WritePad® SDK Handwriting Recognition API

Developer's Manual

PhatWare Corp.
1314 S. GRAND BLVD. #2-175
Spokane, WA 99202-1174
USA

Telephone:	(509) 456-2179
SDK Support:	developer@phatware.com
Sales:	sales@phatware.com
Developer Web:	www.phatware.com/developer
Twitter:	@phatware
Facebook:	www.facebook.com/phatware

Table of Contents

Welcome

WritePad SDK allows you to harness the power of WritePad® natural handwriting recognition technology in your applications. It recognizes all handwriting styles: *cursive (script)*, PRINT, and MIX*ed*. Employing advanced fuzzy logic and neural net techniques; WritePad recognizes arbitrary symbol strings as well as words from a user-defined or included dictionary.

WritePad SDK includes components that allow creation of custom WritePad-based applications for Apple iOS, MAC OS, Android, Microsoft Windows (Desktop/Metro/RT), Windows Phone, and Linux.

This version of WritePad SDK supports handwriting recognition in the following languages:

- English (US, US, US Medical dictionaries)
- Danish
- Dutch
- Finnish
- French
- German
- Italian
- Norwegian
- Portuguese (Brazil and Portugal)
- Spanish
- Swedish

Please visit our Web site at http://www.phatware.com to get the latest news on WritePad and other PhatWare products. For the latest WritePad information, you may go directly to http://www.phatware.com/writepad.

Please feel free to contact us with your questions and comments by emailing us at developer@phatware.com. Please use *WritePad SDK* in the subject line, so your email can promptly reach the person best able to address your needs.

WritePad API Reference

WritePad SDK is a natural handwriting recognition system, capable of recognizing cursive, printed, and mixed handwriting. It provides dictionary support and other lexical constraints facilitating robust recognition of common words, yet still allowing entry of various mixed character sequences to elevate recognition quality. The system can perform dictionary-supported word segmentation, and in the current release does not require any training. In addition for handwriting recognition, WritePad SDK also provides spell checking for all supported languages and a module for digital ink storage, manipulation, and serialization.

API Overview

The WritePad API has two sets of functions: Ink Data Object (IDO) API and Handwriting Recognition Engine (HRE) API.

Recognition functions allow processing of digital ink into characters, symbols, or words. This API also includes functions that let you load dictionaries, add words to a user dictionary, retrieve the changed dictionary image back for saving, spell check a word, and receive a list of possible alternatives for a misspelled or a partial word.

Ink Data functions allow you to store, serialize, and manipulate digital ink in the form of individual strokes. In addition to x and y coordinates of pixels within the stroke, data also contains stroke attributes such as color and width. Ink Data object can also store and serialize text and image data with applicable attributes.

Using API

Adding basic WritePad functionality to a program is an easy process:

1. At the beginning of the application, initialize the engine by calling `HWR_InitRecognizer` and `HWR_GetRecognitionFlags`. This may be done only once (on application start), or, if you support multiple languages, when switching between languages. Make sure to provide full path names to main dictionary, user dictionary, auto corrector, and learner files. Setting any file name to NULL will disable the corresponding feature. For example, if you pass NULL as the Main dictionary file name, the main dictionary will be disabled and only a user dictionary (if specified) will be used for recognition and spell checking.

```
static BOOL enableRecognizer( BOOL bEnableReco )
{
      // TODO: you may want to use full path; otherwise do
      // not forget to copy main dictionary into the application folder.
      const char *     strUserDict = USER_DICTIONARY;
      const char *     strLearner = USER_STATISTICS;
      const char *     strCorrector = USER_CORRECTOR;
      const char *     strMainDict = DEFAULT_DICTIONARY;

      if ( bEnableReco )
      {
        if ( NULL != _recognizer )
        {
                return HWR_Reset( _recognizer );
        }
        else
        {
                int     flags = 0;
                _recognizer = HWR_InitRecognizer( strMainDict,
                                              strUserDict,
                                              strLearner,
                                              strCorrector,
                                              LANGUAGE_ENGLISH,
                                              &flags );
                if ( NULL != _recognizer )
                {
                    if ( (flags & FLAG_CORRECTOR) == 0 )
                       printf( "Warning: autocorrector did not initialize.\n" );
                    if ( (flags & FLAG_ANALYZER) == 0 )
                       printf( "Warning: statistical analyzer did not initialize.\n");
                    if ( (flags & FLAG_USERDICT) == 0 )
                       printf( "Warning: user dictionary did not initialize.\n" );
                    if ( (flags & FLAG_MAINDICT) == 0 )
                       printf( "Warning: main dictionary did not initialize.\n" );

                    // set recognizer options
                    flags = HWR_GetRecognitionFlags( _recognizer );

                    // TODO: do something with flags...

                    HWR_SetRecognitionFlags( _recognizer, flags );
                    printf( "%s recognizer is enabled.\n", HWR_GetLanguageName() );
                }
                return (NULL != _recognizer) ? TRUE : FALSE;
          }
      }
      else if ( NULL != _recognizer )
      {
           HWR_FreeRecognizer( _recognizer, strUserDict, strLearner, strCorrector );
           _recognizer = NULL;
      }
      return TRUE;
}
```

2. Prepare ink object using `INK_InitData` and `INK_AddStroke` APIs. You can create the ink object from previously stored data, or asynchronously while writing by adding new pixels to the current stroke. When the ink data object is no longer needed, do not forget to delete memory by calling `INK_FreeData`.

```
static BOOL initializeInkData()
{
      inkData = INK_InitData();
      if ( NULL == inkData )
        return FALSE;
      INK_Erase( inkData );
      for ( int i = 0; i < sizeof( aStrokes )/sizeof( aStrokes[0] ); i++ )
      {
        CGStroke ptStroke = aStrokes[i].stroke;
        INK_AddStroke( inkData, ptStroke, aStrokes[i].length, 1, 0 );
      }
      return TRUE;
}
```

3. Pass the ink object to the recognizer using `HWR_RecognizeInkData`. The recognizer will process the ink and return the most probable recognition result. Depending on the amount of the input data, the output may be a single character, or one or more words.

```
static const UCHR * recognizeInk1()
{
      const UCHR * pText = NULL;

      HWR_Reset( _recognizer );

      // HWR_RecognizeInkData function does not return until all ink is
      // recognized and may take a long time.
      // It is recommended to call HWR_RecognizeInkData from a background thread.
      // You can terminate recognizer by calling HWR_StopAsyncReco function.
      // YOU CANNOT CALL HWR_RecognizeInkData and HWR_StopAsyncReco functions
      // FROM THE SANME THREAD.

      pText = HWR_RecognizeInkData( _recognizer, inkData, -1,
                                    FALSE, FALSE, FALSE, FALSE );
      if ( pText == NULL || *pText == 0 )
      {
           return "*Error*";
      }
      if ( strcmp( pText, kEmptyWord ) == 0 )
      {
           return "*Error*";
      }

      // TODO: process the result...
      return pText;
}
```

4. In addition to the most probable text result, the recognition engine also generates multiple suggestions for each recognized word or character. You can retrieve this result as well as probability coefficients (numbers between 51 and 100 that correspond to engine's confidence) for each word using the `HWR_GetResultWord` and `HWR_GetResultWeight` APIs. This is very useful for post processing recognition results, for example checking multiple variations of the recognized word with the database to improve

```
// get multiple suggestions for each word
int wordCnt = HWR_GetResultWordCount( _recognizer );
for ( int i = 0; i < wordCnt; i++ )
{
      int     flags = HW_SPELL_CHECK | HW_SPELL_USERDICT;
        int nAltCnt = HWR_GetResultAlternativeCount( _recognizer, i );
        for ( int j = 0; j < nAltCnt; j++ )
        {
                // TODO: in this sample we add only dictionary words
                const UCHR * chrWord = HWR_GetResultWord( _recognizer, i, j );
                printf( "  %s", chrWord );
                // TODO: process recognition probability, if needed
                USHORT weight = HWR_GetResultWeight( _recognizer, i, j );
                printf( " %d\n", weight );
                if ( ! HWR_IsWordInDict( _recognizer, chrWord ) )
                {
                        // TODO: process if needed... for example, spell check this word
                        UCHR  *pWordList = (UCHR *)malloc( MAX_STRING_BUFFER );
                        *pWordList = 0;
                        if ( HWR_SpellCheckWord( _recognizer,
                                        chrWord, pWordList,
                                        MAX_STRING_BUFFER-1, flags ) == 0 &&
                                        *pWordList != 0 )
                        {
                                for ( register int n = 0; 0 != pWordList[n]
                                                && n < MAX_STRING_BUFFER; n++ )
                                {
                                        if ( pWordList[n] == PM_ALTSEP )
                                                pWordList[n] = 0;
                                }
                                for ( register int k = 0; k < MAX_STRING_BUFFER; k++ )
                                {
                                        UCHR * word = (UCHR *)&pWordList[k];
                                        printf( "    %s\n", word );
                                        while ( 0 != pWordList[k] )
                                                k++;
                                        if ( 0 == pWordList[k+1] )
                                                break;
                                }
                        }
                        free( (void *)pWordList );
                }
                // must free memory allocated for a word returned by HWR_GetResultWord
                free( (void *)chrWord );
        }
}
```

5. You can also use the handwriting recognition API without preparing the ink data object first. Instead, you can send strokes to the engine directly, as arrays of points (CGStroke) using the `HWR_RecognizerAddStroke` function. In this case, start the new recognition session by resetting the engine using `HWR_Reset`, then call `HWR_RecognizerAddStroke` repeatedly for each stroke (it is recommended to do this in a separate thread). When all strokes are sent to the engine call `HWR_Recognize` to process the data and get the result. Note that each strokes is processed immediately as received by the engine and most of the ink is already recognized before `HWR_Recognize` call. As the result, `HWR_Recognize` does not much time.

```
static const UCHR * recognizeInk2()
{
      const UCHR * pText = NULL;

      HWR_Reset( _recognizer );

      // This version does not use inkData object at all; the recognition
      // is happening on the background while strokes are added. it is
      // recommended to call HWR_RecognizerAddStroke from a different thread
      // to implement asynchronous recognizer
```

```
    for ( int i = 0; i < sizeof( aStrokes )/sizeof( aStrokes[0] ); i++ )
    {
        CGStroke ptStroke = aStrokes[i].stroke;
        HWR_RecognizerAddStroke( _recognizer, ptStroke, aStrokes[i].length );
    }

    if ( HWR_Recognize( _recognizer ) )
    {
        pText = HWR_GetResult( _recognizer );
        if ( pText == NULL || *pText == 0 )
        {
            return "*Error*";
        }

        if ( strcmp( pText, kEmptyWord ) == 0 )
        {
            return "*Error*";
        }
    }
    return pText;
}
```

6. When recognizer and/or ink data object are no longer needed, do not forget to release memory using HWR_FreeRecognizer and `INK_FreeData`. The HWR_FreeRecognizer function allows you to specify names for user dictionary, learner, and auto corrector files if you want to save any changes. Otherwise, you can pass NULL as a parameter instead of file name(s).
7. This is all you need to do to add basic hardwiring recognition to your application. Of course, WritePad SDK contains many other functions that allow user dictionary manipulation, autocorrection, recognizer learner feedback, digital ink manipulation, etc.

