LSAPI
SERVICE INTERFACE

By
C.T.S. Electronics S.p.A.
Corso Vercelli 332
10015 IVREA (TO)

tel. ++390125235611
fax. ++390125235623
web site: www.ctsgroup.it
e-mail: techsupp@ctsgroup.it

Last revision : 10 November 2014

Contents

1. Overview

This document describes the C-Language compatible application programming interface (API) of the LS service functions for the Win32 subsystem of the Windows operating system, for LS40, LS100, LS150, LS500, LS505, LS510, LS515 and LS800 check scanner models.

The API is independent from the physical connection technology, whether it be SCSI, RS232, USB or Ethernet used to connect the check scanners to the host system.

The reader must have a background knowledge of Win32 environment architecture and programming techniques.

A further little tip for programmers...
this documentation should always be up-to-date, however during the development process, it is advisable to mainly refer to the include file lsapi.h supplied in the last version of the SDK , in order to get all the current and correct interface data (defines, structures, function prototypes...)

The LsApi.dll library extract same library for support, this library are placed in the same folder as where LsApi.dll is located, the folder can be also selected to by setting the following registry key,
HKEY_LOCAL_MACHINE\SOFTWARE\CTS Electronics\LsApiPath
With the complete folder path.

2. Interface Functions Description

All functions exported by the LS service are implemented in **LsApi.DLL.**

To build an application program written in “C”-Language that makes use of these interface functions it is necessary to include the file named **LsApi.H** and link **LsApi.LIB** library.
To start an application program that uses this service it is necessary to install in the application directory the files listed below :

- LsApi.DLL

LS service contains the following modules :

- LsApi.DLL
- LsApi.H
- LsApi.LIB
- IMG_UTIL.DLL
- LFBMP13N.DLL
- LFCMP13N.DLL
- LFFAX13N.DLL
- LTDIS13N.DLL
- LTDLG13N.DLL
- LTFIL13N.DLL
- LTIMG13N.DLL
- LTKRN13N.DLL
- LTEFX13N.DLL
- CtsDecod.DLL
- MicrDecodelib.DLL
- CtsPdf.DLL (optional)
- BarDecode.DLL (optional)
- CtsDataMatrix.DLL (optional)
- CtsQRCode.DLL (optional)
- CtsTopImage.DLL (optional)
- CtsClearPIX.DLL (optional)
- CtsImagePro.DLL (optional)
- CtsPdfDriverLicense.DLL (optional)
- CtsIQA.DLL (optional)
- CtsMetrix.DLL (optional)
- CtsBarcodeLocate.DLL (optional)
- CtsRS.DLL (optional)

*All functions work in **suspensive** (synchronous) mode.*

With the Ethernet connection the images are exchanged over the network with a SSL secure protocol.
This is true is the following libraries are present in the same folder as where LsApi.dll is located on the computer side:

LIBEAY32.DLL
SSLEAY32.DLL
LIBSSL32.DLL

This libraries mentioned above are downloadable from the link www.openssl.org.

3. LsApi.ini Description

All exported functions can be supported from an external file configuration called LsApi.ini.

This file must be copied in the same folder where is installed LsApi.dll.

It's composed of two different sections, one specific for the unit model, another specific for the single functions (this parser must be implemented).
It's not necessary that all section be present in the ini file, only the sections present will be handled.

Note that it is possible to activate emulations via LsApi.ini file. Only one emulation can be activated at the same time.
Hereafter it's explained the flow adopted to connect a LSConnect :

1. Is FLAG Ls150_USB_as_Ls150_IpBox = TRUE goto 2
2. Logical name…   If not found goto 3…..if found connect to it…
3. IP address set in ini file   If not found goto 4….if found connect to it
4. Default IP address 172.16.1.109….if not found the ERROR, if found connect to it…

If an empty entry of [IPBOX] in the ini file will always try to connect to the default IP Address 172.16.1.109.

