsh	Welsh		
WestEuropean	English + French + German + Portuguese + Spanish + Italian	+	+

See also

Working with Dictionaries

Working with Languages

TLanguageID

Supported Image Formats

The ABBYY Mobile OCR Engine library supports loading the images in the following formats:

- JPEG
- PNG

If you need to load an image in another format, you must load it into memory and convert it to the **CFineImage** format before calling the library functions.

Note: The Mobile OCR Engine will not open images larger than 32512*32512 pixels.

For accurate recognition, the images must conform to certain requirements:

- the letters' size must be 2 to 20 mm, and not less than 10 pixels on image
- the best resolution for texts printed in fonts 10 pt or larger is 300 dpi
- the best resolution for texts printed in fonts 9 pt or smaller is 400–600 dpi

See also

How to Use the Native Library

Description of the Sample

Barcode Types

ABBYY Mobile OCR Engine can recognize barcodes of the following types:

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.
GS1-128	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 GS1-128 barcode on your cartons being shipped out to your customers. The former name was UCC-128.
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.
MaxiCode	MaxiCode is two-dimensional machine-readable code that uses dots arranged in a hexagonal grid. It is usually one inch square that can store around one hundred characters of information and usually used for tracking and managing the shipment of packages.
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.
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 Universal Product Code (UPC) compresses the data characters and the checksum into six characters. Only tags with a number system character of zero can be encoded into UPC-E. In addition, the original ten data characters must have at least four zeros. This bar code is ideal for small packages because it is the smallest bar code.

See also

TFineBarcodeType
FineRecognizeBarcode

System Requirements

Supported Operating Systems

The Mobile OCR Engine supports Win32.

ABBYY offers professional services to port the software to other platforms and to customize the software for special tasks.

The ABBYY Mobile OCR Engine native library may be used for testing. The ABBYY Mobile OCR Engine library supplied as DLL and as static library and as a wrapper of the library for Android and iOS may be found in the appropriate distributions.

Backward Compatibility Issues of ABBYY Mobile OCR Engine

This section contains the descriptions of compatibility issues:

- Compatibility with previous releases of version 4
- Compatibility with version 3.0 or older

Compatibility of ABBYY Mobile OCR Engine 4 release 14 with previous releases

Different builds of ABBYY Mobile OCR Engine are not binary compatible. Applications that were compiled using earlier builds of ABBYY Mobile OCR Engine should be recompiled. Some changes in the source code may be necessary because of the improvements in the API. In the table below you can find the API changes introduced in the different builds.

Build #	Changes	Required code modifications
4.0 r12	The <i>rotation</i> property was removed from the FineRecognizeBlocks method.	Remove all mentions of the <i>rotation</i> property of the FineRecognizeBlocks method.

4.0 r10	ABBYY Mobile OCR Engine does not support the Symbian, WinCE, and WinMobile operating systems.	The Symbian, WinCE, and WinMobile operating systems are no longer supported.
	The Yiddish recognition language is not supported any longer.	Remove the Yiddish language from the list of the recognition languages.
	The TCallbackFunction function has been replaced by TFineProgressCallbackFunction.	Replace the TCallbackFunction function by TFineProgressCallbackFunction .
	<p>The following types have been deprecated:</p> <p>PFINE_ANGLE PFINE_BARCODE PFINE_BCR_FIELD PFINE_BUSINESS_CARD PFINE_FIELD_COMPONENT PFINE_IMAGE PFINE_IMAGE_TRANSFORMATION_INFO PFINE_LAYOUT PFINE_LICENSE_INFO PFINE_RECTS PFINE_TEXT_BLOCK PFINE_TEXT_CHARACTER PFINE_TEXT_LINE PFINE_WARNING_DATA_WRONG_LANGUAGE PFINE_WORD_SUGGESTION</p>	Replace the PFINE_* types to the corresponding pointers.
	The constants of the TFineBarcodeSupplement enumeration have been renamed.	<p>Rename the TFineBarcodeSupplement enumeration constants as follows:</p> <p>FBS_VOID = FBS_Void, FBS_2DIGIT = FBS_2Digit, FBS_5DIGIT = FBS_5Digit, FBS_AUTODETECT = FBS_AutoDetect, FBS_ANY_SUPPLEMENT = FBS_AnySupplement</p>
	The constants of the TFineBarcodeOrientation enumeration have been renamed.	<p>Rename the TFineBarcodeOrientation enumeration constants as follows:</p> <p>FBO_LEFT_TO_RIGHT = FBO_LeftToRight, FBO_DOWN_TO_TOP = FBO_DownToTop, FBO_RIGHT_TO_LEFT = FBO_RightToLeft, FBO_TOP_TO_DOWN = FBO_TopToDown, FBO_AUTODETECT = FBO_AutoDetect</p>
	The constants of the TFineBarcodeType enumeration have been renamed.	<p>Rename the TFineBarcodeType enumeration constants as follows:</p> <p>FBT_UNRECOGNIZED = FBT_Unrecognized FBT_CODE39 = FBT_Code39 FBT_INTERLEAVED25 = FBT_Interleaved25 FBT_EAN13 = FBT_Ean13 FBT_CODE128 = FBT_Code128 FBT_EAN8 = FBT_Ean8 FBT_PDF417 = FBT_Pdf417 FBT_CODABAR = FBT_Codabar FBT_UPCE = FBT_Upce FBT_INDUSTRIAL25 = FBT_Industrial25 FBT_IATA25 = FBT_Iata25 FBT_MATRIX25 = FBT_Matrix25 FBT_CODE93 = FBT_Code93 FBT_POSTNET = FBT_Postnet FBT_UCC128 = FBT_Ucc128 FBT_PATCH = FBT_Patch</p>