Ink Format

The digital ink is defined as a sequence of pixels arranged in the same order as they were written on the screen. The recognizer receives ink input as a series of points. Each point consists of three values: x and y coordinates (float value) and optional pressure (1…255). The input needs to be scaled so that the range of coordinates which may be recognized are 0-16,000 for x and y. In order for the recognizer to work properly, the average size of writing should be no less than 80 vertical points for letters such as lowercase ‘e’ or ‘o’. The pressure value is not required and ignored by the recognition engine.

Handwriting Tips

Screen protection films may negatively affect digital ink flow and, therefore, quality of handwriting recognition, especially when finder is used instead of stylus. Generally, recognition quality may depend on the quality and resolution of the touch screen digitizer.

The handwriting recognition engine and its character set and dictionaries are optimized for supported languages only. If you use words that are not in the main or user dictionary, such as rare names or words form a different language, we recommend adding these words to the user dictionary.

Always complete the entire word in the current recognition session. Do not try writing part of a word per session, it will result in bad recognition quality, because partial words are not found in the dictionary.

You can write multiple words in each recognition session, however, if you always intend to write one word only, set the Singe Word Only flag, so word segmentation is disabled.

Write large (see the *Ink Format* section above), generally, the larger the better.

Always write on the screen horizontally, not at an arbitrary angle. If you allow end users to write at an angle, the digital ink must be appropriately rotated before it is sent to the recognition engine.

If you a user expected to (hand)print characters, set the *Separate Letters* flag. Note that if this option is on, you connected characters will not be properly recognized.

Setting the *Only Known Words* flag will improve the overall recognition quality, but this will make it impossible to write words, numbers, or any other character sequences that are not found in the user or main dictionaries.

Different Operating Systems

While the WritePad SDK is developed in C++, it can be used with native as well as managed development environments on several different operating systems using same simple APIs described in this document. The SDK for each supported operating system and/or development environment includes OS and/or programming language specific Release notes file and sample code. Please refer to these resources for OS-specific information.

Currently, WritePad SDK supports iOS (Objective-C sample code), Android (Java and JNI sample code), Windows (C/C++ and C# sample code for Windows 8 Metro and Desktop).

Ink Data Object API

Data Types

Pointer to the Ink Data Object (IDO) INK_DATA_PTR

Data Structures

Stroke

The Stroke (CGTracePoint *) structure is used to represent digital ink.

```
typedef struct __tagTracePoint
{
    CGPoint     pt;
    int         pressure;
} CGTracePoint;

typedef CGTracePoint *  CGStroke;
```

To maintain compatibly with previous versions of the SDK, some recognizer functions still use CGPoint * parameter instead of CGStroke.

Image Attributes

Image attributes structure used to store and serialize embedded images.

```
typedef struct __ImageAttributes
{
      CGRect            imagerect;
      int               iZOrder;
      int               nIndex;
      void *            pImageByes;
      UInt32            nDataSize;
      void *            userData;
      UInt32            flags;
} ImageAttributes;
```

imagerect	Image frame. The image frame is usually specified in the screen coordinates from the top-left corner of the associated ink page.
iZOrder	Image Z-order for overlapping images. Usually the same as the image index.

nIndex	Image index in the image array.
pImageBytes	Raw image data. The image can be in any format, PNG, JPEG, BITMAP, etc. The SDK does not manipulate image data and treats this parameter as a memory buffer.
nDataSize	Size of the memory allocated for image data in bytes.
userData	User-defined data that can be associated with the image. This data is only stored temporarily and is not serialized. A user is responsible for any memory allocation/de-allocation associated with this parameter.

Text Attributes

Text attributes structure used to store and serialize embedded text blocks (labels).

```
typedef struct __TextAttributes
{
	CGRect		textrect;
	int		iZOrder;
	int		nIndex;
	LPCUSTR		pUnicodeText;
	UInt32		nTextLength;
	LPUSTR		pFontName;
	long		fontSize;
	UInt32		fontAttributes;
	UInt32		alignment;
	COLORREF	fontColor;
	COLORREF	backColor;
	void *		userData;
	UInt32		flags;
} TextAttributes;
```

textrect	Text label frame. The label frame is usually specified in the screen coordinates from the top-left corner of the associated ink page.
iZOrder	Text Z-order for overlapping text labels. Usually the same as the text index.
nIndex	Text label index in the text array.
pUnicodeText	NULL-terminated character string in UNICODE (little-endian 16-bit).
nTextLength	Text length in characters, not including the terminating NULL character

pFontName	Font name in UNICODE (little-endian 16-bit)
fontSize	Font size in pixels
fontAttributes	Font attributes, can be 0 or any combination of LF_FONT_BOLD, LF_FONT_ITALIC, LF_FONT_UNDERSCORE, and LF_FONT_STRIKE. Note: font size, name, and attributes are applied to the entire text label.
aligment	Text alignment (right, left, center).
fontColor	Text color in Windows format, however 4-th byte can be used for transparency, if supported.
backColor	Label background color in Windows format.
userData	User-defined data that can be associated with the text label. This data is only stored temporarily and is not serialized. A user is responsible for any memory allocation/de-allocation associated with this parameter.

Functions

INK_DATA_PTR INK_InitData()

Initializes the Ink Data Object.

Parameters

None.

Returns

Pointer to the Ink Data Object (IDO) or NULL if not enough memory.

void INK_FreeData(INK_DATA_PTR pData)

Releases memory allocated for the Ink Data object.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

None.

---

void INK_Erase(INK_DATA_PTR pData)

Erases all ink in the Ink Data object, but does not release the Ink Data Object itself.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

None.

INK_DATA_PTR INK_CreateCopy(INK_DATA_PTR pData)

Creates a copy of the Ink Data object. You must use the INK_FreeData to release memory allocated for a copied object.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

Pointer to the new Ink Data Object, or NULL if not enough memory.

---

int INK_StrokeCount(INK_DATA_PTR pData)

Returns number of strokes currently stored in the IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

Number of strokes, or 0 if empty.

BOOL INK_DeleteStroke(INK_DATA_PTR pData, int nStroke);

Deletes last stroke in the IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index to delete, if nStroke is -1the last stroke is deleted.

Returns

TRUE if the stroke is successfully deleted, FALSE otherwise.

BOOL INK_AddStroke(INK_DATA_PTR pData,
CGStroke pStroke,
int nStrokeCnt,
int iWidth,
COLORREF color)

Adds a new stroke to the IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGStroke pStroke	Stroke (array of points containing X and Y coordinates of each pixel and optional pressure).
int nStrokeCnt	Number of pixels in the stroke.
int iWidth	Stroke width in pixels (ignored by the recognition engine).
COLORREF color	Stroke color in Windows format (ignored by the recognition engine).

Returns

TRUE if the stroke is successfully added, FALSE otherwise.

---

int INK_AddEmptyStroke(INK_DATA_PTR pData, int iWidth, COLORREF color)

Adds a new empty stroke to the IDO. Use INK_AddPixelToStroke function to add pixels to this stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int iWidth	Stroke width in pixels (ignored by the recognition engine).
COLORREF color	Stroke color in Windows format (ignored by the recognition engine).

Returns

New stroke index, or -1 in case of error.

int INK_AddPixelToStroke(INK_DATA_PTR pData, int nStroke, float x, float y, int p)

Adds a new pixel at the end of the existing stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Index of the stroke to add pixel to.
float x, float y	x and y coordinates of the new pixel.
int p	ink pressure (1…255) (optional). Use DEFAULT_PRESSURE if pressure is unknown.

Returns

New pixel index, or -1 in case of error.

```
BOOL INK_GetStrokePoint(  INK_DATA_PTR pData,
                          int nStroke,
                          int nPoint,
                          float * pX,
                          float * pY )

BOOL INK_GetStrokePointP( INK_DATA_PTR pData,
                          int nStroke,
                          int nPoint,
                          float * pX,
                          float * pY,
                          int *pP )
```

Returns pixels and attributes of a specified stroke in the IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index.
int nPoint	Index of a point within the stroke.
float * pX, * pY	X and Y coordinates of the point.
int * pP	Ink pressure for the point (optional).

Returns

TRUE if is successful, or FALSE if stroke or point index is outside of the pixels array.

---

int INK_GetStroke(INK_DATA_PTR pData,
int nStroke,
CGPoint ** ppoints,
int * nWidth,
COLORREF * color)

int INK_GetStrokeP(INK_DATA_PTR pData,
int nStroke,
CGStroke * ppoints,
int * nWidth,
COLORREF * color)

Returns pixels and attributes of a specified stroke in the IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Number of strokes in IDO.
CGPoint ** ppoints	Pointer to an array of points containing X and Y coordinates of each pixel of the stroke.
CGStroke * ppoints	Pointer to an array of points containing X and Y coordinates of each pixel of the stroke with optional pressure.
int * nWidth	Pointer to the stroke width in pixels (ignored by the recognition engine).
COLORREF * color	Pointer to the stroke color in Windows format (ignored by the recognition engine).

Returns

Number of pixels in the stroke, or -1 if error.

BOOL INK_GetStrokeRect(INK_DATA_PTR pData,
int nStroke,
CGRect * rect)

Returns position and size occupied by the stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Number of strokes in IDO.
CGRect * rect	Stroke rectangle.

Returns

TRUE if the stroke is successfully deleted, FALSE otherwise.

BOOL INK_GetDataRect(INK_DATA_PTR pData, CGRect * rect)

Returns position and size occupied by all strokes currently stored in IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGRect * rect	Data rectangle.

Returns

TRUE if the stroke is successfully deleted, FALSE otherwise.

---

void INK_Undo(INK_DATA_PTR pData)

Undoes the last action, such as add stroke, delete stroke, move stroke, etc.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

None.

---

void INK_Redo(INK_DATA_PTR pData)

Redoes the last undo action.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

None.

---

void INK_EmptyUndoBuffer(INK_DATA_PTR pData)

Empties the undo/redo buffer.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

None.

BOOL INK_CanUndo(INK_DATA_PTR pData)

Checks if the undo command is possible at this time.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

TRUE, if the undo buffer is not empty.

BOOL INK_CanRedo(INK_DATA_PTR pData)

Checks if the redo command is possible at this time.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

TRUE, if the redo buffer is not empty.

---

void INK_SetUndoLevels (INK_DATA_PTR pData, int levels)

Sets the size of the Undo buffer.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData
int levels	number of Undo levels, between 1 and 100.

Returns

None.