Sections description and their possible parameters :

```
;-----------------------------------------------------------------------------
; [Global] - Section to configure global parameters
;
; CheckCartridge= Enables or disables the automatic check on the cartridge level
;               Legal values : TRUE, FALSE, POPUP.
;               Default : FALSE
;
; ThresholdInkDropsCartridge = this parameter defines the number of spots to be used
;                           as a maximum for determining that the cartridge level
;                           is low.
;                           Advised: 112000000
;-----------------------------------------------------------------------------
[Global]
CheckCartridge = true
ThresholdInkDropsCartridge = 106000000


;-----------------------------------------------------------------------------
; [Logging] - Section to configure trace parameters
;
; Enable = Enable disable the logging
;             Legal values : true or false
;
; BatchMode = Enable disable the batch logging
;               Legal values : true or false
;
; LogDir = Logging Folder es. C:\Windows\Temp
;             if missing then directory is CtsTrace of lsapi directory
;
; MaxSize = Max size file logging (per file in batch mode) in bytes. (min value accepted 1 Mb.)
;-----------------------------------------------------------------------------
[Logging]
Enable = false
BatchMode = true
```

```
LogDir = C:\Windows\Temp
MaxSize = 4194304


;-----------------------------------------------------------------------------
; [CtsMetrix] - Section to configure a LS device monitoring.
;
; Enabled = TRUE monitoring enabled, FALSE monitoring disabled
;
; ModulesPath = Path of monitor library and program
;
; LogFontForMICR = Type of MICR monitored CMC7 or E13B (default)
;
;-----------------------------------------------------------------------------
[CtsMetrix]
Enabled = false
ModulesPath = ".\CtsMetrix"
LogFontForMICR = CMC7


;-----------------------------------------------------------------------------
; [LsSharing] - Section to configure a shared connection.
;
; PC_IPAddress = IP address of the PC with the LS unit
; Computer_Name = Network Computer Name
;
;-----------------------------------------------------------------------------
[LsSharing]
PC_IPAddress = 172.16.1.197
;Computer_Name = PC_NAME


;-----------------------------------------------------------------------------
; [LS40] - Section to configure the LS40 model
;
; Deskew=        Enables or disables the automatic image deskew
;                Legal values : TRUE, FALSE.
;                Advised : TRUE
;
; Mocr =     Enables or disables MICR+OCR
;                Legal values : E13B, CMC7, FALSE
;                Advised : E13B or CMC7 depending on the check type used in your market
;
; Ls40_as_Ls100 = If TRUE a LS40 connected works as a LS100
;                Legal values : TRUE, FALSE.
;
; Ls40_USB_as_Ls40_IpBox = If TRUE a LS40IPBOX connected works as a LS40-USB
;                Legal values : TRUE, FALSE.
;
; Ls40_Stamp = If this parameter is present it must be set to FRONT_STAMP to force the
;                stamp  when the LS40 is connected in LS100 emulation mode.
;
; Ls100_Model = String returned by the LSIdentify API when a LS40  is connected as LS100.
;                The string is the version of the Firmware expected by the application.
;
; BitonalMethod = This parameter defines the default method to be used for the bitonal conversion.
;                Legal values :
;                4 ALGORITHM_CTS                (Threshold Dinamyc)
;                6 ALGORITHM_CTS_3              (Threshold Fixed)
;                7 ALGORITHM_TOP_IMAGE          (Threshold Dinamyc)
;               8 ALGORITHM_IMAGE_PRO           (Threshold Dinamyc)
;               9 ALGORITHM_CLEAR_PIX           (Threshold Dinamyc)
;               Advised : 4
;
; BitonalThreshold = This parameter defines the default threshold to be used for the bitonal conversion.
;               Legal values :
;               for method ALGORITHM_CTS (4) from 50 to 600. Advised 290
;               for method ALGORITHM_CTS_3 (6) from 0 to 15. Advised 9
;               for method ALGORITHM_TOP_IMAGE (7) from 50 to 600. Advised 290
;               for method ALGORITHM_CLEAR_PIX (9) from 50 to 600. Advised 290
;
; ThresholdBackground = Value used as threshold for clear the document.
;               Legal values : 0 to 255 (default 68)
;
; BlankInCodeline = This parameter defines the number of blank insert in codeline.
;               Legal values :
;               NO_BLANK
;               ONE_BLANK
;
; LightIntensity = This parameter allows to increase the light intensity for the ls40
;               Legal Values:
;               from 0 to 30
;-----------------------------------------------------------------------------
[LS40]
Deskew = TRUE
Mocr = E13B
BitonalMethod = 4
BitonalThreshold = 290
Ls40_as_Ls100 = TRUE                    ; if True a LS40 connected works as a LS100
Ls40_USB_as_Ls40_IpBox = FALSE          ; if TRUE a Ls40 connected to IPBox work as LS40 USB
Ls100_Model = LS100/7_09                ; Is the Model returned by the LSIdentify when a LS40 is
                                        connected as LS100
;BlankInCodeline = ONE_BLANK
;ThresholdBackGround = 68
;LightIntensity = 0


;-----------------------------------------------------------------------------
; [LS100] - Section to configure the LS100 model
;
; DoubleLeafingValue = It sets the default double leafing sensitivity for each document.
;               Legal values: percentage from 1 to 100 from the default device setting.
;               Advised : 40
;
; DoubleLeafingMinLength= minimum document length in millimeter that can be processed
;
; DoubleLeafingMaxLength= maximum document length in millimeter that can be processed
;
; Deskew=       Enables or disables the automatic image deskew
;               Legal values : TRUE, FALSE.
;               Advised : TRUE
;
; Mocr =        Enables or disables MICR+OCR
;               Legal values : E13B, CMC7, FALSE
;               Advised : E13B or CMC7 depending on the check type used in your market
;
; Ls100_USB_as_Ls100_IpBox= If TRUE a LS100IPBOX connected works as a LS100-USB
;               Legal values : TRUE, FALSE.
;
; Ls100_IpBox_as_Ls100_Ip= If TRUE a LS100IPBOX connected works as a LS100IP
;               Legal values : TRUE, FALSE.
;
; Ls100_Model = String returned by the LSIdentify API when a LS100  is connected in emulation mode.
;               The string is the version of the Firmware expected by the application.
;
; BitonalMethod = This parameter defines the default method to be used for the bitonal conversion.
;               Legal values :
;               4 ALGORITHM_CTS                 (Threshold Dinamyc)
;               6 ALGORITHM_CTS_3               (Threshold Fixed)
;               7 ALGORITHM_TOP_IMAGE           (Threshold Dinamyc)
;               8 ALGORITHM_IMAGE_PRO           (Threshold Dinamyc)
;               9 ALGORITHM_CLEAR_PIX           (Threshold Dinamyc)
;               Advised : 4
;
; BitonalThreshold = This parameter defines the default threshold to be used for the bitonal conversion.
;               Legal values :
;               for method ALGORITHM_CTS (4) from 50 to 600. Advised 290
;               for method ALGORITHM_CTS3 (6) from 0 to 15. Advised 9
;               for method ALGORITHM_TOP_IMAGE (7) from 50 to 600. Advised 290
;               for method ALGORITHM_CLEAR_PIX (9) from 50 to 600. Advised 290
;
; ThresholdBackground = Value used as threshold for clear the document.
;               Legal values : 0 to 255 (default 68)
;
;-----------------------------------------------------------------------------
[LS100]
DoubleLeafingValue = 40
DoubleLeafingMinLength = 150
DoubleLeafingMaxLength = 216
Deskew = TRUE
Mocr = E13B
BitonalMethod = 4
BitonalThreshold = 290
Ls100_USB_as_Ls100_IpBox = FALSE
Ls100_IpBox_as_Ls100_Ip = FALSE
Ls100_Model = LS100/7_09
Ls100_USB_as_Ls100_IpBox = TRUE         ; if TRUE a Ls100 connected to IPBox work as LS100 USB
Ls100_IpBox_as_Ls100_Ip = FALSE         ; if TRUE a Ls100 IPBox work as LS100 IP
;ThresholdBackGround = 68


;-----------------------------------------------------------------------------
; [LS150] - Section to configure the LS150 model
;
; DoubleLeafingValue = It sets the default double leafing sensitivity for each document.
;               Legal values: percentage from 1 to 100 from the default device setting.
;               Advised : 33
;
; DoubleLeafingMinLength= minimum document length in millimeter that can be processed
;
; DoubleLeafingMaxLength= maximum document length in millimeter that can be processed
;
; Deskew =    Enables or disables the automatic image deskew
;               Legal values : TRUE, FALSE.
;               Advised : E13B or CMC7 depending on the check type used in your market
;
; Mocr =      Enables or disables  MICR+OCR
;               Legal values : E13B, CMC7, FALSE
;               Advised : E13B
;
```

```
; Ls150_as_Ls100 = If TRUE a LS150 connected works as a LS100
;               Legal values : TRUE, FALSE.
;
; Ls150_USB_as_Ls150_IpBox = If TRUE a LS150IPBOX connected works as a LS150 USB
;               Legal values : TRUE, FALSE.
;
; Ls150_Stamp =   If this parameter is present it must be set to FRONT_STAMP to force the
;               stamp when the LS150 is connected in LS100 emulation mode.
;
; Ls100_Model =   String returned by the LSIdentify API when a LS150  is connected as LS100.
;               The string is the version of the Firmware expected by the application.
;
; BitonalMethod = This parameter defines the default method to be used for the bitonal conversion.
;               Legal values :
;               4 ALGORITHM_CTS                 (Threshold Dinamyc)
;               6 ALGORITHM_CTS_3               (Threshold Fixed)
;               7 ALGORITHM_TOP_IMAGE           (Threshold Dinamyc)
;               8 ALGORITHM_IMAGE_PRO           (Threshold Dinamyc)
;               9 ALGORITHM_CLEAR_PIX           (Threshold Dinamyc)
;               Advised : 4
;
; BitonalThreshold = This parameter defines the default threshold to be used for the bitonal conversion.
;               Legal values :
;               for method ALGORITHM_CTS (4) from 50 to 600. Advised 290
;               for method ALGORITHM_CTS3 (6) from 0 to 15. Advised 9
;               for method ALGORITHM_TOP_IMAGE (7) from 50 to 600. Advised 290
;               for method ALGORITHM_CLEAR_PIX (9) from 50 to 600. Advised 290
;
; ThresholdBackground = Value used as threshold for clear the document.
;               Legal values : 0 to 255 (default 68)
;
; LightIntensity = This parameter allows to increase the light intensity for the ls150
;               Legal Values:
;               from 0 to 30
;
; Endorsement_Amount_In_Bold = Enable the Bold font ONLY for the amount (the number that follows the
$ ;              sign, in the string can be present max 2 $ sign.
;
; LenDocMinAccepted = Length minimun document accepted from device, 0 for Fw default value.
;-----------------------------------------------------------------------------
[LS150]
DoubleLeafingValue = 33
DoubleLeafingMinLength = 150
DoubleLeafingMaxLength = 225
Deskew = TRUE
Mocr = E13B
BitonalMethod = 4
BitonalThreshold = 290
Ls150_as_Ls100 = TRUE                   ; if True a LS150 connected works as a LS100
Ls150_USB_as_Ls150_IpBox = FALSE        ; if TRUE a Ls150 connected to IPBox work as LS150 USB
Ls150_Stamp = FRONT_STAMP               ; Force the stamp when a LS150 is connected as LS100
Ls100_Model = LS100/3_52                ; Is the Model returned by the LSIdentify when a LS150 is
                                        connected as LS100
;ThresholdBackGround = 68
;LightIntensity = 0
Endorsement_Amount_In_Bold = TRUE
;LenDocMinAccepted = 100


;-----------------------------------------------------------------------------
```

```
; [LS515] - Section to configure the LS515 model
;
; Deskew=        Enables or disables the automatic image deskew
;                Legal values : TRUE, FALSE.
;                Advised : TRUE
;
; Mocr =     Enables or disables  MICR+OCR
;                Legal values : E13B, CMC7, FALSE
;                Advised : E13B or CMC7 depending on the check type used in your market
;
; Ls515_USB_as_Ls515_IpBox = If TRUE a LS515IPBOX connected works as a LS515-USB
;                Legal values : TRUE, FALSE.
;
; BitonalMethod = This parameter defines the default method to be used for the bitonal conversion.
;                Legal values :
;                4 ALGORITHM_CTS                    (Threshold Dinamyc)
;                6 ALGORITHM_CTS_3                  (Threshold Fixed)
;                7 ALGORITHM_TOP_IMAGE              (Threshold Dinamyc)
;                8 ALGORITHM_IMAGE_PRO              (Threshold Dinamyc)
;                9 ALGORITHM_CLEAR_PIX              (Threshold Dinamyc)
;                Advised : 4
;
; BitonalThreshold = This parameter defines the default threshold to be used for the bitonal conversion.
;                Legal values :
;                for method ALGORITHM_CTS (4) from 50 to 600. Advised 290
;                for method ALGORITHM_CTS3 (6) from 0 to 15. Advised 9
;                for method ALGORITHM_TOP_IMAGE (7) from 50 to 600. Advised 290
;                for method ALGORITHM_CLEAR_PIX (9) from 50 to 600. Advised 290
;
; ThresholdBackground = Value used as threshold for clear the document.
;                Legal values : 0 to 255 (default 68)
;
;------------------------------------------------------------------------------
[LS515]
Deskew= TRUE
Mocr = E13B
BitonalMethod = 4
BitonalThreshold = 290
Ls515_USB_as_Ls515_IpBox = FALSE
;ThresholdBackGround = 68


;------------------------------------------------------------------------------
; [LS800] - Section to configure the LS800 model
;
; Deskew=        Enables or disables the automatic image deskew
;                Legal values : TRUE, FALSE.
;                Advised : TRUE
;
; BitonalMethod = This parameter defines the default method to be used for the bitonal conversion.
;                Legal values :
;                4 ALGORITHM_CTS                    (Threshold Dinamyc)
;                6 ALGORITHM_CTS_3                  (Threshold Fixed)
;                7 ALGORITHM_TOP_IMAGE              (Threshold Dinamyc)
;                8 ALGORITHM_IMAGE_PRO              (Threshold Dinamyc)
;                9 ALGORITHM_CLEAR_PIX              (Threshold Dinamyc)
;                Advised : 4
;
; BitonalThreshold = This parameter defines the default threshold to be used for the bitonal conversion.
;                Legal values :
```

```
;               for method ALGORITHM_CTS (4) from 50 to 600. Advised 290
;               for method ALGORITHM_CTS3 (6) from 0 to 15. Advised 9
;               for method ALGORITHM_TOP_IMAGE (7) from 50 to 600. Advised 290
;               for method ALGORITHM_CLEAR_PIX (9) from 50 to 600. Advised 290
;
;------------------------------------------------------------------------------
[LS800]
Deskew = TRUE
BitonalMethod = 4
BitonalThreshold = 290


;------------------------------------------------------------------------------
; [IPBox] - Section to configure a IPBox connection.
;
; IPBox_Address = IP address of the device (used only in emulation modes)
;
; BW_OnBaord = false the conversion will done on PC (more speed)
;                    = true the conversion will done on the LSConnect (less network traffic)
;
;------------------------------------------------------------------------------
[IpBox]
IPBox_NetworkName = ctslsconnect          ; Network logical name
IPBox_Address = 172.16.1.190              ; The IP Address set on the LSConnect
BW_OnBoard = false
```