	<p>FBT_AZTEC = FBT_Aztec FBT_DATAMATRIX = FBT_Datamatrix FBT_QRCODE = FBT_Qrcode FBT_UPCA = FBT_Upca FBT_MAXICODE = FBT_Maxicode FBT_ANY1D = FBT_Any1D FBT_SQUARE2D = FBT_Square2D FBT_ANY1D_WITH_SUPPLEMENT = FBT_Any1DWithSupplement</p>
The TRecognitionConfidenceLevel enumeration and its constants have been renamed.	<p>Rename the TRecognitionConfidenceLevel enumeration to TFineRecognitionConfidenceLevel and its constants as follows:</p> <p>RCL_Level0 = FRCL_Level0, RCL_Level1 = FRCL_Level1, RCL_Level2 = FRCL_Level2, RCL_Level3 = FRCL_Level3, RCL_Level4 = FRCL_Level4</p>
The TRecognitionMode enumeration and its constants have been renamed.	<p>Rename the TRecognitionMode enumeration to TFineRecognitionMode and its constants as follows:</p> <p>RM_Fast = FRM_Fast, RM_Full = FRM_Full</p>
The PFINE_PATTERNS, PFINE_DICTIONARY, and PFINE_KEYWORDS types have been replaced by TFinePatternsPtr, TFineDictionaryPtr, and TFineKeywordsPtr, respectively.	<p>Replace the variables of the PFINE_PATTERNS, PFINE_DICTIONARY, and PFINE_KEYWORDS types by TFinePatternsPtr, TFineDictionaryPtr, and TFineKeywordsPtr, respectively. See Types in ABBYY Mobile OCR Engine Native Library for details.</p>
The FINE_ATTR_ prefixed flags has been replaced by the TFineCharacterAttributes enumeration.	<p>Replace all FINE_ATTR_ prefixed flags by the TFineCharacterAttributes enumeration constants as follows:</p> <p>FINE_ATTR_ITALIC = FCA_Italic, FINE_ATTR_BOLD = FCA_Bold, FINE_ATTR_UNDERLINED = FCA_Underlined, FINE_ATTR_STRIKETHROUGH = FCA_Strikethrough, FINE_ATTR_SMALLCAPS = FCA_Smallcaps, FINE_ATTR_SUPERSCRIPT = FCA_Superscript, FINE_ATTR_UNCERTAIN = FCA_Uncertain, FINE_ATTR_BARCODE_BINARY_DATA_HEXED = FCA_BarcodeBinaryDataHexed, FINE_ATTR_BARCODE_BINARY_ZERO = FCA_BarcodeBinaryZero, FINE_ATTR_BARCODE_START_STOP_SYMBOL = FCA_BarcodeStartStopSymbol</p>
The TFineImageProcessingOptions, TBCRFieldType, TBCRComponentType, FINE_WARNING_CODE enumerations have been renamed to TFineImageProcessingOptionsFlags, TBCrFieldType, TBCrComponentType, and TFineWarningCode, respectively.	<p>Rename the TFineImageProcessingOptions,, TBCRFieldType, TBCRComponentType, FINE_WARNING_CODE enumerations to TFineImageProcessingOptionsFlags, TBCrFieldType, TBCrComponentType, and TFineWarningCode, respectively.</p>
The FINE_ANGLE, FINE_BARCODE, FINE_BCR_FIELD, FINE_BUSINESS_CARD,	<p>Replace variables of the FINE_ANGLE, FINE_BARCODE, FINE_BCR_FIELD,</p>