BOOL INK_SelectAllStrokes(INK_DATA_PTR pData, BOOL bSelect)

Marks all strokes in the IDO as selected (unselected).

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
BOOL bSelect	If TRUE - selected all strokes, if FALSE – unselects all strokes.

Returns

TRUE, if one or more strokes were selected (unselected).

BOOL INK_DeleteSelectStrokes(INK_DATA_PTR pData, BOOL bAll)

Deletes selected or all strokes in the IDO.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData
BOOL bAll	If TRUE – all strokes are deleted.

Returns

TRUE, if one or more strokes were deleted.

---

void INK_SetStrokesRecognizable(INK_DATA_PTR pData,
BOOL bSet,
BOOL bSelectedOnly)

Marks all or selected strokes in the IDO as recognizable (or unrecognizable). When the IDO instance is passed to the Handwriting Recognition Engine, strokes that are marked as unrecognizable are ignored. By default, all strokes are marked as recognizable, except for recognized geometrical shapes (see `INK_EnableShapeRecognition`).

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
BOOL bSet	If TRUE – marks strokes as recognizable; if FALSE – marks strokes as unrecognizable.

BOOL bSelectedOnly	If TRUE – marks selected strokes only; if FALSE – marks all strokes stored in the current IDO instance.

Returns

None.

---

void INK_SetStrokeRecognizable(INK_DATA_PTR pData, int nStroke, BOOL bSet)

Marks the specified strokes in the IDO as recognizable (or unrecognizable). When the IDO instance is passed to the Handwriting Recognition Engine, strokes that are marked as unrecognizable are ignored. By default, all strokes are marked as recognizable, except for recognized geometrical shapes (see `INK_EnableShapeRecognition`).

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index.
BOOL bSet	If TRUE – marks the stroke as recognizable; if FALSE – marks the stroke as unrecognizable.

Returns

None.

---

void INK_SelectStroke (INK_DATA_PTR pData, int nStroke, BOOL bSelect)

Marks the specified strokes in the IDO as selected (or unselected).

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index.
BOOL bSelect	If TRUE – marks the stroke as selected; if FALSE – marks the stroke as unselected.

Returns

None.

BOOL INK_IsStrokeRecognizable(INK_DATA_PTR pData, int nStroke)

Returns TRUE if the specified stroke is marked as recognizable, otherwise returns FALSE.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index.

Returns

TRUE if the specified stroke is marked as recognizable.

SHAPETYPE INK_RecognizeShape(CGStroke * pStroke,
int nStrokeCnt,
SHAPETYPE inType)

Analyzes giving stroke and returns type of a recognized geometrical shape. If no shape is recognized, it returns SHAPE_UNKNOWN. Possible values are:

```
typedef enum {
    SHAPE_UNKNOWN           = 0,
    SHAPE_TRIANGLE          = 0x0001,
    SHAPE_CIRCLE            = 0x0002,
    SHAPE_ELLIPSE           = 0x0004,
    SHAPE_RECTANGLE         = 0x0008,
    SHAPE_LINE              = 0x0010,
    SHAPE_ARROW             = 0x0020,
    SHAPE_SCRATCH           = 0x0040,
    SHAPE_ALL               = 0x00FF
} SHAPETYPE;
```

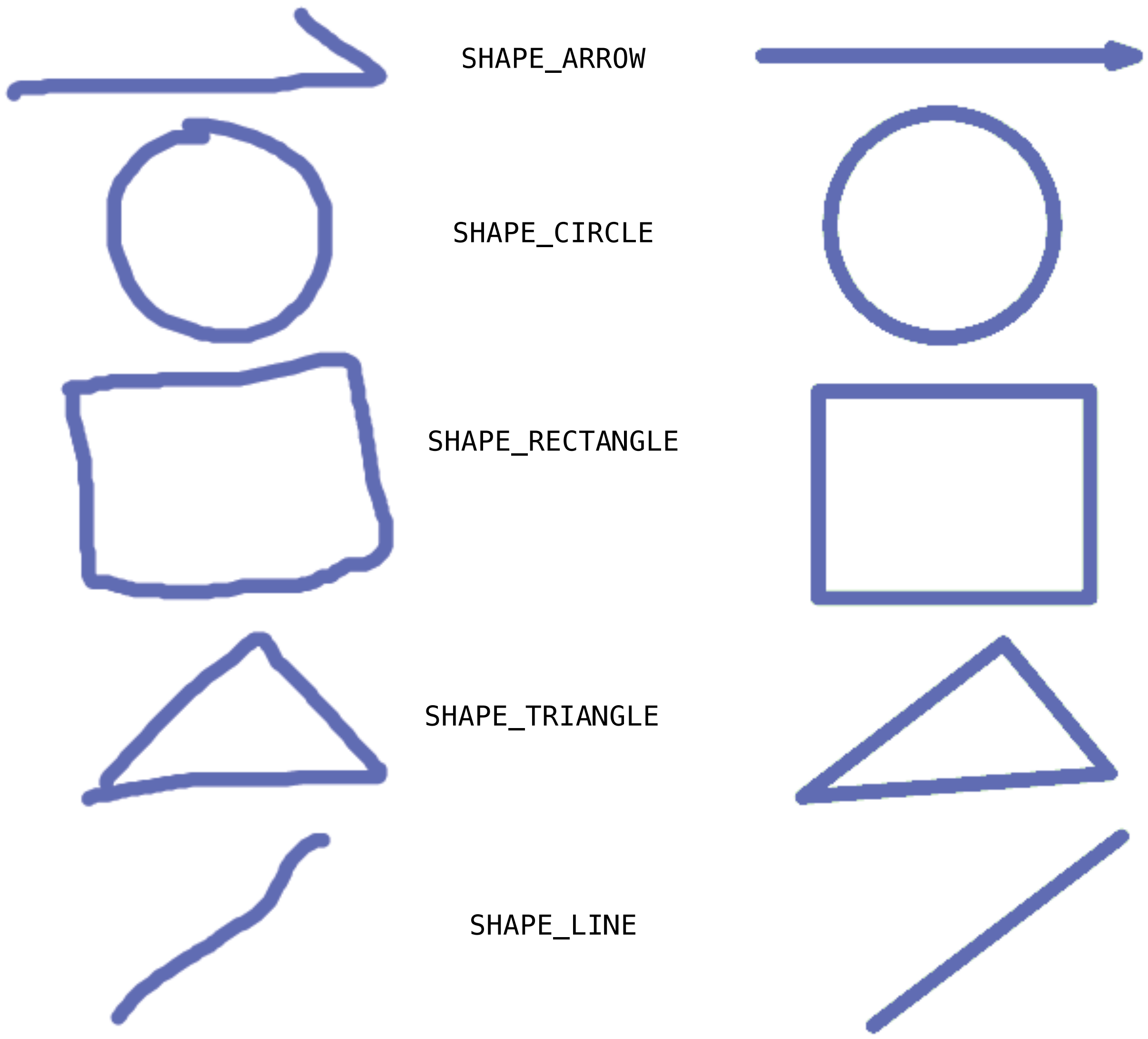
Parameters

CGStroke * pStroke	Pointer to stroke. If a geometrical shape is recognized, returns new array of pixels representing recognized shape.
int nStrokeCnt	Number of pixels in stroke.
SHAPETYPE inType	List of recognized geometrical shape or SHAPE_ALL to recognize all shapes.

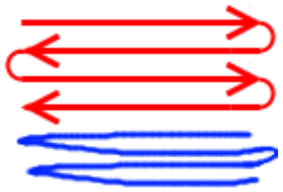
Returns

Returns type of a recognized geometrical shape, or SHAPE_UNKNOWN if no shape is recognized.

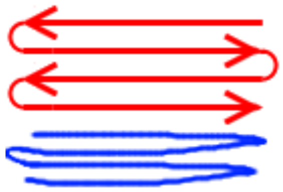
Supported Geometrical Shapes



- Also, the there is the special "scratch" SHAPE_SCRATCH shape which can be used as an erase gesture:



or



BOOL INK_IsStrokeSelected(INK_DATA_PTR pData, int nStroke)

Returns TRUE if the specified stroke is selected, otherwise returns FALSE.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index.

Returns

TRUE if the specified stroke is selected.

BOOL INK_Serialize(INK_DATA_PTR pData,
BOOL bWrite,
FILE * pFile,
void ** ppData,
int * pcbSize)

Writes (reads) compressed IDO data from (to) a file or a memory buffer.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData
BOOL bWrite	If TRUE – stores IDO data in a file or a memory buffer; if FALSE – reads data from file or memory buffer.
FILE * pFile	Pointer to a FILE. If this parameter is NULL, data is written (read) from the memory buffer.
void ** ppData	Double-pointer to a memory buffer. This parameter is ignored if pFile is not NULL.
int * pcbSize	Pointer to a variable containing size of the memory buffer. This parameter is ignored if pFile is not NULL.

Returns

TRUE if serialization was successful.

BOOL INK_Copy(INK_DATA_PTR pData,
const void ** ppRawData,
UInt32 * pcbSize)

Copies the raw (uncompressed) IDO content into the data buffer. The memory allocated for the buffer must be released using the free() function when it is no longer needed.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
const void ** ppRawData	Returns the raw (uncompressed) ink data. The memory allocated for the buffer must be released using the free() function.
UInt32 * pcbSize	Returns the pointer to the buffer size.

Returns

TRUE if operation was successful.

BOOL INK_Paste(INK_DATA_PTR pData,
const void * pRawData,
UInt32 cbSize,
CGPoint atPosition)

Copies ink from the raw data buffer (it can be created using the INK_Copy function) with the specified offset. This function does not delete the current IDO content.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
const void * pRawData	Contains the raw (uncompressed) ink data. It can be created using the INK_Copy function.
UInt32 cbSize	Size of the buffer.
CGPoint atPosition	Specifies offset for all strokes in the buffer from the {0,0} coordinate.

Returns

TRUE if operation was successful.

void INK_EnableShapeRecognition (INK_DATA_PTR pData, BOOL bEnable)

Enables (disables) recognition of basic geometrical shapes, such as a line, an arrow, a circle, a triangle, a square, and a diamond. The shape must be drawn using a single stroke. If shape recognition is enabled, the original stroke is replaced with the recognized shape automatically when the INK_AddStroke function is called.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
BOOL bEnable	Set to TRUE to enable geometrical shapes recognition, FALSE to disable it.

Returns

None.

BOOL INK_IsShapeRecognitionEnabled (INK_DATA_PTR pData)

Returns TRUE if recognition of geometrical shapes is enabled, otherwise returns FALSE. Use the INK_EnableShapeRecognition function to enable/disable recognition of basic geometrical shapes.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData

Returns

Returns TRUE if recognition of geometrical shapes is enabled, otherwise returns FALSE.