4. Basic functions

This section describes the function calls available with the base service.
A function that one specific peripheral doesn't support, returns the LS_OKAY reply.

Some functions are also supported in such a way that the parameters are obtained from a configuration file LsApi.ini that must be present in the same folder where LsApi.dll is located.

Please refer to the section 3 for the details of this configuration file.

4.1. LSConnect

#include "LSApi.h"

```
Result API LSConnect(  HWND      hWnd,
                       HANDLE    hInst,
                       SHORT     LsUnit,
                       SHORT     *hConnect);
```

Description

Open a connection between the application and the LS service.
The function returns a connection handle.

When the parameter *LsUnit* is set to **LS_515_USB**, the function before returning the reply LS_PERIPHERAL_NOT_FOUND tries to connect also the **LS520** unit.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hInst

Hinstance of the application window **(Reserved for future use)**.

LsUnit

Specifies the device that must be connected, the accepted values are :

- LS_40_LSCONNECT
- LS_40_USB
- LS_100_LSCONNECT
- LS_100_USB
- LS_100_ETH
- LS_100_RS232
- LS_150_LSCONNECT
- LS_150_USB
- LS_200_USB
- LS_5xx_SCSI
- LS_515_LSCONNECT
- LS_515_USB
- LS_520_USB
- LS_800_USB
- LS_UNIT_SHARED

hConnect

Connection handle returned by the service, this value shall be used in the most of the other functions that talk to the device.

Return Value

LS_OKAY if successful
LS_TRY_TO_RESET if the peripheral is in error state otherwise standard reply code.
LS_PERIPHERAL_NOT_FOUND

Comments

This function call must be invoked to establish the communication link with the specific LS device.

With the Ethernet models (*LSUnit* parameter set to LS_40_LSCONNECT or LS_100_LSCONNECT or LS_100_ETH or LS_150_LSCONNECT or LS_515_LSCONNECT) the application **must call** the LSSetIPAddress *()* function prior to the *LSConnect()* .

Backwards compatibility for LS100 applications using now LS150 :
To substitute a LS100 with a LS150 without the need to make any changes to the existing application, the library needs the support of the file LsApi.ini. This file must be present in the same folder as the Lsapi.dll.
Please refer to the section 3 for the details of this configuration file. (Parameter Ls150_as_Ls100)

Backwards compatibility for LS100 applications using now LS40 :
To substitute a LS100 with a LS40 without the need to make any changes to the existing application, the library needs the support of the file LsApi.ini. This file must be present in the same folder as the Lsapi.dll.
Please refer to the section 3 for the details of this configuration file. (Parameter Ls40_as_Ls100)

4.2. LSConnectWithNetworkName

#include "LSApi.h"

```
Result API LSConnectWithNetworkName(HWND           hWnd,
                                    HANDLE         hInst,
                                    short          LsUnit,
                                    char          * lpBoxName,
                                    unsigned short NetPort,
                                    short          *hConnect);
```

Description

Open a connection between the application and the LS service.
The function returns a connection handle.
For use this function on the LSConnect MUST be installed the Samba service.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hInst

Hinstance of the application window **(Reserved for future use)**.

LsUnit

Specifies the device that must be connected, the accepted values are :

- LS_40_LSCONNECT
- LS_100_LSCONNECT
- LS_100_ETH
- LS_150_LSCONNECT
- LS_515_LSCONNECT
- LS_UNIT_SHARED

lpBoxName

IP Adddress of the peripheral to connect.

NetPort

Port to connect (fix to 4000 for the moment).

hConnect

Connection handle returned by the service, this value shall be used in the most of the other functions that talk to the device.

Return Value

LS_OKAY if successful
LS_TRY_TO_RESET if the peripheral is in error state otherwise standard reply code.
LS_PERIPHERAL_NOT_FOUND

Comments

This function is used to connect the Ethernet models without the need to call the LSSetIPAddress() function.

4.3. LSConnectWithIPAddress

#include "LSApi.h"

```
Result API LSConnectWithIPAddress  (HWND           hWnd,
                                    HANDLE         hInst,
                                    short          LsUnit,
                                    char          * IpAddress,
                                    unsigned short NetPort,
                                    short          *hConnect);
```

Description

Open a connection between the application and the LS service.
The function returns a connection handle.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hInst

Hinstance of the application window **(Reserved for future use)**.

LsUnit

Specifies the device that must be connected, the accepted values are :

- LS_40_LSCONNECT
- LS_100_LSCONNECT
- LS_100_ETH
- LS_150_LSCONNECT
- LS_515_LSCONNECT
- LS_UNIT_SHARED

IpAddress

IP Adddress of the peripheral to connect.