	<p>FINE_FIELD_COMPONENT, FINE_IMAGE, FINE_IMAGE_TRANSFORMATION_INFO, FINE_LAYOUT, FINE_LICENSE_INFO, FINE_RECTS, FINE_TEXT_BLOCK, FINE_TEXT_CHARACTER, FINE_TEXT_LINE, FINE_WARNING_DATA_WRONG_LANGUAGE, and FINE_WORD_SUGGESTION structures have been replaced by CFineAngle, CFineBarcode, CFineBcrField, CFineBusinessCard, CFineBcrFieldComponent, CFineImage, CFineImageTransformationInfo, CFineLayout, CFineLicenseInfo, CFineRects, CFineTextBlock, CFineTextCharacter, CFineTextLine, CFineWarningDataWrongLanguages, and CFineWordSuggestion, respectively.</p>	<p>FINE_BUSINESS_CARD, FINE_FIELD_COMPONENT, FINE_IMAGE, FINE_IMAGE_TRANSFORMATION_INFO, FINE_LAYOUT, FINE_LICENSE_INFO, FINE_RECTS, FINE_TEXT_BLOCK, FINE_TEXT_CHARACTER, FINE_TEXT_LINE, FINE_WARNING_DATA_WRONG_LANGUAGE, and FINE_WORD_SUGGESTION types by CFineAngle, CFineBarcode, CFineBcrField, CFineBusinessCard, CFineBcrFieldComponent, CFineImage, CFineImageTransformationInfo, CFineLayout, CFineLicenseInfo, CFineRects, CFineTextBlock, CFineTextCharacter, CFineTextLine, CFineWarningDataWrongLanguages, and CFineWordSuggestion, respectively.</p>
	<p>The FINE_ERROR_CODE enumeration and its constants have been renamed.</p>	<p>Rename the FINE_ERROR_CODE enumeration to TFineErrorCode and its constants as follows:</p> <p>FINE_ERR_NO_ERROR = FEC_NoError, FINE_ERR_NOT_INITIALIZED = FEC_NotInitialized, FINE_ERR_LICENSE_ERROR = FEC_LicenseError, FINE_ERR_INVALID_ARGUMENT = FEC_InvalidArgument, FINE_ERR_NOT_ENOUGH_MEMORY = FEC_NotEnoughMemory, FINE_ERR_INTERNAL_FAILURE = FEC_InternalFailure, FINE_ERR_TERMINATED_BY_CALLBACK = FEC_TerminatedByCallback</p>
	<p>The FINE_CUSTOMER_KEY and FINE_WARN_WRONG_LANG structures have been removed.</p>	<p>Remove variables of the FINE_CUSTOMER_KEY and FINE_WARN_WRONG_LANG types.</p>
4.0 r7	<p>The TrialKey.h file has been deleted. The trial license is stored in the /License/Sample.license file.</p>	<p>Delete the TrialKey.h file from the included files, load the license file to the memory, and pass loaded information to ABBYY Mobile OCR Engine library. See Licensing section for details.</p>
4.0 r5	<p>The BCT_AddressClarification and BCT_AddressUnclassifiedPart constants have been removed from the TBcrComponentType enumeration, and the BCT_State constant has been renamed to BCT_Region.</p>	<p>Use the BCT_StreetAddress constant instead of BCT_AddressClarification and BCT_AddressUnclassifiedPart. Rename the BCT_State constant.</p>
4.0 r2	<p>The ability to detect the page orientation has been added (the FIPO_DetectPageOrientation constant of the TFineImageProcessingOptions enumeration). Due to this change the <i>rotation</i> parameter has been added in the FineRecognizeImage, FineRecognizeRegion and FineRecognizeBusinessCard functions.</p>	<p>Add 0 as the <i>rotation</i> parameter in all calls of the FineRecognizeImage, FineRecognizeRegion and FineRecognizeBusinessCard functions.</p>
4.0.1	<p>The licensing of ABBYY Mobile OCR Engine has been changed. A customer key provides access to certain functionality of ABBYY Mobile OCR Engine. A user can have a set of keys representing necessary</p>	<p>Add license information with the help of the FineSetLicenseInfo function.</p>

