

CIDAR v.1.0

Container ID Automatic Reader

DLL Reference Manual

Table of Contents

Initialization / Finalization	4
<i>cidarInit</i>	4
<i>cidarEnd</i>	5
Reading Container ID Codes	6
<i>cidarRead</i>	6
<i>cidarReadRGB24</i>	7
<i>cidarReadRGB32</i>	8
<i>cidarReadBMP</i>	9
<i>cidarReadJPG</i>	10
Retrieving Results	11
<i>cidarCodeFound</i>	11
<i>cidarCodeVerified</i>	12
<i>cidarGetCode</i>	13
<i>cidarGetNumberOfCharacters</i>	14
<i>cidarGetGlobalConfidence</i>	15
<i>cidarGetAverageCharacterHeight</i>	16
<i>cidarGetCharacterConfidence</i>	17
<i>cidarGetRectangle</i>	18
Time Management	19
<i>cidarGetProcessingTime</i>	19
<i>cidarSetTimeOut</i>	20
Optional Configuration	21
<i>cidarSetCorrectionCoefficients</i>	21
<i>cidarSetDistortionCorrectionOff</i>	22
<i>cidarConfigureAutomaticCharacterHeight</i>	23
<i>cidarSetRectangle</i>	24
<i>cidarConfigureCodeFormat</i>	25

Storing User Data into the HASP Dongle	26
<i>cidarWriteHASP</i>	26
<i>cidarReadHASP</i>	27
Example of Use	28

Initialization / Finalization

cidarInit

Initializes the **Container ID Automatic Reader (CIDAR)**. It loads the Artificial Neural Networks used by the OCR and initializes parameters. This function must be called before calling any other function in this library.

```
long cidarInit ( long lAverageCharacterHeight = -1,  
                bool bDuplicateLines = false,  
                bool bTrace = false );
```

Arguments

- lAvCharacterHeight* Approximate average height of the characters in the codes to read. If this argument is **-1**, the library uses *automatic height mode* and tries to read characters of any height (between 30 and 100 pixels). If *automatic height mode* is selected (**-1**), the processing time will be increased.
- bDuplicateLines* In order to properly recognize images acquired with only half of the scan lines, this argument must be **true**. For images acquired with all the lines, this parameter must be **false**.
- bTrace* This parameter must be set to **false**.

Return Value

- 0** → Error.
- 1** → Ok.

cidarEnd

Frees the memory allocated by the **Container ID Automatic Reader (CIDAR)**.

```
void cidarEnd ( void );
```

Reading Container ID Codes

cidarRead

This function reads the container ID code present within an image. The input to this function is an image. A set of functions is supplied to retrieve the result of the recognition (see *Retrieving Results* chapter below).

The input supplied to this function is the *image buffer* in **256 greyscale** levels (1 byte per pixel). The *width* and *height* of the image must be supplied as well.

```
long cidarRead ( long lWidth,  
                long lHeight,  
                unsigned char * pbImageData );
```

Arguments

<i>lWidth</i>	Width (in pixels) of the image that will be analyzed.
<i>lHeight</i>	Height (in pixels) of the image that will be analyzed.
<i>pbImageData</i>	Buffer with the image data (pixels), in 256 grey levels (1 byte per pixel).

Return Value

- 0** → Error.
- 1** → Ok.

cidarReadRGB24

This function reads the container ID code present within an image. The input to this function is an image. A set of functions is supplied to retrieve the result of the recognition (see *Retrieving Results* chapter bellow).

The input supplied to this function is the *image buffer* in **RGB-24 bits** (3 bytes per pixel) format. The *width* and *height* of the image must be supplied as well.

```
long cidarReadRGB24 ( long lWidth,  
                    long lHeight,  
                    unsigned char * pbImageData,  
                    bool bFlip = false );
```

Arguments

<i>lWidth</i>	Width (in pixels) of the image that will be analyzed.
<i>lHeight</i>	Height (in pixels) of the image that will be analyzed.
<i>pbImageData</i>	<i>Buffer</i> with the image data (pixels) using 3 bytes per pixel (RED, GREEN, BLUE).
<i>bFlip</i>	This value must be true only if the RGB buffer contains first the bottom row of the image, then the next one upwards, and so on. The last line of values in the buffer contains the top row of pixels in the image. Some devices acquire the RGB buffer in this way (<i>bottom-up</i>).