BOOL INK_MoveStroke(INK_DATA_PTR pData,
int nStroke,
float xOffset,
float yOffset,
CGRect * pRect,
BOOL recordUndo)

Moves the specified stroke by a specified offset relative to the current position of the first pixel of the stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Index of the stroke to move.

float xOffset	Specifies the horizontal offset relative to the current position. The offset can be negative or positive.
float yOffset	Specifies the vertical offset relative to the current position. The offset can be negative or positive.
CGRect * pRect	Returns the rectangle that may need to be redrawn. This parameter can be NULL.
BOOL recordUndo	Set to TRUE if you want to record undo information, otherwise set to FALSE.

Returns

TRUE if the stroke has been successfully moved, FALSE otherwise.

```
BOOL INK_ResizeStroke(    INK_DATA_PTR pData,
                          int nStroke,
                          float x0,
                          float y0,
                          float scalex,
                          float scaley,
                          BOOL bReset,
                          CGRect * pRect,
                          BOOL recordUndo )
```

Moves the specified stroke by a specified offset relative to the current position of the first pixel of the stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Index of the stroke to move.
float x0	Specifies new horizontal offset coordinate of the resized stroke.
float y0	Specifies new vertical offset coordinate of the resized stroke.
float scalex	Specifies new horizontal scale.
float scaley	Specifies new vertical scale.
BOOL bReset	Currently unused.

CGRect * pRect	Returns the rectangle that may need to be redrawn. This parameter can be NULL.
BOOL recordUndo	Set to TRUE if you want to record undo information, otherwise set to FALSE.

Returns

TRUE if the stroke has been successfully resized, FALSE otherwise.

**int INK_SetStrokeWidthAndColor (INK_DATA_PTR pData,
int nStroke, COLORREF color, int nWidth)**

Changes the color and width of the selected stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Index of the stroke to change.
COLORREF color	New stroke color.
int nWidth	New stroke width.

Returns

Number of selected strokes. 0 if no strokes were selected; -1 in case of the error.

---

void INK_ChangeSelZOrder(INK_DATA_PTR pData, int iDepth, BOOL bFwd)

Changes index (Z-order) of the selected strokes by the specified offset. It is assumed that strokes are always drawn in the order of the stroke index value (from 0 to INK_StrokeCount()-1).

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int iDepth	Specifies the depth of the Z-order offset. This must be a positive number.
BOOL bFwd	If TRUE, increases the selected strokes index by iDepth value (moves strokes to front); if FALSE – decreases index (moves strokes back).

Returns

None.

int INK_GetStrokeZOrder(INK_DATA_PTR pData, int nStroke)

Returns 0-based index (Z-order) of the specified stroke. Returns -1 if the stroke Z-order is not set or if specified stroke index is incorrect.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Specifies the stroke index in the strokes list (not the same as Z-order).

Returns

Stroke Z-order index, or -1 if the stroke Z-order is not set or if specified stroke index is incorrect.

BOOL INK_SetStrokeZOrder(INK_DATA_PTR pData, int nStroke, int iZOrder)

Sets 0-based index (Z-order) for the specified stroke. Returns TRUE if new Z-order is set, or FALSE in case of the error.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Specifies the stroke index in the strokes list (not the same as Z-order).
int iZOrder	New Z-order for the specified stroke. Set to -1 to remove Z-order index.

Returns

TRUE if successful, FALSE if error.

int INK_FindStrokeByPoint(INK_DATA_PTR pData, CGPoint thePoint)

Returns 0-based index of the stroke (not to confuse with Z-order index) if it is passes through or near the specified point coordinates.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGPoint thePoint	Desired point coordinates.

Returns

0-based stroke index, or -1 in case of the error.

---

int INK_SelectStrokesInRect(INK_DATA_PTR pData, CGRect rect)

Marks strokes that contain any portion in the specified rectangle as selected. This function does not unselect any previously selected strokes. If needed, use the INK_SelectAllStrokes to deselect all strokes before calling this function.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGRect rect	Selection rectangle.

Returns

Number of selected strokes. 0 if no strokes were selected; -1 in case of the error.

**BOOL INK_CurveIntersectsStroke(INK_DATA_PTR pData,
int nStroke,
const CGStroke points,
int nPointCount)**

Returns TRUE of a given stroke intersects a stroke specified by index.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nStroke	Stroke index.
CGStrokg points	Stroke points.
int nPointCount	number of points in the stroke.

Returns

Returns TRUE if strokes intersect, otherwise FALSE.

**int INK_DeleteIntersectedStrokes(INK_DATA_PTR pData,
const CGStroke points,
int nPointCount)**

Deletes all strokes in the ink data object that intersect given stroke.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGStrokg points	Stroke points.
int nPointCount	number of points in the stroke.

Returns

Returns number of deleted strokes, 0, if no strokes were deleted, or -1 if ink data is empty or parameter is wrong.

---

int INK_AddImage(INK_DATA_PTR pData, ImageAttributes * pImage)

Adds a new image to the image array. Image is stored as a raw data and can be in any format (JPEG, PNG, BITMAP). The SDK does not manipulate image data and treats it simply as a memory buffer.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
ImageAttributes * pImage	Image attributes. See the *Image Attributes* section above for the description of structure parameters.

Returns

0-based index of the added image in the image array; -1 in case of error.

---

int INK_SetImage(INK_DATA_PTR pData, int nImageIndex, ImageAttributes * pImage)

Replaces image and its attributes at the specified index.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nImageIndex	0-bases image index in the image array.
ImageAttributes * pImage	Image attributes.

Returns

0-based index of the image in the image array; -1 in case of error.

**BOOL INK_SetImageUserData(INK_DATA_PTR pData,
int nImageIndex,
void * userData)**

Replaces a pointer to the user-defined data associated with the specified image. Use the INK_GetImage to retrieve userData. This parameter is not used by the SDK internally.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nImageIndex	0-bases image index in the image array.
void * userData	Pointer to the user-defined data. A user is responsible for any memory allocation/de-allocation associated with this parameter

Returns

TRUE if successful, FALSE if image index is out of range.

**BOOL INK_GetImage(INK_DATA_PTR pData,
int nImageIndex,
ImageAttributes * pImage)**

Returns image and its attributes. See the Image Attributes section above for the description of the ImageAttributes structure.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nImageIndex	0-bases image index in the image array.
ImageAttributes * pImage	Returns image attributes. Memory for the pImage structure must be allocated by a user before calling this function.

Returns

TRUE if successful, FALSE if image index is out of range.

**int INK_GetImageFromPoint(INK_DATA_PTR pData,
CGPoint point,
ImageAttributes * pImage)**

Returns image and its attributes if the specified point is within the image frame.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGPoint point	Point for which image is retrieved.
ImageAttributes * pImage	Returns image attributes. Memory for the pImage structure must be allocated by a user before calling this function.

Returns

0-based index of the image in the image array; -1 in case of error or if the image frame does not contain the point.

BOOL INK_DeleteImage(INK_DATA_PTR pData, int nImageIndex)

Deletes image from the image array and releases memory allocated for the image data.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nImageIndex	0-bases image index in the image array.

Returns

TRUE if successful, FALSE if image index is out of range.

BOOL INK_DeleteAllImages(INK_DATA_PTR pData)

Deletes all images and releases all associated memory.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

TRUE if one or more images were deleted, otherwise returns FALSE.

---

int INK_CountImages(INK_DATA_PTR pData)

Returns number of images stored in the image array.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

Number of images in the image array. 0 if empty.

BOOL INK_SetImageFrame(INK_DATA_PTR pData, int nImageIndex, CGRect frame)

Sets image frame. The image frame is usually specified in the screen coordinates from the top-left corner of the associated ink page.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nImageIndex	0-bases image index in the image array.
CGRect frame	New image frame in screen coordinates.

Returns

TRUE if successful, FALSE if image index is out of range.

BOOL INK_AddText(INK_DATA_PTR pData, TextAttributes * pText)

Adds a new text label to the labels array. Text is stored as a NULL-terminated string of 16-bit UNICODE characters.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
TextAttributes * pText	Text attributes. See the Text Attributes section above for the description of structure parameters.

Returns

TRUE if a new text label was added to the array, FALSE otherwise.

BOOL INK_SetText(INK_DATA_PTR pData, int nTextIndex, TextAttributes * pText)

Replaces text label at the specified index.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nTextIndex	0-bases text label index in the labels array.

TextAttributes * pText	Text attributes. See the Text Attributes section above for the description of structure parameters.

Returns

TRUE if successful, FALSE if label index is out of range.

BOOL INK_SetTextUserData(INK_DATA_PTR pData,
int nTextIndex,
void * userData)

Replaces a pointer to the user-defined data associated with the specified text label. Use the INK_GetText to retrieve userData. This parameter is not used by the SDK internally.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nTextIndex	0-bases text label index in the labels array.
void * userData	Pointer to the user-defined data. A user is responsible for any memory allocation/de-allocation associated with this parameter

Returns

TRUE if successful, FALSE if label index is out of range.

BOOL INK_GetText(INK_DATA_PTR pData,
int nTextIndex,
TextAttributes * pText)

Returns text and its attributes. See the *Text Attributes* section above for the description of the TextAttributes structure.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nTextIndex	0-bases text label index in the labels array.
TextAttributes * pText	Text attributes. Memory for the pText structure must be allocated by a user before calling this function.

Returns

TRUE if successful, FALSE if label index is out of range.

int INK_GetTextFromPoint(INK_DATA_PTR pData,
CGPoint point,
TextAttributes * pText)

Returns text attributes if the specified point is within the text label frame.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGPoint point	Point for which text label should be retrieved.
TextAttributes * pText	Text attributes. Memory for the pText structure must be allocated by a user before calling this function.

Returns

0-based index of the text label in the labels array; -1 in case of error or if the label frame does not contain the point.

BOOL INK_DeleteText(INK_DATA_PTR pData, int nTextIndex)

Deletes text label from the labels array and releases memory allocated text and its attributes.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
int nTextIndex	0-bases text label index in the labels array.

Returns

TRUE if successful, FALSE if label index is out of range.

BOOL INK_DeleteAllTexts(INK_DATA_PTR pData, BOOL bRecordUndo)

Deletes all text labels and releases all associated memory.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
BOOL bRecordUndo	If TRUE, text labels are stored in the undo buffer before being deleted.

Returns

TRUE if one or more text labels were deleted, otherwise returns FALSE.

---

int INK_CountTexts(INK_DATA_PTR pData)

Returns number of text labels stored in the labels array.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.

Returns

Number of text labels in the labels array. 0 if the array is empty.

BOOL INK_SetTextFrame(INK_DATA_PTR pData, int nTextIndex, CGRect frame)

Sets text label frame. The label frame is usually specified in the screen coordinates from the top-left corner of the associated ink page.

Parameters

INK_DATA_PTR pData	Pointer to IDO created by INK_InitData.
CGRect frame	New text label frame in screen coordinates.

Returns

TRUE if successful, FALSE if label index is out of range.