	functionality. The FineSetLicenseInfo function has been created for adding license information. You have to call the FineSetLicenseInfo function for your license after calling of the FineInitialize function.	
4.0	The AllDictionaryLanguages.rom and AllLanguages.rom files have been removed. The European.rom file must be used instead.	Load the European.rom file instead of the AllDictionaryLanguages.rom and AllLanguages.rom files.
	The Quality field has been added into CFineTextCharacter structure. The FIPO_SkipDictionaryUncertainty flag has been removed from the TFineImageProcessingOptions set.	Use the value of the Quality field instead of the FIPO_SkipDictionaryUncertainty flag.
	The definition of the BFT_Text field has been changed. Now the recognized text is stored in this field.	If you want to get text fragments from the field of the BFT_Text type, which is not stored in another field, you can remove text fragments stored in another field using information about character rectangles.
	<p>The CJK languages (Chinese Simplified, Chinese Traditional, Japanese, and Korean) are now available for recognition. The <i>cjkPatterns</i> parameter has been added into FineRecognizeImage, FineRecognizeRegion, FineRecognizeBusinessCard functions. This parameter contains the zero-terminated list of pointers to the patterns for CJK languages. It must not be null and must contain valid patterns, if one of these languages is specified in the <i>languages</i> parameter.</p> <p>Notes:</p> <ul style="list-style-type: none"> The ChineseJapanese.rom file must be specified in the <i>cjkPatterns</i> parameter for recognition of text in Chinese or Japanese language. Both the ChineseJapanese.rom and KoreanSpecific.rom files must be specified in the <i>cjkPatterns</i> parameter for recognition of text in Korean language. 	Pass 0 as <i>cjkPatterns</i> parameter in the recognition functions, if the code does not perform recognition of texts in CJK languages.

Compatibility of ABBYY Mobile OCR Engine with version 3.0 and older

Different builds of ABBYY Mobile OCR Engine are not binary compatible. Applications that were compiled using earlier builds of ABBYY Mobile OCR Engine should be recompiled. Some changes in the source code may be necessary because of the improvements in the API. In the table below you can find the API changes introduced in the different builds.

Build #	Changes	Required code modifications
3.0 r4	The TrialKey.h file has been removed.	You have to use the serial number from our re-sellers.
3.0 r3	<p>The format of TFineProgressCallbackFunction has been changed:</p> <ul style="list-style-type: none"> The <i>warning</i> parameter is a 	<p>Redefine your callback function and warning handling. Here is a sample implementation:</p> <pre>int TFineProgressCallbackFunction(</pre>

	<p>TFineWarningCode constant instead of a combination.</p> <ul style="list-style-type: none"> The new <i>warningData</i> parameter has been added. It depends on the value of the <i>warning</i> parameter. If the <i>warning</i> parameter is <i>FWC_ProbablyWrongLanguages</i>, the <i>warningData</i> parameter gets a pointer to a structure which should be cast to the CFineWarningDataWrongLanguages type. For other constants of the TFineWarningCode enumerations, this parameter does not make sense and should be ignored. 	<pre>int processedPercentage, DWORD warning, void* warningData) { fprintf(TraceFile, "%d%% of the work is done.\n", processedPercentage); if(warning == FWC_ProbablyBadImage) { fprintf(TraceFile, "The image quality is too low.\n"); } if(processedPercentage < 50 && (warning == FWC_SlowRecognition) != 0) { return 0; } else { return 1; } }</pre>
2.0	The FinePreprocessImage function returns the angle by which the image skew was corrected.	Add to the FinePreprocessImage call the <i>transformationInfo</i> argument. If FinePreprocessImage completes successfully, free up <i>transformationInfo</i> using the FineFreeMemory function.
	Simple types (BYTE, etc.) are now defined through conditional compilation.	No changes are required.
	<ul style="list-style-type: none"> TFineImagePreprocessingMode enumeration was renamed to TFineImageProcessingOptions enumeration. All functions that work with images (FinePreprocessImage, FineRecognizeBusinessCard, FineGetTextLines, FineRecognizeImage, and FineRecognizeRegion) now have an argument of type TFineImageProcessingOptions. 	Pass the FIPO_Default value to the corresponding functions for the same behavior.
	Client code may obtain debug information via a callback function of type FineExecutionLogFunction .	Pass 0 for the <i>executionLogFunction</i> argument of the FineInitialize function if you do not need to receive information about the processing.
	The FineBinarizeImage function has been removed.	Use the FinePreprocessImage function.
	The FineRecognizeImage and FineRecognizeRegion image recognition functions return information about the text blocks instead of the list of lines. Now these functions return the CFineLayout structure, which contains an array of blocks (CFineTextBlock), each of which in turn contains an array of lines CFineTextLine .	To iterate through all the lines of a recognized text, you need to iterate through all the blocks.