NetPort

Port to connect (fix to 4000 for the moment).

hConnect

Connection handle returned by the service, this value shall be used in the most of the other functions that talk to the device.

Return Value

LS_OKAY if successful
LS_TRY_TO_RESET if the peripheral is in error state otherwise standard reply code.
LS_PERIPHERAL_NOT_FOUND

Comments

This function is used to connect the Ethernet models without the need to call the LSSetIPAddress() function.

4.4. LSDisconnect

```
#include "LSApi.h"

Result API LSDisconnect (SHORT      hConnect,
                         HWND       hWnd);
```

Description

Close a connection between a client application and LS service.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Return Value

LS_OKAY if successful, otherwise standard reply code
LS_PERIPHERAL_NOT_FOUND

Comments

4.5. LSSetNetworkName

```
#include "LSApi.h"

Result API LSSetNetworkName( short              LsUnit,
                             char             * New_Node_Name,
                             unsigned short New_Net_Port);
```

Description

Set the network logical name of the LSConnect to connect.

For use this function on the LSConnect MUST be installed the Samba service.

Parameters

LsUnit

Specifies the device that must be connected, the accepted values are :

- LS_40_LSCONNECT
- LS_100_LSCONNECT
- LS_100_ETH
- LS_150_LSCONNECT
- LS_515_LSCONNECT
- LS_UNIT_SHARED

New_Eth_IpAddress

Network node logical name.

New_Net_Port

Port to connect (fix to 4000 at the moment).

Return Value

LS_OKAY

LS_PERIPHERAL_NOT_FOUND

Comments

4.6. LSSetIPAddress

#include "LSApi.h"

```
Result API LSSetIPAddress (short          LsUnit,
                           char         * New_Eth_IpAddress,
                           unsigned short New_Net_Port);
```

Description

Set the IP Address of the LSConnect unit to connect.

Parameters

LsUnit

Specifies the device that must be connected, the accepted values are :

- LS_40_LSCONNECT
- LS_100_LSCONNECT
- LS_100_ETH
- LS_150_LSCONNECT
- LS_515_LSCONNECT
- LS_UNIT_SHARED

New_Eth_IpAddress

IP Adddress of the LSConnect unit to connect.

New_Net_Port

Port to connect (fix to 4000 at the moment).

Return Value

LS_OKAY
LS_PERIPHERAL_NOT_FOUND

Comments

4.7. LSGetVersion

#include "LSApi.h"

```
Result API LSGetVersion(CHAR     * VersionLibrary,
                        SHORT    LengthStr);
```

Description

Return the version of the library.

Parameters

VersionLibrary

This string contains the release of the driver library. The length of this string is 64 bytes maximum.

LengthStr

Length of the string returned in *VersionLibrary*.

Return Value

LS_OKAY if successful.
LS_STRING_TRUNCATED

Comments

4.8. LSUnitConfiguration

#include "LSApi.h"

```
Result API LSUnitConfiguration(  short                    hConnect,
                                 HWND                     hWnd,
                                 LPSTR                    pReserved,
                                 UNITCONFIGURATION        *DeviceFeatures,
                                 LPSTR                    LsModel,
                                 LPSTR                    Fw_Version,
                                 LPSTR                    Fw_Date,
                                 LPSTR                    PeripheralID,
                                 LPSTR                    BoardVersion,
                                 LPSTR                    DecoderExpVersion,
                                 LPSTR                    InkJetVersion,
                                 LPSTR                    FeederVersion,
                                 LPSTR                    SorterVersion,
                                 LPSTR                    MotorVersion,
                                 LPSTR                    Reserved1,
                                 LPSTR                    Reserved2);
```

Description

Returns specific information about hardware configuration and capabilities of the LS device connected to the service.
The *Size* field of the struct UNITSTRUCTCONFIGURATION give as pointer must be compiled with the size of function from the caller application.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

pReserved

Reserved, must be set to NULL.

DeviceFeatures

Pointer to a struct the return the features of the unit connected. The structure is described in the comments section.

LsModel

Pointer to a string that will contain the peripheral model, 15 bytes long, maximum.

FW_Version

Pointer to a string that will contain the firmware revision of the peripheral, 10 bytes long, maximum.

FW_Date

Pointer to a string that will contain the firmware date of the peripheral, 10 bytes long, maximum.

PeripheralID

Pointer to a string that will contain the internal peripheral serial number, 12 bytes long, maximum.

BoardVersion

Pointer to a string that will contain the internal board version, 4 bytes long, maximum.

DecoderExpVersion

Pointer to a string that will contain the firmware revision of the decoder board expansion, 10 bytes long, maximum.

InkJetVersion

Pointer to a string that will contain the firmware revision of the inkjet board expansion, 10 bytes long, maximum.

FeederVersion

Pointer to a string that will contain the firmware revision of the feeder connected, 10 bytes long maximum.

SorterVersion

Pointer to a string that will contain the firmware revision of the sorter/s connected, 10 bytes long maximum, all the sorters connected must have the same firmware, otherwise this field contain the character ?.
Parameter returned only in case of LS800 model.

MotorVersion

Pointer to a string that will contain the firmware revision of the motor, 10 bytes long maximum.
Parameter returned only in case of LS800 model.

Reserved1

Reserved for future use, must be set to NULL.

Reserved2

Reserved for future use, must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_RESERVED_ERROR

Comments

This function can be used only after a successful **LSConnect** command.

The application must compile the Size fileld with the size of the structure.

UNITCONFIGURATION.Size = sizeof(UNITCONFIGURATION);

Description UNITSTATUS structure :

```
typedef struct _UNITCONFIGURATION
{
    int      Size;                      // Size of the structure

    BOOL     MICR_Reader;               //      Ls100 Ls150 Ls515 Ls800
    BOOL     CMC7_Reader_only;          //      Ls100 Ls150 Ls515 Ls800
    BOOL     E13B_Reader_only;          //      Ls100 Ls150 Ls515 Ls800
    BOOL     Scanner_Front;             //      Ls100 Ls150 Ls515 Ls800
    BOOL    Scanner_Rear;             //     Ls100 Ls150 Ls515 Ls800
    BOOL    InkJet_Printer;           //     Ls100 Ls150 Ls515 Ls800
    BOOL    InkJet_HD_Printer_4_lines;//            Ls150 Ls515 Ls800
    BOOL    Feeder;                   //     Ls100
    BOOL    Double_Leafing_sensor;    //                  Ls515
    BOOL    Voiding_Front_Stamp;      //     Ls100       Ls515
    BOOL    Voiding_Rear_Stamp;       //                Ls515
    BOOL    No_Blanks;                //                Ls515
    BOOL    Badge_Track123;           //     Ls100 Ls150
    BOOL    Badge_Track12;            //     Ls100 Ls150 Ls515
    BOOL    Badge_Track23;            //     Ls100 Ls150 Ls515
    BOOL    OCR_Reader;               //     Ls100
    int     Sorters_Nr;               //                        Ls800
    BOOL    Module_Encoder;           //                        Ls800
    BOOL     Process_Card;            //Ls40
    BOOL     Capacitor;               //Ls40
    BOOL    Scanner_UltraViolet;      //        Ls150 Ls515
    BOOL     Scanner_Color;           //        Ls150
    BOOL     Hight_Speed;             //        Ls150
    BOOL    Feeder_Motorized;         //        Ls150
    BOOL    Feeder_Electromagnet;     //        Ls150
    BOOL    ID_Card_Color;            //        Ls150
    BOOL    License_ClearPIX;         //Ls40 Ls100 Ls150 Ls515
    BOOL    License_2D_Barcode;       //Ls40 Ls100 Ls150 Ls515
    BOOL    License_IQA;              //Ls40 Ls100 Ls150 Ls515
    BOOL    License_Micro_Hole;       //Ls40 Ls100 Ls150 Ls515
    BOOL    Ghost_Reactor;            //     Ls100
```

} UNITCONFIGURATION, *PUNITCONFIGURATION;

4.9. LSUnitReserve

#include "LSApi.h"

Result API LSUnitReserve (**short** *hConnect,*
HWND *hWnd*,
long *Timeout*);

Description

Reserve the device for the calling application.
This function must be called when an LS device, connected via USB to the PC, can be used from more than one application.
This function can also be used when the device is shared among different PCs.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Timeout

Timeout value expressed in milliseconds, only with USB connections for –1 the function wait indefinitely.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_INVALID_TIMEOUT
LS_PERIPHERAL_RESERVED

Comments

An application must call this function to reserve the device before issuing the following commands: *LSIdentify()*, *LSReset()* or *LSDocHandle()* plus *LSReadImage()* or LSAutoDocHandle() plus *LSGetDocData()* .
When those operations are finished, the application must call the *LSUnitRelease()* to allow another application to use the device.
It is important that the application reserve the device before feeding the checks and releases it when the boundle of checks is finished.