Return Value

- 0** → Error.
- 1** → Ok.

cidarReadRGB32

This function reads the container ID code present within an image. The input to this function is an image. A set of functions is supplied to retrieve the result of the recognition (see *Retrieving Results* chapter bellow).

The input supplied to this function is the *image buffer* in **RGB-32 bits** (4 bytes per pixel) format. The *width* and *height* of the image must be supplied as well.

```
long cidarReadRGB32 ( long lWidth,  
                    long lHeight,  
                    unsigned char * pbImageData,  
                    bool bFlip = false );
```

Arguments

<i>lWidth</i>	Width (in pixels) of the image that will be analyzed.
<i>lHeight</i>	Height (in pixels) of the image that will be analyzed.
<i>pbImageData</i>	<i>Buffer</i> with the image data (pixels) using 4 bytes per pixel (RED, GREEN, BLUE, ALPHA).
<i>bFlip</i>	This value must be true only if the RGB buffer contains first the bottom row of the image, then the next one upwards, and so on. The last line of values in the buffer contains the top row of pixels in the image. Some devices acquire the RGB buffer in this way (<i>bottom-up</i>).

Return Value

- 0** → Error.
- 1** → Ok.

cidarReadBMP

This function reads the container ID code present within an image. The input to this function is an image. A set of functions is supplied to retrieve the result of the recognition (see *Retrieving Results* chapter below).

The input supplied to this function is an image file in standard Bitmap (BMP) format.

```
long cidarReadBMP ( char * strFilename );
```

Arguments

strFilename Filename of BMP image to process.

Return Value

0 → Error.
1 → Ok.

cidarReadJPG

This function reads the container ID code present within an image. The input to this function is an image. A set of functions is supplied to retrieve the result of the recognition (see *Retrieving Results* chapter below).

The input supplied to this function is an image file in standard Jpeg (JPG) format.

```
long cidarReadJPG ( char * strFilename );
```

Arguments

strFilename Filename of JPG image to process.

Return Value

0 → Error.

1 → Ok.

Retrieving Results

cidarCodeFound

Returns whether a container ID code was found or not.

```
bool cidarCodeFound ( void );
```

Return Value

Returns **true** if a code was found and read and **false** otherwise.

cidarCodeVerified

Returns whether the container ID code read was successfully verified using the checksum digit.

```
bool cidarCodeVerified ( void );
```

Return Value

Returns **true** if code's checksum digit (the eleventh digit) matches the checksum calculated from the first ten characters read. Returns **false** otherwise.

cidarGetCode

Returns the text (in **ASCII** format) of the container ID code read in the last processed image.

```
long cidarGetCode ( char * strResult );
```

Arguments

strResult String where the ASCII text is returned.

Return Value

0 → Error.
1 → Ok.

cidarGetNumberOfCharacters

Returns the number of characters present in the last code recognized by **CIDAR**.

```
long cidarGetNumberOfCharacters ( void );
```

Return Value

Number of recognized characters in the last code recognized.

cidarGetGlobalConfidence

Returns a confidence factor for the result of the last code recognized by the **Container ID Automatic Reader**.

This value is expressed as a percentage (0% – 100%).

```
float cidarGetGlobalConfidence ( void );
```

Return Value

The Confidence Factor for the last recognition process.

cidarGetAverageCharacterHeight

Returns the average height (in pixels) of the characters present in the last code recognized by the **Container ID Automatic Reader**.

```
float cidarGetAverageCharacterHeight ( void );
```

Return Value

The average height (in pixels) of the characters in the last ID code analyzed.

cidarGetCharacterConfidence

Returns a confidence factor for a given character within the last ID code analyzed.

This value is expressed as a percentage (0% – 100%).

```
float cidarGetCharacterConfidence ( long lIndex );
```

Arguments

lIndex (0..n) Index of the character we want to retrieve the confidence factor for.