<p>Changes were made to memory allocation. Previously, pre-allocated memory blocks were passed to API functions. The API functions worked with these blocks and issued the results via these blocks. Now, when a library is initialized, pointers to the memory allocation and release functions are passed to the FineInitialize function, and the library uses these pointers to work with the memory. Memory for the results returned by an API function is allocated inside this function. The client code must free up this memory using the FineFreeMemory function.</p>	<ul style="list-style-type: none"> For any platforms other than those identified above, define two functions of types TFineAllocMemoryFunction and TFineFreeMemoryFunction and pass their pointers to FineInitialize. In addition, you can pass pointers to system functions malloc and free as these arguments. Delete the <i>ram</i> and <i>ramSize</i> arguments in your API functions. You don't need to handle the error FINE_ERR_OUTPUT_BUFF_TOO_SMALL, as buffers for output results are now allocated inside API functions and not in client code. Change the way you work with output data produced by API functions. After using the output buffer you have to free it up using the FineFreeMemory function.
<p>The FINE_ATTR_CERTAIN_SPACE flag was removed.</p>	<p>The confidence for a space, just like confidences for other characters, should be determined via the FCA_Uncertain flag of the TFineCharacterAttributes enumeration.</p>
<p>The <i>confidenceLevel</i> argument was added to the recognition functions.</p>	<p>In the FineRecognizeImage and FineRecognizeRegion recognition functions, pass the FRCL_Level3 value as the <i>confidenceLevel</i> argument.</p>
<p>A special WestEuropean language was introduced to speed up OCR that involves multiple European languages. The WestEuropean language combines English, French, German, Portuguese, Spanish, and Italian.</p>	<p>No changes are required.</p>
<p>The CFineBcrField structure no longer contains a property describing the region of a field on the image. A region is a list of coordinates of the rectangles that enclose the field.</p>	<p>The region of a field can now be obtained by merging the rectangles from the CFineTextLine::Rect property which enclose the field lines.</p>
<p>In the CFineBcrField structure, in the TextLines property, the text of a field is now written not as a WCHAR string, but as an array of CFineTextLine lines.</p>	<p>To get the text of a field:</p> <ul style="list-style-type: none"> iterate the values of the Chars property of the CFineTextLine structure. get the value of CFineTextCharacter::Unicode from each element of the Chars property.
<p>In the TBcrFieldType enumeration, the BFT_Max constant was renamed to BFT_Count.</p>	<p>Rename BFT_Max to BFT_Count.</p>

ABBYY Mobile OCR Engine Distribution Kit

The ABBYY Mobile OCR Engine library supplied as a static library and as a wrapper of the library for Android and iOS may be found in the appropriate distributions.

This section contains a list of all files you will find in the distribution package for testing of the recognizing quality and describes the function of each file.

The list of files supplied in different ABBYY Mobile OCR Engine distribution kits may not be the same as in the list below and may vary depending on the product's version.

You can use the ABBYY Mobile OCR Engine native library for testing (Native Library API Reference). All the resources you will find in the **TestShell** folder. Select the necessary files, depending on functionality and recognition languages you intend to work with.

Folder	File name	Description
lib\	MobileOcrEngine.dll, MobileOcrEngine.lib	These files are necessary to run the applications which use ABBYY Mobile OCR Engine on desktop computers, e.g. for quality testing purposes.
	pthreadVC2.dll	This is a file for multi-threaded processing with ABBYY Mobile OCR Engine.
	COPYING	Pthreads-win32 license.
	COPYING.LIB	GNU Lesser General Public License.
TestShell\BcrData	Brazil.akw	Portuguese (Brazilian) language support for business card recognition.
	ChineseSimplified.akw	Chinese (PRC) language support for business card recognition.
	ChineseTraditional.akw	Chinese (Taiwan) language support for business card recognition.
	Czech.akw	Czech language support for business card recognition.
	Danish.akw	Danish language support for business card recognition.
	Dutch.akw	Dutch language support for business card recognition.
	English.akw	English language support for business card recognition.
	Eston.akw	Estonian language support for business card recognition.
	Finnish.akw	Finnish language support for business card recognition.
	French.akw	French language support for business card recognition.
	German.akw	German language support for business card recognition.
	Greek.akw	Greek language support for business card recognition.
	Indones.akw	Indonesian language support for business card recognition.
	Italian.akw	Italian language support for business card recognition.
	Japanese.akw	Japanese language support for business card recognition.