Handwriting Recognition Engine API

Data Types

Pointer to the Handwriting Recognition Engine (HRE) **RECOGNIZER_PTR**

Recognizer character **UCHR** is the same as *unichar* or *unsigned short*. The recognition engine returns all text results in the `UNICODE (UTF-16)` encoding.

Exception: dictionary and other user-modifiable data file names are in UTF-8 encoding.

Dictionary Types

Dictionary Types used by the HWR_SetDictionaryData, HWR_GetDictionaryData, HWR_GetDictionaryLength, and HWR_HasDictionaryChnaged.

```
enum  {
          kDictionaryType_Main,
          kDictionaryType_Alternative,
          kDictionaryType_User

};
```

Functions

```
RECOGNIZER_PTR HWR_InitRecognizer(    const char * inDictionaryMain,
                                      const char * inDictionaryCustom,
                                      const char * inLearner,
                                      const char * inAutoCorrect,
                                      int language,
                                      int *    flags )
```

Initializes Handwriting Recognition Engine and loads dictionary and analyzer data from files.

Parameters

const char * inDictionaryMain	Pointer to the string containing the main dictionary file name (usually included with application resource) (UTF-8).
const char * inDictionaryCustom	Pointer to the string containing the user dictionary file name (must be stored in the Documents or other user-specific folder) (UTF-8).
const char * inLearner,	Pointer to the string containing the auto learner file name (must be stored in the Documents or other user-specific folder) (UTF-8).
const char * inAutoCorrect.	Pointer to the string containing the auto corrector file name (must be stored in the Documents or other user-specific folder) (UTF-8).
int language	Specifies the handwriting recognition language, see HWR_GetLanguageID function for the list of supported languages.
int * flags	Returns one or more of the following flags: FLAG_MAINDICT , FLAG_USERDICT, FLAG_ANALYZER, FLAG_CORRECTOR. It

	can be used to check if specific dictionaries and features initialized correctly.

Returns

A pointer to the Handwriting Recognition Engine (HRE); or NULL in case of an error (usually means insufficient memory or invalid parameter).

RECOGNIZER_PTR HWR_InitRecognizerFromMemory(
const char * inDictionaryMain,
const char * inDictionaryCustom,
const char * inLearner,
const char * inAutoCorrect,
int language,
int * flags)

Initializes Handwriting Recognition Engine and loads dictionary and analyzer data from memory instead of files, allowing users store essential data without using file system.

Parameters

const char * inDictionaryMain	Pointer the raw main dictionary data in the correct WritePad format
const char * inDictionaryCustom	Pointer the raw user (custom) dictionary data in the correct WritePad format
const char * inLearner,	Pointer the raw statistical analyzer data in the correct WritePad format
const char * inAutoCorrect.	Pointer the raw autocorrector data in the correct WritePad format.
int language	Specifies the handwriting recognition language, see HWR_GetLanguageID function for the list of supported languages.
int * flags	Returns one or more of the following flags: FLAG_MAINDICT , FLAG_USERDICT, FLAG_ANALYZER, FLAG_CORRECTOR. It can be used to check if specific dictionaries and features initialized correctly.

Returns

A pointer to the Handwriting Recognition Engine (HRE); or NULL in case of an error (usually means insufficient memory or invalid parameter).

---

void HWR_FreeRecognizer(RECOGNIZER_PTR pRecognizer,
const char * inDictionaryCustom,
const char * inLearner,
const char * inAutoCorrect)

Releases Handwriting Recognition Engine and frees memory allocated for it.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inDictionaryCustom	Pointer to the string containing the user dictionary file name (must be stored in the Documents or other user-specific folder) (UTF-8).
const char * inLearner,	Pointer to the string containing the auto learner file name (must be stored in the Documents or other user-specific folder) (UTF-8).
const char * inAutoCorrect.	Pointer to the string containing the auto corrector file name (must be stored in the Documents or other user-specific folder) (UTF-8).

Returns

None.

BOOL HWR_RecognizerAddStroke(RECOGNIZER_PTR pRecognizer,
CGStroke pStroke,
int nStrokeCnt)

Adds a new stroke to the current recognition session.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
CGStroke pStroke	Pointer to the array of X and Y coordinates of each pixel in the stroke plus optional pressure.
int nStrokeCnt	Number of pixels in the array.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_Recognize(RECOGNIZER_PTR pRecognizer)

Processes all strokes added to the current recognition session by HWR_RecognizerAddStroke function and generates the result.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_Reset(RECOGNIZER_PTR pRecognizer)

Resets the recognizer, and releases memory allocated for the current recognition results (if any). This function should be called before each new recognition session.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

TRUE if successful, otherwise FALSE.

---

const UCHR * HWR_GetResult RECOGNIZER_PTR pRecognizer)

Returns the most probable recognition result. Call this function after HWR_Recognize. The function returns an internal string pointer, which is only valid until recognizer is reset or a new recognition session is started.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

Pointer to the string containing the list of characters representing the most probable recognition result. In case of an error, returns NULL.

```
const UCHR * HWR_RecognizeInkData( RECOGNIZER_PTR pRecognizer,
                                   INK_DATA_PTR pInkData,
                                   int  nDataLen,
                                   BOOL bAsync,
                                   BOOL bFlipY,
                                   BOOL bSort,
                                   BOOL bSelOnly )
```

Starts new recognition session and processes ink stored in the Ink Data Object. Returns the most probable recognition result. . The function returns an internal string pointer, which is only valid until recognizer is reset or a new recognition session is started.

It is recommended to call this function in a separate thread, because this function may take a long time to complete. You can interrupt the recognition process by calling HWR_StopAsyncReco from a different thread.

Note: Never call HWR_StopAsyncReco and HWR_RecognizeInkData in the same thread because it will cause mutex lock.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
INK_DATA_PTR pInkData	Pointer to the Ink Data Object (see the Ink Data Object API).
int nDataLen	Number of strokes in the pInkData object to recognize, or -1 to recognize all strokes.
BOOL bAsync	Set to TRUE if the HWR_InitRecognizer is called from a recognition thread and HWR_StopAsyncReco is used to terminate the recognition session. Otherwise, set to FALSE.
BOOL bFlipY	If TRUE, rotates ink 180 degrees.
BOOL bSort	If TRUE, sorts strokes left-to-write as they appear on screen. Works only if handwritten text is in the single line.
BOOL bSelOnly	If TRUE, only strokes that are marked as selected are recognized.

Returns

Pointer to the array of characters representing the most probable recognition result. In case of an error, returns NULL.

void HWR_StopAsyncReco (RECOGNIZER_PTR pRecognizer)

Stops recognition, if the ink is processed by the HWR_RecognizeInkData function, otherwise has no effect. Calling this function causes HWR_RecognizeInkData to return almost immediately, however, HWR_StopAsyncReco does not wait until HWR_RecognizeInkData returns.

Note: Never call HWR_StopAsyncReco and HWR_RecognizeInkData in the same thread because it will cause mutex lock.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

None.

BOOL HWR_RecognizeSymbol (RECOGNIZER_PTR pRecognizer,
INK_DATA_PTR pInkData,
int baseline,
int symbol_size)

Recognizes a single character, digit, or a punctuation symbol. Returns TRUE if a character is recognized, otherwise returns FALSE. To retrieve recognized symbol(s), use HWR_GetResultWord API.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
INK_DATA_PTR pInkData	Pointer to the Ink Data Object (see the Ink Data Object API).
int baseline	Character baseline, used to distinguish between upper and lower case.
int symbol_size	Maximum symbol size.

Returns

None.

BOOL HWR_PreRecognizeInkData (RECOGNIZER_PTR pRecognizer,
INK_DATA_PTR pInkData,
int nDataLen,
BOOL bFlipY)

Starts new recognition session and pre-processes ink stored in the Ink Data Object. This function is useful when pInkData object already contains data, but more ink data will be added to the same recognition session in the future. This function leaves the current recognition open for more data, you can use HWR_RecognizerAddStroke to add more data or HWR_Recognize and HWR_GetResult to obtain the result.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
INK_DATA_PTR pInkData	Pointer to the Ink Data Object (see the Ink Data Object API).
int nDataLen	Number of strokes in the pInkData object to recognize, or -1 to recognize all strokes.
BOOL bFlipY	If TRUE, rotates ink 180 degrees.

Returns

TRUE if success, or FALSE if ink data cannot be added to the current recognition session.

BOOL HWR_EnablePhatCalc (RECOGNIZER_PTR pRecognizer, BOOL bEnable)

Toggles the built-in calculator functionality. The calculator recognizes input like 34.6+45/7.3= and produces the result. It supports addition, division, multiplication, and subtraction of numbers with or without decimal points. The calculator is enabled by default.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
BOOL bEnable	If TRUE, enabled calculator functionality, if FALSE – disabled it.

Returns

TRUE if successful, otherwise FALSE.

USHORT HWR_GetResultWeight(RECOGNIZER_PTR pRecognizer,
int nWord,
int nAlternative)

Returns the probability of the recognized word for the specified row and column. Use the HWR_GetResultWordCount() and HWR_GetResultAlternativeCount() to get number of recognized words (rows) and number of alternatives for each word (columns).

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nWord	0-based word index (row).
int nAlternative	0-based alternative index (column).

Returns

Word probability value in percent. Probability cannot be lower than 51 (minimum allowed probability) or higher than 100.

const UCHR * HWR_GetResultWord(RECOGNIZER_PTR pRecognizer,
int nWord,
int nAlternative)

Returns the recognized word for the specified row and column. Use the HWR_GetResultWordCount() and HWR_GetResultAlternativeCount() to get number of recognized words (rows) and number of alternatives for each word (columns). The function returns an internal string pointer, which is only valid until recognizer is reset or a new recognition session is started.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nWord	0-based word index (row).
int nAlternative	0-based alternative index (column).

Returns

Pointer to the string containing the list of characters representing the most probable recognition result. In case of an error, returns NULL.

---

int HWR_GetResultWordCount(RECOGNIZER_PTR pRecognizer)

Returns number of words in the current recognition result. The recognition result is stored in the memory after each recognition session until HWR_Reset() or HWR_RecognizeInkData() is called.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

Number of words in the current recognition result.

---

int HWR_GetResultAlternativeCount(RECOGNIZER_PTR pRecognizer, int nWord)

Returns number of alternatives for the specified word in the current recognition result. The recognition result is stored in the memory after each recognition session until HWR_Reset() or HWR_RecognizeInkData() is called.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nWord	0-based word index

Returns

Number of alternatives for the specified word in the current recognition result.

---

int HWR_GetResultWordCount(RECOGNIZER_PTR pRecognizer)

Returns number of words in the current recognition result. The recognition result is stored in the memory after each recognition session until HWR_Reset() or HWR_RecognizeInkData() is called.

Parameters

RECOGNIZER_PTR pRecognizer — Pointer to the HRE returned by HWR_InitRecognizer.