Return Value

Character Confidence Factor.

cidarGetRectangle

Returns the coordinates of the rectangle containing the container ID code within the last image processed.

```
void cidarGetRectangle ( long * plLeft,  
                        long * plTop,  
                        long * plRight,  
                        long * plBottom );
```

Arguments

<i>plLeft</i>	X-Coordinate of the Upper-Left corner of the rectangle.
<i>plTop</i>	Y-Coordinate of the Upper-Left corner of the rectangle.
<i>plRight</i>	X-Coordinate of the Lower-Right corner of the rectangle.
<i>plBottom</i>	Y-Coordinate of the Lower-Right corner of the rectangle.

Time Management

cidarGetProcessingTime

Returns the processing time for the last reading operation.

This value is expressed in milliseconds.

```
long cidarGetProcessingTime ( void );
```

Return Value

Processing time (in milliseconds) for the last reading.

cidarSetTimeout

This function specifies the maximum processing time for reading operations.

This value is expressed in milliseconds.

```
void cidarSetTimeout (unsigned long ulMilliseconds );
```

Arguments

lMilliseconds Maximum processing time (in milliseconds).

Remarks

This maximum time is approximate. This means that the actual processing time can be (in some cases) slightly longer than the time-out specified.

Optional Configuration

cidarSetCorrectionCoefficients

This function sets the distortion correction coefficients that will be applied to all the images before being analyzed.

The types of distortions that can be corrected are: tangential distortion (horizontal and/or vertical perspective) and rotation.

The specified coefficients will be applied to all the images until this function is called again with different arguments of **cidarSetDistortionCorrectionOff** is called.

```
void cidarSetCorrectionCoefficients ( float fDistance,  
                                     float fVerticalCoeff,  
                                     float fHorizontalCoeff,  
                                     float fAngle );
```

Arguments

<i>fDistance</i>	Approximate distance between camera and object (in meters).
<i>fVerticalCoeff</i>	Coefficient for correcting Vertical Perspective.
<i>fHorizontalCoeff</i>	Coefficient for correcting Horizontal Perspective.
<i>fAngle</i>	Angle for correcting Rotation.

cidarSetDistortionCorrectionOff

This function deactivates the distortion correction pre-process.

Use ***cidarSetCorrectionCoefficients*** to activate it again.

```
void cidarSetDistortionCorrectionOff ( void );
```

cidarConfigureAutomaticCharacterHeight

This function configures the *Automatic Character Height* range.

When **cidarInit** is called with the argument *IAvCharacterHeight* set to -1, the automatic character height mode is selected. By default, the range of character heights scanned in this mode is from 30 pixels to 100 pixels.

By using this function, the user can select a different height range.

```
long cidarConfigureAutomaticCharacterHeight ( long lMinCharHeight,  
                                             long lMaxCharHeight );
```

Arguments

<i>lMinCharHeight</i>	Minimum character height (in pixels) of the characters to read.
<i>lMaxCharHeight</i>	Maximum character height (in pixels) of the characters to read.

Return Value

- 0** → Error.
- 1** → Ok.

Remarks

To recover the default configuration, the following code must be executed:

```
cidarConfigureAutomaticCharacterHeight ( 30, 100 );
```

cidarSetRectangle

Sets the rectangle within the image where the inspection process will take place. Only the provided region will be inspected.

This function can be used to speed up the process when the approximate position of the ID code within the image is known.

```
long cidarSetRectangle ( long lLeft,  
                        long lTop,  
                        long lWidth,  
                        long lHeight );
```

Arguments

lLeft, lTop Left-top corner coordinates of the rectangle (in *pixels*).

lWidth, lHeight Rectangle dimensions (in *pixels*).

Return Value

0 → Error.

1 → Ok.

Remarks

The inspection rectangle specified will be applied to all the further inspections until **cidarSetRectangle** is called again with different parameters.

In order to inspect the whole image, **cidarSetRectangle** must be called with all its parameters set to 0:

```
cidarSetRectangle (0, 0, 0, 0);
```

By default, after library initialization with **cidarInit**, the inspection rectangle is set to inspect the whole image.

cidarConfigureCodeFormat

Provides information about the possible formats of the container codes to read.