	Korean.akw	Korean language support for business card recognition.
	NorwBok.akw	Norwegian (Bokmal) language support for business card recognition.
	NorwNyn.akw	Norwegian (Nynorsk) language support for business card recognition.
	Polish.akw	Polish language support for business card recognition.
	Portug.akw	Portuguese language support for business card recognition.
	Russian.akw	Russian language support for business card recognition.
	Spanish.akw	Spanish language support for business card recognition.
	Swedish.akw	Swedish language support for business card recognition.
	Turkish.akw	Turkish language support for business card recognition.
	Ukrain.akw	Ukrainian language support for business card recognition.
	WestEuropean.akw	West European (English, French, German, Portuguese, Spanish, Italian) languages set support for business card recognition.
TestShell\Dictionaries	Brazil.edc	Portuguese (Brazilian) language dictionary support.
	Bulgar.edc	Bulgarian language dictionary support.
	Czech.edc	Czech language dictionary support.
	Danish.edc	Danish language dictionary support.
	Dutch.edc	Dutch language dictionary support.
	English.edc	English language dictionary support.
	Eston.edc	Estonian language dictionary support.
	Finnish.edc	Finnish language dictionary support.
	Flemmish.edc	Dutch (Belgium) language dictionary support.
	French.edc	French language dictionary support.
	German.edc	German language dictionary support.
	GermanNS.edc	German (new spelling) language dictionary support.
	Greek.edc	Greek language dictionary support.
	Indones.edc	Indonesian language dictionary support.
	Italian.edc	Italian language dictionary support.
	NorwBok.edc	Norwegian (Bokmal) language dictionary support.
	NorwNyn.edc	Norwegian (Nynorsk) language dictionary support.
	Polish.edc	Polish language dictionary support.
	Portug.edc	Portuguese language dictionary support.

	Russian.edc	Russian language dictionary support.
	Spanish.edc	Spanish language dictionary support.
	Swedish.edc	Swedish language dictionary support.
	Turkish.edc	Turkish language dictionary support.
	Ukrain.edc	Ukrainian language dictionary support.
	WestEuropean.edc	West European (English, French, German, Portuguese, Spanish, Italian) languages set dictionary support.
TestShell\Dictionaries\Full		This folder contains larger and more comprehensive dictionaries for the same languages. You can substitute any of them for the files of the same name in the TestShell\Dictionaries folder, if the size issue is unimportant for you.
TestShell\Patterns	ChineseJapanese.rom	Recognition database for Chinese, Japanese and Korean languages.
	KoreanHanja.rom	Recognition database for Korean (Hanja) language.
	KoreanSpecificBCR.rom	Recognition database for recognizing business cards in Korean language.
	KoreanSpecific.rom	Recognition database for Korean language.
	European.rom	Recognition database for all supported recognition languages except Chinese, Japanese and Korean.
	Micr.rom	Recognition database for Magnetic Ink Character Recognition (MICR) mode.
	PatternsFilesInfo.txt	This file contains the information about which patterns file can be used for recognizing which languages.

See also

List of the Recognition Languages in ABBYY Mobile OCR Engine

What's New in ABBYY Mobile OCR Engine 4 release 15

Here you can find the list of new features in ABBYY Mobile OCR Engine 4 release 15.

- Bugs fixed.

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 Mobile OCR Engine 4

You can order ABBYY Mobile OCR Engine 4 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

Technical Support

If you have any questions regarding the use of ABBYY Mobile OCR Engine 4, first of all consult this Developer's Help. Useful information can also be found in our Knowledge Base.

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 build number (to determine the build number, see the BuildInfo.txt file in the Help folder of the distribution package)
- a description of the problem
- a project that demonstrates the problem (with the necessary data files). 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 type of operating system

You can also provide any additional information you consider important.

Support contacts

North/Central Americas	Customers from USA, Canada, Japan, Mexico or other Central American countries, please contact ABBYY USA at dev_support@abbyyusa.com
Western Europe	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

Eastern Europe and the Mediterranean	Customers from Albania, Bosnia and 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
All other regions	Customers from the countries not mentioned above, please contact ABBYY Russia at SDK_Support@abbyy.com