4.10. LSUnitRelease

#include "LSApi.h"

Result API LSUnitRelease (**short** *hConnect,*
HWND *hWnd);*

Description

The calling application release the peripheral when the current operation is finished.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_ILLEGAL_REQUEST

Comments

4.11. LSDocHandle

#include "LSApi.h"

```
Result API LSDocHandle (   short           hConnect,
                           HWND            hWnd,
                           short           Stamp,
                           short           Stamp,
                           short           Validate,
                           short           CodeLine,
                           char            Side,
                           short           ScanMode,
                           short           Feeder,
                           short           Sorter,
                           short           WaitTimeout,
                           short           Beep,
                           unsigned long   *NrDoc,
                           short           ScanDocType,
                           long            Reserved2);
```

Description

The behavior of this command will be dependent on the model of LS device employed. It will set the LS device in a waiting-for-document-introduction state, for models where documents are manually fed one at a time. It will activate the automatic feeding from an input feeder, for models where documents are processed in a bundle. After being drawn in the document is processed. The document processing will be done according to the options specified in the input parameters. The device will store in its internal memory the scanned images and MICR codeline of the last processed document (when the relevant parameters in the command are set for document scanning and/or MICR codeline reading). In order to retrieve the image(s) of the processed document the application must use the **LSReadImage** command, and the **LSReadCodeline** command is likewise required to retrieve the MICR codeline data.
Please also refer to the LSConfigDoubleLeafingAndDocLength() function to configure the paper sensibility in the correct way to handle the double leafing functionality.

NOTE : LS100 model does NOT support the 300 dpi resolution.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Stamp

NO_FRONT_STAMP = do not stamp document
FRONT_STAMP = stamp front side of document
REAR_STAMP = stamp rear side of document
FRONT_AND_REAR_STAMP = stamp both front and rear sides of document (this option is possible only with LS5xx models)

Validate

NO_PRINT_VALIDATE = do not print validation string
PRINT_VALIDATE = print validation string (the text of the validation string must be loaded using the ***LSLoadString*** command before invoking this command). The LS device must be equipped with the optional validation printer.

PRINT_DIGITAL_VALIDATE = print digital validation string (the text of the validation string must be loaded using the ***LSLoadDigitalStringWithCounter*** command before invoking this command).

Codeline

NO_READ_CODELINE = do not read codeline.
READ_CODELINE_MICR = read magnetic codeline CMC7 or E13B. The LS device must be equipped with a magnetic character reader.
READ_CODELINE_E13B_MICR_AND_OCR = read a codeline E13B in magnetic / optic mode
READ_CODELINE_CMC7_MICR_AND_OCR = read a codeline CMC7 in magnetic / optic mode
READ_CODELINE_MICRO_HOLES = read a MicroHoles codeline present on the document. **IMPORTANT** if this define is set in order to obtain Micro codelines, then the application must call the function *LSReadImageMH()* instead of *LSReadImage()*.
READ_CODELINE_MICR_AND_MICRO_HOLES = read a codeline CMC7 or E13B and the MicroHoles codeline present on the document. **IMPORTANT** if this define is set in order to obtain Micro codelines, then the application must call the function *LSReadImageMH()* instead of *LSReadImage()*.
READ_CODELINE_CMC7_MOCR_AND_MICRO_HOLES = read a codeline CMC7 in magnetic / optic mode and the MicroHoles codeline present on the document. **IMPORTANT** if this define is set in order to obtain Micro codelines, then the application must call the function *LSReadImageMH()* instead of *LSReadImage()*.

The following values are applicable when the LS device is equipped with an optional barcode reader or optical reader :
READ_BARCODE_HW = read a 2 of 5 barcode (LS510 model only)

Side

It specifies which side(s) of the document to scan
SIDE_FRONT_IMAGE = Scan Front side
SIDE_BACK_IMAGE = Scan Rear side
SIDE_ALL_IMAGE = Scan Both sides
SIDE_NONE_IMAGE = Do not scan document

ScanMode

This parameter sets the resolution applied when scanning the document. Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 grey scale at 100 dpi
SCAN_MODE_16GR120 = 16 grey scale at 120 dpi
SCAN_MODE_16GR200 = 16 grey scale at 200 dpi
SCAN_MODE_16GR240 = 16 grey scale at 240 dpi
SCAN_MODE_16GR300 = 16 grey scale at 300 dpi
SCAN_MODE_256GR100 = 256 grey scale at 100 dpi
SCAN_MODE_256GR120 = 256 grey scale at 120 dpi
SCAN_MODE_256GR200 = 256 grey scale at 200 dpi
SCAN_MODE_256GR240 = 256 grey scale at 240 dpi
SCAN_MODE_256GR300 = 256 grey scale at 300 dpi
SCAN_MODE_256GR100_AND_UV = 256 gray scale at 100 dpi and Ultra Violet images
SCAN_MODE_256GR200_AND_UV = 256 gray scale at 200 dpi and Ultra Violet images
SCAN_MODE_256GR300_AND_UV = 256 gray scale at 300 dpi and Ultra Violet images
SCAN_MODE_COLOR_100 = Color 24 bit 100 dpi
SCAN_MODE_COLOR_200 = Color 24 bit 200 dpi
SCAN_MODE_COLOR_300 = Color 24 bit 300 dpi
SCAN_MODE_COLOR_100_AND_UV = 256 gray scale at 100 dpi and Ultra Violet images
SCAN_MODE_COLOR_200_AND_UV = 256 gray scale at 200 dpi and Ultra Violet images
SCAN_MODE_COLOR_300_AND_UV = 256 gray scale at 300 dpi and Ultra Violet images

SCAN_MODE_256GR100BN = 256 gray scale at 100 dpi Brutto and Netto images (LS515 model only)
SCAN_MODE_256GR200BN = 256 gray scale at 200 dpi Brutto and Netto images (LS515 model only)
SCAN_MODE_COLOR_AND_RED_100 = Color 24 bit at 100 dpi and Netto images (LS100 and LS515 model)
SCAN_MODE_COLOR_AND_RED_200 = Color 24 bit at 200 dpi and Netto images (LS100 and LS515 model)
SCAN_MODE_256GR100_ONLY_RED = 256 gray scale at 100 dpi Red images
SCAN_MODE_256GR200_ONLY_RED = 256 gray scale at 200 dpi Red images
SCAN_MODE_256GR300_ONLY_RED = 256 gray scale at 300 dpi Red images

Feeder

It specifies the source of the document
AUTO_FEED = document from feeder.
PATH_FEED = document from path. This value should be used when the document processing requires two or more passes. It requires that in the previous pass the value set for *Sorter* parameter is HOLD_DOCUMENT. Used also for Linear Entry on LS5xx series.