Returns

Number of words in the current recognition result.

int HWR_GetResultStrokeNumber(RECOGNIZER_PTR pRecognizer,
int nWord,
int nAlternative)

Returns the index of the last stroke for the specified word and alternative in the current recognition result. Currently this function returns the same index for any alternative for the specified word.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nWord	0-based word index (row)
int nAlternative	0-based alternative index (column).

Returns

Index of the last stroke for the specified word and alternative in the current recognition result.

int HWR_SetRecognitionMode (RECOGNIZER_PTR pRecognizer, int nMode)

Sets current recognition mode, possible modes are (defined in the RecognizerAPI.h):

RECMODE_GENERAL	Normal recognition -- all symbols allowed
RECMODE_CAPS	All recognized text converted to capitals
RECMODE_NUM	Numeric and punctuation recognition mode
RECMODE_WWW	Internet address mode (no spaces, special dictionary)
RECMODE_NUMBERSPURE	pure numeric mode: recognizes 0123456789 only
RECMODE_CUSTOM	custom charset for numbers and punctuation, no alpha
RECMODE_ALPHAONLY	Alpha characters only, no punctuation or numbers

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer
int nMode	Recognition mode (see above for possible values).

Returns

Previous recognition mode.

int HWR_GetRecognitionMode (RECOGNIZER_PTR pRecognizer)

Returns the current recognition mode, possible modes are (defined in the RecognizerAPI.h):

RECMODE_GENERAL	Normal recognition -- all symbols allowed
RECMODE_CAPS	All recognized text converted to capitals
RECMODE_NUM	Numeric and punctuation recognition mode
RECMODE_WWW	Internet address mode (no spaces, special dictionary)
RECMODE_NUMBERSPURE	pure numeric mode: recognizes 0123456789 only
RECMODE_CUSTOM	custom charset for numbers and punctuation, no alpha
RECMODE_ALPHAONLY	Alpha characters only, no punctuation or numbers

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

Current recognition mode.

void HWR_SetCustomCharset(RECOGNIZER_PTR pRecognizer, const UCHR * pCustomNum, const UCHR * pCustPunct)

If you set RECMODE_CUSTOM recognition mode, use this function to set custom numeric and punctuation characters. Only specified custom character will be recognized. No alpha characters allowed. Use c RECMODE_ALPHAONLY mode instead.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pCustomNum	String containing custom numeric characters, for example “1245”.
const UCHR * pCustPunct	String containing custom punctuation characters, or example: “,.!?”.

Returns

None.

unsigned int HWR_GetRecognitionFlags(RECOGNIZER_PTR pRecognizer
unsigned int flags)

Returns the current recognition flags, possible values are:

FLAG_SEPLET	Separate letters mode.
FLAG_USERDICT	User dictionary enabled (default).
FLAG_MAINDICT	Main dictionary enabled (default).
FLAG_ONLYDICT	Recognizes only dictionary words.
FLAG_STATICSEGMENT	Static word segmentation.
FLAG_SINGLEWORDONLY	Disable word segmentation.
FLAG_INTERNATIONAL	Support international characters.
FLAG_SUGGESTONLYDICT	Suggests only dictionary words.
FLAG_ANALYZER	Enable statistical analyzer (default).
FLAG_CORRECTOR	Enable autocorrector (default).
FLAG_SPELLIGNORENUM	Ignore words with number when spelling.
FLAG_SPELLIGNOREUPPER	Ignore words in upper case when spelling.
FLAG_NOSINGLELETSPACE	Do not add space after single letter.
FLAG_ENABLECALC	Enable calculator.
FLAG_ALTDICT	Use alternative dictionary for recognition.
FLAG_NOSPACE	Do not add space at the end of the result.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
unsigned int flags	Combination of flags to set. Old flags will be overwritten.

Returns

Previous recognition flags (see above for possible values).

int HWR_SpellCheckWord(RECOGNIZER_PTR pRecognizer,
const UCHR * pszWord,
UCHR * pszAlternatives,
int cbSize,
int nFlags)

Checks spelling of the specified word and returns possible alternatives. Depending on the option, can also return list of words containing the specified word. Possible flags are:

HW_SPELL_CHECK	Spell check word.
HW_SPELL_LIST	Get word list.
HW_SPELL_USEALTDICT	Use alternative dictionary instead of main dictionary.
HW_SPELL_IGNORENUM	Ignore words containing numbers.

HW_SPELL_IGNOREUPPER	Ignore words in UPPERCASE.
HW_SPELL_USERDICT	Use user dictionary .

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pszWord	Pointer to a string containing the word to spell check.
UCHR * pszAlternatives	Pre-allocated buffer for alternatives. Alternatives will be separated with `PM_ALTSEP`.
int cbSize	Length of the alternatives buffer in characters.
int nFlags	Spell checker flags (see above).

Returns

Returns number of possible alternatives.

BOOL HWR_AddUserWordToDict(RECOGNIZER_PTR pRecognizer, const UCHR * pszWord)

Adds a new word to the user dictionary.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pszWord	Pointer to a string containing the new word.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_IsWordInDict(RECOGNIZER_PTR pRecognizer, const UCHR * pszWord)

Verifies if the specified word exist in the dictionary.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

const UCHR * pszWord	Pointer to a string containing the word to check.

Returns

TRUE if word is in the user or main dictionary, otherwise FALSE.

BOOL HWR_LoadAlternativeDict(RECOGNIZER_PTR pRecognizer, const UCHR * inDictionaryAlt)

Loads alternative dictionary that can be used instead of main dictionary for character recognition and spell checking.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pszWord	Pointer to a string containing the path name of the dictionary file

Returns

TRUE if dictionary is successfully loaded, otherwise FALSE.

---

int HWR_EnumUserWords(RECOGNIZER_PTR pRecognizer, PRECO_ONGOTWORD callback, void * pParam)

Uses user-defined callback function to enumerate words in the user dictionary. The callback function prototype:

```
int GetWordList( const char * szWord, void * pParam )
```

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
PRECO_ONGOTWORD callback	The user-defined callback function called for each word in the user dictionary. First parameter is the word, second is user-defined parameter.
void * pParam	The user-defined parameter which is passed in the callback function. Can be NULL.

Returns

Number of words in the user dictionary.

BOOL HWR_NewUserDict(RECOGNIZER_PTR pRecognizer)

Creates a new user dictionary. If another user dictionary is loaded, it is released.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_SaveUserDict(RECOGNIZER_PTR pRecognizer, const char * inDictionaryCustom)

Saves the current user dictionary in a file.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inDictionaryCustom	Pointer to a string containing the file name for the user dictionary, must be in UTF-8 encoding.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_SaveWordList(RECOGNIZER_PTR pRecognizer, const char * inWordListFile)

Saves the current Autocorrector word list in a file.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inWordListFile	Pointer to a string containing the file name for the user dictionary, must be in UTF-8 encoding.

Returns

TRUE if successful, otherwise FALSE.

int HWR_EnumWordList(RECOGNIZER_PTR pRecognizer, RECO_ONGOTWORDLIST callback, void * pParam)

Uses user-defined callback function to enumerate words in the Autocorrector word list. The callback function prototype:

```
int GetWordList(     const UCHR * szWordFrom,
                     const UCHR * szWordTo,
                     unsigned int  nFlags,
                     void * pParam)
```

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
RECO_ONGOTWORDLIST callback	User-defined callback function called for each pair of words in the Autocorrector list. First parameter is the word to change from, second is word to change to, third contain flags, and the last is the user-defined parameter.
void * pParam	User-defined parameter which is passed in the callback function.

Returns

Number of words in the Autocorrector word list.

BOOL HWR_EmptyWordList (RECOGNIZER_PTR pRecognizer)

Removes all entries from the Autocorrector word list and releases associated memory.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_AddWordToWordList(RECOGNIZER_PTR pRecognizer,
const UCHR * pszWord1,
const UCHR * pszWord2,
int nFlags,
BOOL bReplace)

Adds a new pair of words to the Autocorrector word list.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pszWord1	Pointer to a string containing the word which will be replaced
const UCHR * pszWord2	Pointer to a string containing the word which will be replaced with.
int nFlags	Autocorrector flags. Possible values are: `WCF_IGNORECASE`, `WCF_ALWAYS`, `WCF_DISABLED`.
BOOL bReplace	TRUE to replace existing word pair with a new one.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_LearnNewWord(RECOGNIZER_PTR pRecognizer,
const UCHR * pszWord,
USHORT nWeight)

Adds a new word to the statistical analyzer.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pszWord	Pointer to a string containing the new word to learn.
USHORT nWeight	Recognizer of the word returned by recognizer, or calculated probability in percent. If unknown, use 0.

Returns

TRUE if successful, otherwise FALSE.

**BOOL HWR_AnalyzeWordList(RECOGNIZER_PTR pRecognizer,
const UCHR *pszWordList,
UCHR *pszResult)**

Changes order of words in the specified word list based on the current statistical analyzer data. Returns resorted word list. You must allocate a buffer for pszResult before calling this function. The size of the pszResult buffer must be equal or greater than pszWordList size.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer
const UCHR * pszWordList	Pointer to a string containing the word list. Words must be separated by SP_ALTSEP character.
UCHR * pszResult	Resorted word list. Depending on the statistical analyzer data, the result might be the same as the original word list.

Returns

TRUE if successful, otherwise FALSE.

**BOOL HWR_ReplaceWord(RECOGNIZER_PTR pRecognizer,
const UCHR * pszWord1,
USHORT nWeight1,
const UCHR * pszWord2,
USHORT nWeight2)**

Adds a new pair of word replacements to the statistical analyzer. This function is usually called by the UI when a user replaces an incorrectly recognized word with a correct one.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const UCHR * pszWord1	Pointer to a string containing the incorrectly recognized word.
USHORT nWeight1	Recognition probability of the first word, can be 0 if unknown.
const UCHR * pszWord2	Pointer to a string containing the correct word.

USHORT nWeight2	Recognition probability of the second word, can be 0 if unknown.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_SaveLearner(RECOGNIZER_PTR pRecognizer, const char * pszFileName)

Saves the current statistical analyzer database.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * pszFileName	Pointer to a string containing the file name for the statistical analyzer data, must be in UTF-8 encoding.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_ResetUserDict(RECOGNIZER_PTR pRecognizer, const char * inDictionaryCustom)

Removes all words from the current user dictionary. If the file name is specified, creates the default user dictionary and saves it.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inDictionaryCustom	Pointer to a string containing the file name for the user dictionary; must be in UTF-8 encoding. Can be NULL.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_ResetAutocorrector(RECOGNIZER_PTR pRecognizer, const char * inWordListFile)

Removes all words from the current Autocorrector word list. If the file name is not NULL creates the default Autocorrector word list and saves it.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inWordListFile	Pointer to a string containing the file name for the word list, must be in UTF-8 encoding. Can be NULL.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_ResetLearner(RECOGNIZER_PTR pRecognizer,
const char * inLearnerFile)

Removes statistical analyzer data. If the file name is specified the file is also removed.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inLearnerFile	Pointer to a string containing the file name for the statistical analyzer, must be in UTF-8 encoding. Can be NULL.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_ImportWordList(RECOGNIZER_PTR pRecognizer,
const char * inImportFile)

Imports a new Autocorrector word list from a CSV (comma separated values) file. The file must contain text in the `Windows CP1252` encoding.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inImportFile	Pointer to a string containing the file name for the CSV file; must be in UTF-8 encoding. Cannot be NULL.