The container ID codes can be horizontal or vertical. If they are horizontal, all the characters can be in one line or split in two lines.

In any of the cases above, the characters of the ID code can be darker than the background where they are printed, or lighter than the background.

By default, the CIDAR tries to recognize all the possible types of codes but if we can restrict the formats of the codes to read, the processing time will be decreased. For example, if we know that for a given system/installation, the orientation of all the codes to be processed will be vertical, we can inform the CIDAR library and it will not try to find and recognize horizontal codes. If we are installing a system where we know that all the containers will be dark and will have the characters printed in white or yellow, we can tell the CIDAR that the characters will be brighter than the background and this will save processing time also.

```
long cidarConfigureCodeFormat ( unsigned short usOrientation,  
                                unsigned short usLines,  
                                unsigned short usPolarity );
```

Arguments

<i>usOrientation</i>	Indicates the orientation of the ID codes to recognize. It can take the following values: CODE_FORMAT_HORIZONTAL CODE_FORMAT_VERTICAL CODE_FORMAT_HORIZONTAL_OR_VERTICAL
<i>usLines</i>	Indicates the number of lines the horizontal codes can have: CODE_FORMAT_ONE_LINE CODE_FORMAT_TWO_LINES CODE_FORMAT_ANY_LINES
<i>usPolarity</i>	Indicates the polarity of the characters related to the background where they are printed: CODE_FORMAT_BLACK_ON_WHITE CODE_FORMAT_WHITE_ON_BLACK CODE_FORMAT_ANY_ON_ANY

Return Value

- 0** → Error.
- 1** → Ok.

Storing User Data into the HASP Dongle

cidarWriteHASP

Writes data into the internal memory of the HASP dongle. This capability for storing data into the HASP can be used for any purpose. Data is encrypted automatically before being written into the HASP memory.

A maximum of 24 bytes can be written.

```
long cidarWriteHASP ( unsigned char * pData,  
                    long lSize );
```

Arguments

pData Buffer containing the data to be written in to the HASP internal memory.

lSize Size (in bytes) of the data to write (maximum 24 bytes).

Return Value

0 → Error.
1 → Ok.

cidarReadHASP

Reads the data stored in the internal memory of the HASP dongle. Data is automatically decrypted after being read from the HASP memory.

```
long cidarReadHASP ( unsigned char * pData,  
                    long lSize );
```

Arguments

pData Buffer where the retrieved data will be stored.

lSize Size (in bytes) of the data to read.

Return Value

0 → Error.

1 → Ok.

Example of Use

```
void func ()
{
    // The characters in the images to process can vary in height, there is
    // no need to duplicate horizontal lines (all the scan lines acquired).
    bool ok = (bool) cidarResult (-1, false );
    if (ok)
    {
        // The character height can vary between 30 and 50 pixels.
        cidarResultConfigureAutomaticCharacterHeight (30, 50);

        unsigned char * buffer;    // Image buffer.
        bool verified;            // Code verified or not.
        long numchars;            // Number of characters read.
        char text[32];            // String where result will be stored.
        float cf;                 // Confidence Factor of the reading.
        float characterCf[32];    // Array to store the characters
                                // confidence factor.

        // Buffer memory allocation.
        . . .
        // Acquire the image buffer in 256 greyscale levels (1 byte per
        // pixel) and store it into buffer.
        . . .
        // Image dimensions are 384x288 pixels.

        ok = (bool) cidarResultRead ( 384, 288, buffer );
        if( (ok) && ( cidarResultCodeFound ( ) )
            {
                verified = cidarResultCodeVerified ( );
                cidarResultGetCode ( text );
                cf = cidarResultGetGlobalConfidence ( );
                numchars = cidarResultGetNumberOfCharacters ( );
                for (long i = 0; i < numchars; i++)
                {
                    characterCf[i] = cidarResultGetCharacterConfidence ( i );
                }
            }
        else
        {
            // Error reading ID code or code not found.
        }
        cidarResultEnd ();
    }
    else
    {
        // Error initializing the CIDAR.
    }
}
```