Sorter

It specifies the destination of the processed document. The applicable values depend on the model of LS device being used :
HOLD_DOCUMENT = hold document. In the next pass the parameter *Feeder* must take the value PATH_FEED
SORTER_BAY1 = document is stacked in sorting pocket 1.
SORTER_BAY2 = document is stacked in sorting pocket 2
SORTER_AUTOMATIC = send the document to the destination set by a previous LSSetSorterCriteria command (on LS510, LS515 models).
SORTER_SWITCH_1_TO_2 = documents are sorted in Pocket 1, until it becomes full at which point sorting continues to Pocket 2. When Pocket 2 becomes full the service will return a LS_SORTERS_BOTH_FULL error code.
EJECT_DOCUMENT = the document is returned to the input feeding slot. (LS40 and LS100 models)

WaitTimeout

This parameter sets the behavior of the device when there are no more documents to process.
WAIT_YES = the device waits for approximately 7 seconds for a new document to process, when this internal timer expires and there is not a new document to process the command completes with return code LS_FEEDER_EMPTY .
WAIT_NO   = if no document is present, or after the last document in the feeder has been processed, the service will immediately complete with return code LS_FEEDER_EMPTY.

Beep

Specifies whether the internal beeper should emit an acoustical sound when an error occurs. Accepted values are :
NO_BEEP : do not activate beeper.
BEEP : activate beeper.

NrDoc

Required ONLY for LS100 Models otherwise can be set to NULL.
This is an output parameter that will be set when the command processing completes.
It returns the number of processed documents. The numbers in the range 1 to NrDoc must be used as ID’s to point to a given document’s data when retrieving the scanned image(s) (by means of LSReadImage).

ScanDocType

Specifies the type of document to processed.
Accepted values are :
SCAN_PAPER_DOCUMENT : for paper.
SCAN_CARD : for card.
SCAN_LONG_DOCUMENT : for receipt, sales check.
SCAN_A4_DOCUMENT : for scan document A4 (only for LS150 G).

Reserved2
Reserved for future use, must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_COMMAND_IN_EXECUTION_YET
LS_DOUBLE_LEAFING_ERROR
LS_DOUBLE_LEAFING_WARNING
LS_REPLACE_CARTRIDGE

Comments

Note :
If the application is requesting to read MicroHole codeline, the function will force the resolution at 300 dpi. The image returned, however, will be the one requested by the application.

This function is not available on the LS800.

4.12. LSReadCodeline

#include "LSApi.h"

```
Result API LSReadCodeline (   short    hConnect,
                              HWND     hWnd,
                              LPSTR    Codeline,
                              SHORT    *Length_Codeline,
                              LPSTR    Barcode,
                              SHORT    *Length_Barcode,
                              LPSTR    Optic,
                              SHORT    *Length_Optic);
```

Description

This function should be used to retrieve the codeline data read from a document previously processed by **LSDocHandle** command. It applies only to LS device models equipped with the relevant hardware (MICR reader, Barcode reader, OCR reader) for codeline reading (check LS device capabilities).

NOTE : This function must be called before the **LSReadImage** or **LSReadImageMH**.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Codeline

Pointer to the user buffer where the magnetic codeline data will be transferred by the service. This buffer will be filled also in case Magnetic and Optical is requested.

Length_Codeline

Pointer to a variable of input/output, in input it specifies the size of the buffer pointed by *Codeline* parameter, in output it will return the actual length of the codeline data transferred by the service.

Barcode

Pointer to the user buffer where the barcode codeline data will be transferred by the service.

Length_Barcode

Pointer to a variable of input/output, in input it specifies the size of the buffer pointed by *Barcode* parameter, in output it will return the actual length of the codeline data transferred by the service.

Optic

Pointer to the user buffer where the optical codeline data will be transferred by the service.

Length_Optic

Pointer to a variable of input/output, in input it specifies the size of the buffer pointed by *Optic* parameter, in output it will return the actual length of the codeline data transferred by the service.

Return Value

LS_OKAY

LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_DATA_TRUNCATED
LS_DATA_LOST

Comments

When the service returns more than one codeline there will be a NULL character as separator between one codeline data and the next.
This function is not available on the LS800 model.

4.13. LSReadImage

#include "LSApi.h"

```
Result API LSReadImage (    short              hConnect,
                            HWND               hWnd,
                            short              ClearBlack,
                            CHAR               Side,
                            short              ReadMode,
                            unsigned long      NrDoc,
                            LPHANDLE           FrontImage,
                            LPHANDLE           BackImage,
                            LPHANDLE           FrontImage2,
                            LPHANDLE           BackImage2);
```

Description

This function should be used to retrieve the images of the front and/or back side of a document previously processed and scanned by **LSDocHandle** command.

Parameters

hConnect

Handle returned by LSConnect.

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

ClearBlack

Specifies whether or not to apply the filter for cleaning the black area around the image(s).
NO_CLEAR_BLACK = no image cleaning, the black area is not removed.
CLEAR_ALL_BLACK = clean the document's image removing all the black around the image according to the level of filter specified by **LSSetThresholdClearBlack** command.
CLEAR_AND_ALIGN_IMAGE = removing all the black around the image and align the document's image. (deskew)

Side

Specifies which of the document's scanned sides to return.
SIDE_FRONT_IMAGE = Front side image
SIDE_BACK_IMAGE = Rear side image
SIDE_ALL_IMAGE = Both Front and Rear side images

ReadMode

Fixed to **READMODE_BRUTTO.**

NrDoc

Required ONLY for LS100 models, otherwise can be set to 0.
This is a value that must be grater than zero and less than or equal to the *NrDoc* value returned by **LSDocHandle** command. It specifies the sequence number for multiple documents processed by **LSDocHandle**.

FrontImage

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in DIB format.

BackImage

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in DIB format.

FrontImage2

Pointer to a handle where will be returned the handle of memory buffer containing the **Ultra Violet** front side image of the requested document, in DIB format.

BackImage2

Reserved for future improvement, must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_OPEN_NOT_DONE
LS_COMMAND_SEQUENCE_ERROR
LS_INVALID_TYPE_COMMAND
LS_INVALID_CLEARBLACK
LS_INVALID_SIDE

Comments

The image handle retrieved by this function must be released by the application.

This function is not available on the LS800 model.

4.14. LSReadImageMH

#include "LSApi.h"

```
Result API LSReadImageMH ( short                  hConnect,
                           HWND                   hWnd,
                           short                  ClearBlack,
                           CHAR                   Side,
                           short                  ReadMode,
                           unsigned long          NrDoc,
                           LPHANDLE               FrontImage,
                           LPHANDLE               BackImage,
                           LPHANDLE               FrontImage2,
                           LPHANDLE               BackImage2,
                           BOOL                   VerifyHole,
                           short                  UnitMeasure,
                           short                  nrRegions,
                           MICROHOLE_STRUCT       stMicroHole);
```

Description

This function should be used to retrieve the images of the front and/or back side of a document previously processed and scanned by **LSDocHandle()** function, but return also the MicroHole codelines, if the parameter *Codeline* is set to **READ_CODELINE_MICR_AND_MICRO_HOLES** or **READ_CODELINE_CMC7_MOCR_AND_MICRO_HOLES**.

Parameters

hConnect

Handle returned by LSConnect.

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

ClearBlack

Specifies whether or not to apply the filter for cleaning the black area around the image(s).
NO_CLEAR_BLACK = no image cleaning, the black area is not removed.
CLEAR_ALL_BLACK = clean the document's image removing all the black around the image according to the level of filter specified by **LSSetThresholdClearBlack** command.
CLEAR_AND_ALIGN_IMAGE = removing all the black around the image and align the document’s image. (deskew)

Side

Specifies which of the document’s scanned sides to return.
SIDE_FRONT_IMAGE = Front side image
SIDE_BACK_IMAGE = Rear side image
SIDE_ALL_IMAGE = Both Front and Rear side images

ReadMode

Fixed to **READMODE_BRUTTO.**

NrDoc

Required ONLY for LS100 models, otherwise can be set to 0.
This is a value that must be grater than zero and less than or equal to the *NrDoc* value returned by **LSDocHandle** command. It specifies the sequence number for multiple documents processed by **LSDocHandle**.

FrontImage

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in DIB format.

BackImage

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in DIB format.

FrontImage2

Pointer to a handle where will be returned the handle of memory buffer containing the **Ultra Violet** front side image of the requested document, in DIB format.

BackImage2

Reserved for future improvement, must be set to NULL.

VerifyHole

Specify if verification of micro perforation of the paper is required.