Returns

TRUE if successful, otherwise FALSE.

**BOOL HWR_ImportUserDictionary(RECOGNIZER_PTR pRecognizer,
const char * inImportFile)**

Imports a new user dictionary word list from a text file. The file must contain text in the `Windows CP1252` encoding, a single word per line with no spaces.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer
const char * inImportFile	Pointer to a string containing the file name for the text file; must be in UTF-8 encoding. Cannot be NULL.

Returns

TRUE if successful, otherwise FALSE.

**BOOL HWR_ExportWordList(RECOGNIZER_PTR pRecognizer,
const char * inExportFile)**

Exports the current Autocorrector word list as a CSV (comma separated values) file. The resulting file is in the `Windows CP1252` encoding.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * inExportFile	Pointer to a string containing the file name for the CSV file; must be in UTF-8 encoding. Cannot be NULL.

Returns

TRUE if successful, otherwise FALSE.

**BOOL HWR_ExportUserDictionary(RECOGNIZER_PTR pRecognizer,
const char * inExportFile)**

Exports the current user dictionary as a text file. The resulting file is in the `Windows CP1252` encoding.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

const char * inExportFile	Pointer to a string containing the file name for the user dictionary file; must be in UTF-8 encoding. Cannot be NULL.

Returns

TRUE if successful, otherwise FALSE.

BOOL HWR_SetDictionaryData(RECOGNIZER_PTR pRecognizer,
const char * pData,
int nDictType)

Loads dictionary from memory instead of a file.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const char * pData	Pointer to a buffer containing the dictionary data in the correct WritePad format.
int nDictionaryType	Dictionary Type: kDictionaryType_Main, kDictionaryType_Alternative, or kDictionaryType_User.

Returns

TRUE if successful, otherwise FALSE.

---

int HWR_GetDictionaryData(RECOGNIZER_PTR pRecognizer,
char ** pData,
int nDictType)

Returns pointer to the binary dictionary data in WritePad format. This pointer may be used in HWR_SetDictionaryData. The function allocates memory for dictionary data using malloc(). A user must use free() function to release memory when raw dictionary data is no longer needed.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
char ** pData	Pointer to a buffer containing the specified dictionary data in the WritePad format.

int nDictionaryType	Dictionary Type: kDictionaryType_Main, kDictionaryType_Alternative, or kDictionaryType_User.

Returns

Positive integer specifying the size of pData memory buffer;
0, if dictionary does not exist or empty;
-1 in case of the error.

---

int HWR_GetLanguageID(RECOGNIZER_PTR pRecognizer)

Language ID of the specified instance of the handwriting recognition engine. Language IDs are defined in RecoDefs.h file. The possible values are:

LANGUAGE_NONE	0	International
LANGUAGE_ENGLISH	1	English
LANGUAGE_FRENCH	2	French
LANGUAGE_GERMAN	3	German
LANGUAGE_SPANISH	4	Spanish
LANGUAGE_ITALIAN	5	Italian
LANGUAGE_SWEDISH	6	Swedish
LANGUAGE_NORWEGIAN	7	Norwegian
LANGUAGE_DUTCH	8	Dutch
LANGUAGE_DANISH	9	Danish
LANGUAGE_PORTUGUESE	10	Portuguese (Portugal)
LANGUAGE_PORTUGUESEB	11	Portuguese (Brazil)
LANGUAGE_FINNISH	13	Finnish
LANGUAGE_INDONESIAN	14	Indonesian

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

Language ID of the specified instance of the handwriting recognition engine.

const char * HWR_GetLanguageName(RECOGNIZER_PTR pRecognizer)

Returns language name (in English) of the specified instance of the handwriting recognition engine.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

English name of the specified instance of the handwriting recognition engine.

int HWR_GetSupportedLanguages(int ** languages)

Returns list and number of languages supported by the handwriting recognition library.

Parameters

int ** languages	Contains array of language IDs, see HWR_GetLanguageID function for possible values. Note that this is a pointer to a static array, do not attempt to modify or free memory.

Returns

Number of elements in the *languages* array

BOOL HWR_IsLanguageSupported(int languageID)

Returns TRUE if the handwriting recognition library supports the specified language.

Parameters

int language	Language ID, see HWR_GetLanguageID function for possible values.

Returns

TRUE if the handwriting recognition library supports the specified language.

BOOL HWR_HasDictionaryChanged (RECOGNIZER_PTR pRecognizer, int nDictType)

Returns TRUE if the specified dictionary has changed.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nDictionaryType	Dictionary Type: kDictionaryType_Main, kDictionaryType_Alternative, or kDictionaryType_User.

Returns

TRUE if the specified dictionary has changed, otherwise FALSE.

BOOL HWR_HasDictionaryChanged(RECOGNIZER_PTR pRecognizer, int nDictType)

Returns TRUE if the specified dictionary has changed.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nDictionaryType	Dictionary Type: kDictionaryType_Main, kDictionaryType_Alternative, or kDictionaryType_User.

Returns

TRUE if the specified dictionary has changed, otherwise FALSE.

BOOL HWR_GetDictionaryLenght(RECOGNIZER_PTR pRecognizer, int nDictType)

Returns size of the raw dictionary data (in bytes).

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
int nDictionaryType	Dictionary Type: kDictionaryType_Main, kDictionaryType_Alternative, or kDictionaryType_User.

Returns

Positive integer specifying the size of raw dictionary data;
0, if dictionary does not exist or empty;
-1 in case of the error.

BOOL HWR_SetDefaultShapes(RECOGNIZER_PTR pRecognizer)

Restores default configuration for handwritten letter shapes.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

TRUE if handwritten letter shapes were successfully reset to the default configuration, otherwise FALSE.

BOOL HWR_SetLetterShapes(RECOGNIZER_PTR pRecognizer, const unsigned char * pShapes)

Sets the letter shapes configuration.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.
const unsigned char * pShapes	Pointer to a memory buffer containing letter shapes configuration.

Returns

TRUE if letter shapes configuration was successfully set, otherwise FALSE.

const unsigned char * HWR_SetLetterShapes(RECOGNIZER_PTR pRecognizer)

Returns the current letter shapes configuration.

Parameters

RECOGNIZER_PTR pRecognizer	Pointer to the HRE returned by HWR_InitRecognizer.

Returns

Pointer to a memory buffer containing letter shapes configuration, or NULL in case of the error.

GESTURE_TYPE HWR_CheckGesture(GESTURE_TYPE gtCheck,
CGStroke stroke,
int nPoints,
int nScale,
int nMinLen)

Check if the given stroke is a gesture. The function returns the type of the recognized gesture, or GEST_NONE if no gesture is recognized. Possible values are:

```
typedef  enum  {
    GEST_NONE      = 0x00000000,
    GEST_DELETE    = 0x00000001,
    GEST_SCROLLUP  = 0x00000002,
    GEST_BACK      = 0x00000004,
    GEST_SPACE     = 0x00000008,
    GEST_RETURN    = 0x00000010,
    GEST_CORRECT   = 0x00000020,
    GEST_SPELL     = 0x00000040,
    GEST_SELECTALL = 0x00000080,
    GEST_UNDO      = 0x00000100,
    GEST_SMALLPT   = 0x00000200,
    GEST_COPY      = 0x00000400,
    GEST_CUT       = 0x00000800,
    GEST_PASTE     = 0x00001000,
    GEST_TAB       = 0x00002000,
    GEST_MENU      = 0x00004000,
    GEST_LOOP      = 0x00008000,
    GEST_REDO       = 0x00010000,
    GEST_SCROLLDN  = 0x00020000,
    GEST_SAVE       = 0x00040000,
    GEST_SENDMAIL  = 0x00080000,
    GEST_OPTIONS   = 0x00100000,
    GEST_SENDTODEVICE = 0x00200000,
    GEST_BACK_LONG = 0x00400000,

    GEST_ALL       = 0x0FFFFFFF
} GESTURE_TYPE, *pGESTURE_TYPE;
```

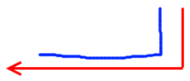
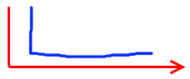
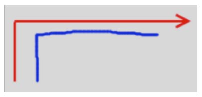
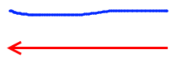
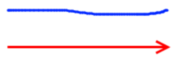
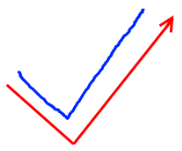
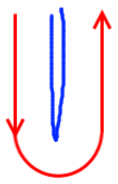
Parameters

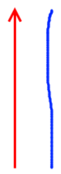
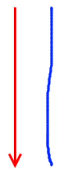
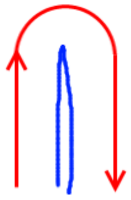
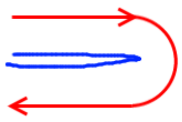
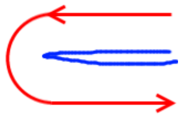
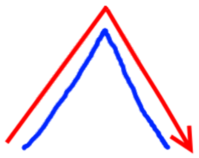
GESTURE_TYPE gtCheck	Mask of the gesture(s) to be recognized. Use GEST_ALL to check for all gestures.
CGStroke pStroke	Stroke points.
int nPoints	Number of pixels in the stroke.
int nScale	Scale factor, usually 1.
int nMinLen	Minimum length of the back space stroke, in pixels.

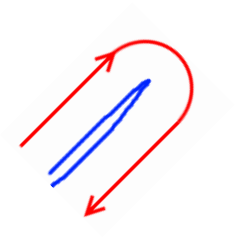
Returns

Type of the recognized gesture, or GEST_NONE.