UnitMeasure

Specify whether the *Start_X, Start_Y, SizeW* and *SizeH* measures are expressed in millimeters or in inches.
The possible values are either **UNIT_MM** or **UNIT_INCH**.

nrRegions

Number of codeline to decode.

stMicroHole

Array of structure with parameters per each region, described in **Comments** section of the function *LSReadMicroHolesCodelines()*.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_OPEN_NOT_DONE
LS_COMMAND_SEQUENCE_ERROR
LS_INVALID_TYPE_COMMAND
LS_INVALID_CLEARBLACK
LS_INVALID_SIDE

Comments

The image handle retrieved by this function must be released by the application.

4.15. LSDocHandleAndReadImage

#include "LSApi.h"

```
Result API LSDocHandleAndReadImage (short        hConnect,
                                    HWND         hWnd,
                                    short        Stamp,
                                    short        Validate,
                                    short        CodeLine,
                                    char         Side,
                                    short        ScanMode,
                                    short        Feeder,
                                    short        Sorter,
                                    short        WaitTimeout,
                                    short        Beep,
                                    short        ScanDocType,
                                    long         Reserved2,
                                    short        ClearBlack,
                                    LPHANDLE     FrontImage,
                                    LPHANDLE     BackImage,
                                    LPHANDLE     FrontImage2,
                                    LPHANDLE     BackImage2);
```

Description

This command MUST be called after a command of **LSDocHandle**, in place of a **LSReadImage**, but after a **LSReadCodeline**, if the MICR codeline if required.
The behavior of this command it's alike to do a **LSReadImage** with a **LSDocHandle**, but optimize the performance of the device, because before to read the image of the previous handled document it to start another one.
Remark, when the function return with a reply different from LS_OKAY, the application **must** call one time the function **LSReadImage** for keep the images of the last document scanned.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Stamp

NO_FRONT_STAMP = do not stamp document
FRONT_STAMP = stamp front side of document
REAR_STAMP = stamp rear side of document
FRONT_AND_REAR_STAMP = stamp both front and rear sides of document (this option is possible only with LS5xx models)

Validate

NO_PRINT_VALIDATE = do not print validation string
PRINT_VALIDATE = print validation string (the text of the validation string must be loaded using the ***LSLoadString*** command before invoking this command). The LS device must be equipped with the optional validation printer.
PRINT_DIGITAL_VALIDATE = print digital validation string (the text of the validation string must be loaded using the ***LSLoadDigitalStringWithCounter*** command before invoking this command).

Codeline

NO_READ_CODELINE = do not read codeline.
READ_CODELINE_MICR = read magnetic codeline CMC7 or E13B. The LS device must be equipped with a magnetic character reader.
READ_CODELINE_E13B_MICR_AND_OCR = read a codeline E13B in magnetic / optic models (LS40 LS100/7 LS150)

The following values are applicable when the LS device is equipped with an optional barcode reader or optical reader :
READ_BARCODE_HW = read a 2 of 5 barcode (LS510 model only)

Side

It specifies which side(s) of the document to scan
SIDE_FRONT_IMAGE = Scan Front side
SIDE_BACK_IMAGE = Scan Rear side
SIDE_ALL_IMAGE = Scan Both sides
SIDE_NONE_IMAGE = Do not scan document

ScanMode

This parameter sets the resolution applied when scanning the document. Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 grey scale at 100 dpi
SCAN_MODE_16GR120 = 16 grey scale at 120 dpi
SCAN_MODE_16GR200 = 16 grey scale at 200 dpi
SCAN_MODE_16GR240 = 16 grey scale at 240 dpi
SCAN_MODE_16GR300 = 16 grey scale at 300 dpi (LS40, LS150 and LS520 model)
SCAN_MODE_256GR100BN = 256 gray scale at 100 dpi Brutto and Netto images (LS515 model only)
SCAN_MODE_256GR200BN = 256 gray scale at 200 dpi Brutto and Netto images (LS515 model only)
SCAN_MODE_256GR100 = 256 grey scale at 100 dpi
SCAN_MODE_256GR120 = 256 grey scale at 120 dpi
SCAN_MODE_256GR200 = 256 grey scale at 200 dpi
SCAN_MODE_256GR240 = 256 grey scale at 240 dpi
SCAN_MODE_256GR300 = 256 grey scale at 300 dpi (LS40, LS150 and LS520 model)
SCAN_MODE_256GR100_AND_UV = 256 gray scale at 100 dpi and Ultra Violet images (LS150 model)
SCAN_MODE_256GR200_AND_UV = 256 gray scale at 200 dpi and Ultra Violet images (LS150 model)
SCAN_MODE_COLOR_100 = Color 24 bit 100 dpi
SCAN_MODE_COLOR_200 = Color 24 bit 200 dpi
SCAN_MODE_COLOR_300 = Color 24 bit 300 dpi (LS150 and LS520 model)
SCAN_MODE_COLOR_AND_RED_100 = Color 24 bit at 100 dpi and Netto images (LS100 and LS515 model)
SCAN_MODE_COLOR_AND_RED_200 = Color 24 bit at 200 dpi and Netto images (LS100 and LS515 model)
SCAN_MODE_256GR100_ONLY_RED = 256 gray scale at 100 dpi Red images
SCAN_MODE_256GR200_ONLY_RED = 256 gray scale at 200 dpi Red images
SCAN_MODE_256GR300_ONLY_RED = 256 gray scale at 300 dpi Red images

Feeder

It specifies the source of the document
AUTO_FEED = document from feeder.
PATH_FEED = document from path. This value should be used when the document processing requires two or more passes. It requires that in the previous pass the value set for *Sorter* parameter is HOLD_DOCUMENT. Used also for Linear Entry on LS5xx series.

Sorter

It specifies the destination of the processed document. The applicable values depend on the model of LS device being used :

HOLD_DOCUMENT = hold document. In the next pass the parameter *Feeder* must take the value PATH_FEED

SORTER_BAY1 = document is stacked in sorting pocket 1.

SORTER_BAY2 = document is stacked in sorting pocket 2

SORTER_SWITCH_1_TO_2 = documents are sorted in Pocket 1, until it becomes full at which point sorting continues to Pocket 2. When Pocket 2 becomes full the service will return a LS_SORTERS_BOTH_FULL error code.

WaitTimeout

This parameter sets the behavior of the device when there are no more documents to process.

WAIT_YES = the device waits for approximately 7 seconds for a new document to process, when this internal timer expires and there is not a new document to process the command completes with return code LS_FEEDER_EMPTY .

WAIT_NO = if no document is present, or after the last document in the feeder has been processed, the service will immediately complete with return code LS_FEEDER_EMPTY.

Beep

Specifies whether the internal beeper should emit an acoustical sound when an error occurs. Accepted values are :

NO_BEEP : do not activate beeper

BEEP : activate beeper

ScanDocType

Specifies the type of document to processed.

Accepted values are :

SCAN_PAPER_DOCUMENT : for paper

SCAN_CARD : for card

SCAN_LONG_DOCUMENT : for receipt, sales check

Reserved2

Reserved for future use, must be set to NULL.

ClearBlack

Specifies whether or not to apply the filter for cleaning the black area around the image(s).

NO_CLEAR_BLACK = no image cleaning, the black area is not removed.

CLEAR_ALL_BLACK = clean the document's image removing all the black around the image according to the level of filter specified by **LSSetThresholdClearBlack** command.

CLEAR_AND_ALIGN_IMAGE = removing all the black around the image and align the document’s image. (deskew)

FrontImage

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in DIB format.

BackImage

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in DIB format.

FrontImage2

Pointer to a handle where will be returned the handle of memory buffer containing the **Ultra Violet** front side image of the requested document, in DIB format.