List of standard gestures

	**Return**: Similar to pressing enter on the keyboard. GEST_RETURN
	**Space**: Inserts a space. GEST_SPACE
	**Tab**: Inserts a tabulation character. GEST_TAB
	**Backspace**: Removes a character to the left of cursor or the selected text. To perform the gesture, draw a horizontal line from right to left. To avoid interference with handwriting this gesture needs a length of 150 or more pixels. The minimum gesture length can be specified as a function parameter. GEST_BACK
	**Delete**: Removes a character to the right of the cursor or the selected text. To avoid interference with handwriting this gesture needs a length of 150 or more pixels. The minimum gesture length can be specified as a function parameter. GEST_DELETE
	**Spell Check**: If a single word is selected this gesture opens the spell checker window for the selected word with possible alternatives, otherwise brings up the **Last Recognition Alternatives** window containing multiple suggestions for each written word. GEST_CORRECT
	**Keyboard**: Opens the **Punctuation Keyboard** containing hard-to-write punctuation characters. Same as the **Keyboard** button. If no punctuation keyboard is implemented, may be used as **Redo**. GEST_MENU

	**Scroll Up**: Scrolls the content of the edit window up. To avoid interference with handwriting this gesture needs a length of 150 or more pixels. The minimum gesture length can be specified as a function parameter. `GEST_SCROLLUP`
	**Scroll Down**: Scrolls the content of the edit window down. To avoid interference with handwriting this gesture needs a length of 150 or more pixels. The minimum gesture length can be specified as a function parameter. `GEST_SCROLLDN`
	**Undo**: Undoes the last editing operation. `GEST_UNDO`
	**Copy**: Copies the selected block of text to the clipboard. `GEST_COPY`
	**Cut**: Copies the selected block of text to the clipboard and deletes it. `GEST_CUT`
	**Paste**: Inserts the clipboard content at the current cursor location. `GEST_PASTE`

	**Select All**: Selects all text in the editor. `GEST_SELECTALL`

Code Samples (Objective C)

Listing 1 – Using HWR_RecognizerAddStroke & HWR_Recognize

Recognize ink using HWR_RecognizerAddStroke/HWR_Recognize functions.

…………………………

```
- (const char *) recognizeInk1
{
     const char * pText = NULL;

     HWR_Reset( _recognizer );

     for ( int i = 0; i < STROKE_CNT; i++ )
     {
       CGStroke ptStroke = aStrokes[i].stroke;
       HWR_RecognizerAddStroke( _recognizer,
                     ptStroke, aStrokes[i].length );
     }

     if ( HWR_Recognize( _recognizer ) )
     {
       pText = HWR_GetResult( _recognizer );
       if ( pText == NULL || *pText == 0 )
       {
              return "*Error*";
       }

       NSMutableString * strResult =
       [[NSMutableString alloc] initWithCString:pText
                              encoding:NSUnicodeStringEncoding];


       NSComparisonResult comp = [strResult
                         compare:kEmptyWord
                         options:NSCaseInsensitiveSearch
                         range:NSMakeRange( 0, 5 )];
       if ( NSOrderedSame == comp )
       {
              return "*Error*";
       }

       // TODO: process the result

       [strResult release];

     }
     return pText;
}
```

Listing 2 – Using HWR_RecognizeInkData

Recognize ink using the HWR_RecognizeInkData function.

……………………….

```
- (const char *) recognizeInk2
{
    const char * pText = NULL;

    HWR_Reset( _recognizer );
    g_bRunRecognizer = TRUE;     // you can call HWR_RecognizeInkData
                                 // in thread, setting g_bRunRecognizer
                                 // to FALSE will terminate the
                                 // recognition session
    pText = HWR_RecognizeInkData( _recognizer, inkData, FALSE );

    if ( pText == NULL || *pText == 0 )
    {
      return "*Error*";
    }


    NSMutableString * strResult =
    [[NSMutableString alloc] initWithCString:pText
                                  encoding:NSUnicodeStringEncoding];


    NSComparisonResult comp = [strResult
                          compare:kEmptyWord
                          options:NSCaseInsensitiveSearch
                          range:NSMakeRange( 0, 5 )];
    if ( NSOrderedSame == comp )
    {
      return "*Error*";
    }

    // TODO: process the result

    [strResult release];

    return pText;
}
```

Listing 3 – Enumerating Recognition results

Enumerating multiple recognition results

……………………….

```
NSMutableArray * arrWords = [[NSMutableArray alloc] init];
NSString *          word;
// get multiple suggestions for each word
```

```
int wordCnt = HWR_GetResultWordCount( _recognizer );
for ( int i = 0; i < wordCnt; i++ )
{
     int nAltCnt = HWR_GetResultAlternativeCount( _recognizer, i );
     for ( int j = 0; j < nAltCnt; j++ )
     {
        const char * chrWord = HWR_GetResultWord( _recognizer, i, j );
        if ( ! HWR_IsWordInDict( _recognizer, chrWord ) )
        {
                // TODO: process if needed...
                // for example, spell check the word
                char  *       pWordList = malloc( MAX_STRING_BUFFER );
                int           flags = HW_SPELL_CHECK | HW_SPELL_USERDICT;
                if ( HWR_SpellCheckWord( _recognizer, chrWord,
                              pWordList, MAX_STRING_BUFFER-1, flags ) == 0 )
                {
                        for ( int n = 0;
                              0 != pWordList[n] && n < MAX_STRING_BUFFER;
                              n++ )
                        {
                              if ( pWordList[n] == PM_ALTSEP )
                                      pWordList[n] = 0;
                        }
                        for ( int k = 0; k < MAX_STRING_BUFFER; k++ )
                        {
                              word = [[NSString alloc]
                                              initWithCString:&pWordList[k]
                                              encoding:NSUnicodeStringEncoding];

                              [arrWords addObject:word];
                              [word release];

                              while ( 0 != pWordList[k] )
                                      k++;
                              if ( 0 == pWordList[k+1] )
                                      break;
                        }
                }
                free( (void *)pWordList );
        }
        else
        {
                // TODO: in this sample we add only dictionary words
                word = [[NSString alloc]
                              initWithCString:chrWord
                              encoding:NSInicodeStringEncoding];
                [arrWords addObject:word];
                [word release];
                // TODO: process recognition probability, if needed
                // USHORT weight=HWR_GetResultWeight( _recognizer, i, j );
        }
        // must free memory allocated for a word
        free( (void *)chrWord );
```

```
    }
}

// TODO: show the word list in the debugger
for ( int i = 0; i < [arrWords count]; i++ )
{
    NSLog( @"%@", [arrWords objectAtIndex:i] );
}
```

Listing 4 – Initializing Recognition Engine

Enabling/Disabling Handwriting Recognition Engine

```
…………………………

// geberate user dictionary name
NSBundle* bundle = [NSBundle mainBundle];
NSArray * paths = NSSearchPathForDirectoriesInDomains(
                                NSDocumentDirectory,
                                NSUserDomainMask,
                                YES);
NSString * strUserFile = [[paths objectAtIndex:0]
                   stringByAppendingPathComponent:USER_DICTIONARY];
NSString * strLearner =  [[paths objectAtIndex:0]
                   stringByAppendingPathComponent:USER_STATISTICS];
NSString * strCorrector =  [[paths objectAtIndex:0]
                   stringByAppendingPathComponent:USER_CORRECTOR];

if ( bEnableReco )
{
     if ( NULL != _recognizer )
     {
       return HWR_Reset( _recognizer );
     }
     else
     {
       _recognizer = HWR_InitRecognizer(
                                [[bundle pathForResource:DEFAULT_DICTIONARY
                                               ofType:@"dct"] UTF8String],
                                 [strUserFile UTF8String],
                                 [strLearner UTF8String],
                                 [strCorrector UTF8String],
                                 LANGUAGE_ENGLISH, NULL );

       if ( NULL != _recognizer )
       {
             NSUserDefaults* defaults = [NSUserDefaults standardUserDefaults];
             NSData * data = [defaults dataForKey:kRecoOptionsLetterShapes];
             if ( [data length] > 0 )
                   HWR_SetLetterShapes( _recognizer, [data bytes] );
             else
                   HWR_SetDefaultShapes( _recognizer );
             BOOL b = [defaults boolForKey:kRecoOptionsFirstStartKey];
             if ( b == YES )
             {
                   // set recognizer options
                   unsigned int flags = HWR_GetRecognitionFlags(_recognizer);
```

```
                    if ( [defaults boolForKey:kRecoOptionsSingleWordOnly] )
                          flags |= FLAG_SINGLEWORDONLY;
                    else
                          flags &= ~FLAG_SINGLEWORDONLY;
                    if ( [defaults boolForKey:kRecoOptionsSeparateLetters] )
                          flags |= FLAG_SEPLET;
                    else
                          flags &= ~FLAG_SEPLET;
                    if ( [defaults boolForKey:kRecoOptionsInternational] )
                          flags |= FLAG_INTERNATIONAL;
                    else
                          flags &= ~FLAG_INTERNATIONAL;
                    if ( [defaults boolForKey:kRecoOptionsDictOnly] )
                          flags |= FLAG_ONLYDICT;
                    else
                          flags &= ~FLAG_ONLYDICT;
                    if ( [defaults boolForKey:kRecoOptionsSuggestDictOnly] )
                          flags |= FLAG_SUGGESTONLYDICT;
                    else
                          flags &= ~FLAG_SUGGESTONLYDICT;
                    if ( [defaults boolForKey:kRecoOptionsUseUserDict] )
                          flags |= FLAG_USERDICT;
                    else
                          flags &= ~FLAG_USERDICT;
                    if ( [defaults boolForKey:kRecoOptionsUseLearner] )
                          flags |= FLAG_ANALYZER;
                    else
                          flags &= ~FLAG_ANALYZER;

                    if ( [defaults boolForKey:kRecoOptionsUseCorrector] )
                          flags |= FLAG_CORRECTOR;
                    else
                          flags &= ~FLAG_CORRECTOR;
                    if ( ! [defaults boolForKey:kRecoOptionsSpellIgnoreNum] )
                          flags |= FLAG_SPELLIGNORENUM;
                    else
                          flags &= ~FLAG_SPELLIGNORENUM;

                    if ( ! [defaults boolForKey:kRecoOptionsSpellIgnoreUpper] )
                          flags |= FLAG_SPELLIGNOREUPPER;
                    else
                          flags &= ~FLAG_SPELLIGNOREUPPER;

                    HWR_SetRecognitionFlags( _recognizer, flags );
              }
        }
      }
}
else if ( NULL != _recognizer )
{
     HWR_FreeRecognizer( _recognizer,
                              [strUserFile UTF8String],
                              [strLearner UTF8String],
                              [strCorrector UTF8String] );
     _recognizer = NULL;
}
```

…………………………

Document Revision History

This table describes the changes WritePad SDK Recognizer API.

Date	Notes
2015-02-26	WritePad SDK 5.0: added Indonesian language, new API
2014-05-29	WritePad SDK 4.2: UCHR is now unsigned short (UNICODE, UTF-16)
2014-04-09	Minor updates reflecting changes in WritePad SDK 4.1.
2013-11-06	Minor updates reflecting changes in WritePad SDK 4.0.
2013-01-27	Fixed minor error in the recognizer API. Updated contact information.
2012-09-10	Added new sample code and updated overview section
2012-08-20	Added new WritePad 3.5 APIs. Several existing APIs changed.
2012-01-18	Updated for version 3.0. New sample code, new APIs.
2011-05-10	Added new WritePad 2.0 APIs, changed some existing APIs
2010-10-23	Updated with WritePad version 1.5 APIs. Updated product description.