BackImage2

Reserved for future improvement, must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_COMMAND_IN_EXECUTION_YET
LS_DOUBLE_LEAFING_ERROR
LS_DOUBLE_LEAFING_WARNING

Comments

A little piece of “C” code for illustrate the behavior of the function.

```
NrDoc = 0;

Reply = LSDocHandle(hLS, hDlg,
                     NO_STAMP,
                     NO_PRINT_VALIDATE,
                     READ_CODELINE_MICR,
                     SIDE_ALL_IMAGE,
                     SCAN_MODE_BW,
                     AUTO_FEED,
                     SORTER_BAY1,
                     WAIT_YES,
                     NO_BEEP,
                     NULL,
                     0,
                     0);

while( Reply == LS150_OKAY )
{
       len_codeline = CODE_LINE_LENGTH;
       Reply = LSReadCodeline(hLS,
                              hDlg,
                              BufCodelineHW,
                              &len_codeline,
                              NULL,
                              NULL,
                              NULL,
                              NULL);

       // Here you can stop the loop if the codeline is invalid !

       // Here call the LSLoadSringh if you need to print different strings

       Reply = LSDocHandleAndReadImage(hLS, hDlg,
                                           NO_STAMP,
                                           NO_PRINT_VALIDATE,
                                           READ_CODELINE_MICR,
                                           SIDE_ALL_IMAGE,
                                           SCAN_MODE_BW,
                                           AUTO_FEED,
                                           SORTER_BAY1,
                                           WAIT_YES,
                                           NO_BEEP,
                                           0,
                                           0,
                                           CLEAR_ALL_BLACK,
                                           &BufFrontImage,
                                           &BufBackImage,
                                           NULL,
                                           NULL);

       if( Reply == LS_OKAY || Reply == LS_FEEDER_EMPTY )
       {
               NrDoc ++;

               // Show the codeline and images read
               ShowCodelineAndImage(LS_OKAY,     //Reply,
                                       0,
                                       NrDoc,
                                       (unsigned char *)BufFrontImage,
                                       (unsigned char *)BufBackImage,
                                       NULL, NULL,
                                       NULL,
                                       BufCodelineHW,
                                       NULL,
                                       NULL);

               LSFreeImage(hDlg, &BufFrontImage);
               LSFreeImage(hDlg, &BufBackImage);
       }
}

// When you finish the while read the last image for reply Ok or not

Reply = LSReadImage(hLS, hDlg,
               CLEAR_ALL_BLACK,
               SIDE_FRONT_IMAGE,
               0, 0,
               (LPHANDLE)&BufFrontImage,
               (LPHANDLE)&BufBackImage,
               NULL,
               NULL);
```

4.16. LSSetThresholdClearBlack

#include "LSApi.h"

Result API LSSetThresholdClearBlack (**short** *hConnect,*
HWND *hWnd*,
unsigned char *Threshold);*

Description

This function is used to set the desired threshold value for cleaning the black coloring that may result around the scanned image of the document. This value will be used by the **LSReadImage** command to apply the appropriate level of filter in cleaning the black area before returning the scanned image to the application.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Threshold

Value of black used as threshold for cleaning the image, range 0 (black) to 255 (white), the default value is **DEFAULT_BLACK_THRESHOLD**.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE

Comments

4.17. LSLoadString

#include "LSApi.h"

```
Result API LSLoadString(   short     hConnect,
                           HWND      hWnd,
                           char      Format,
                           short     Length,
                           LPSTR     String);
```

Description

Loads the text of invalidation string, which may be printed onto the **BACK** side of the document with the ink-jet printer.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Format

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font, low position.
PRINT_FORMAT_BOLD = Bold font, low position.
PRINT_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font, low position.
PRINT_UP_FORMAT_NORMAL = Normal font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_BOLD = Bold font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font , high position. (LS100, LS150 models)

Length

Length of the invalidation string

String

Invalidation string

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_FORMAT

Comments

Supported only by the peripheral LS models equipped with a backside invalidation printer.

4.18. LSLoadStringWithCounterEx

#include "LSApi.h"

```
Result API LSLoadStringWithCounterEx(  short            hConnect,
                                       HWND             hWnd,
                                       char             Format,
                                       LPSTR            String,
                                       short            Length,
                                       unsigned long    StartNumber,
                                       short            Step);
```

Description

Loads the invalidation string, which may be printed onto the back side of the document with the ink-jet printer.. It differs from **LSLoadString** in that it is possible to specify a starting number that will be automatically incremented or decremented for each new document.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Format

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font, low position.
PRINT_FORMAT_BOLD = Bold font, low position.
PRINT_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font, low position.
PRINT_UP_FORMAT_NORMAL = Normal font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_BOLD = Bold font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font , high position. (LS100, LS150 models)

String

Invalidation string having the same syntax of the C-language PRINTF function.

Length

Length of the string give in the *String* parameter.

StartNumber

Starting number used in the invalidation print numeration.

Step

When greater than zero it sets the increment step applied to *StartNumber* .
When less than zero it sets the decrement step applied to *StartNumber* .

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_SYSTEM_ERROR
LS_USB_ERROR

LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_FORMAT

Comments

Supported only by the peripheral LS models equipped with a backside invalidation printer.

4.19. LSLoadMixedString

#include "LSApi.h"

```
Result API LSLoadMixedString( short            hConnect,
                              HWND             hWnd,
                              unsigned short   Reserved1,
                              short            Reserved2,
                              char             Font1,
                              LPSTR            String1,
                              short            Length1);
                              char             Font2,
                              LPSTR            String2,
                              short            Length2);
                              char             Font3,
                              LPSTR            String3,
                              short            Length3);
                              char             Font4,
                              LPSTR            String4,
                              short            Length4);
```

Description

Loads the text of invalidation string, which may be printed onto the **BACK** side of the document with the ink-jet printer. It different from *LSLoadString()* that is possible print a string with some word in normal font mixed with some word in bold font.
Available only on LS150 model with Firmware 1.38 or higher.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Reserved1

Reserved for future use. Must be set to 0.

Reserved2

Reserved for future use. Must be set to 0.

Font1

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font, low position.
PRINT_FORMAT_BOLD = Bold font, low position.
PRINT_UP_FORMAT_NORMAL = Normal font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_BOLD = Bold font , high position. (LS100, LS150 models)

String1

First part of the invalidation string.

Length1

Length of the first part of the invalidation string.

Font2

Set the character style for the invalidation string,

PRINT_FORMAT_NORMAL = Normal font, low position.
PRINT_FORMAT_BOLD = Bold font, low position.
PRINT_UP_FORMAT_NORMAL = Normal font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_BOLD = Bold font , high position. (LS100, LS150 models)

String2
First part of the invalidation string.

Length2
Length of the first part of the invalidation string.

Font3
Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font, low position.
PRINT_FORMAT_BOLD = Bold font, low position.
PRINT_UP_FORMAT_NORMAL = Normal font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_BOLD = Bold font , high position. (LS100, LS150 models)

String3
First part of the invalidation string.

Length3
Length of the first part of the invalidation string.

Font4
Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font, low position.
PRINT_FORMAT_BOLD = Bold font, low position.
PRINT_UP_FORMAT_NORMAL = Normal font , high position. (LS100, LS150 models)
PRINT_UP_FORMAT_BOLD = Bold font , high position. (LS100, LS150 models)

String4
First part of the invalidation string.

Length4
Length of the first part of the invalidation string.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_FORMAT

Comments

Supported only by the peripheral LS models equipped with a backside invalidation printer.

For the part of string not needed set it to NULL, i.e. if my string is done of a part of normal and the other of bold, the parameters *Font3, String3, Length3, Font4, String4* and *Length4,* must be set to NULL.

4.20. LSLoadMultiStrings

#include "LSApi.h"

```
Result API LSLoadMultiStrings( short        hConnect,
                               HWND         hWnd,
                               char         Font1,
                               LPSTR        String1,
                               short        Length1);
                               char         Font2,
                               LPSTR        String2,
                               short        Length2);
                               char         Font3,
                               LPSTR        String3,
                               short        Length3);
                               char         Font4,
                               LPSTR        String4,
                               short        Length4);
```

Description

Loads up to 4 text lines of invalidation string, which may be printed onto the **BACK** side of the document with the ink-jet printer.
Available only on LS150 and LS515 model with the print header of 4 lines.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Font1

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font.
PRINT_FORMAT_BOLD = Bold font.
PRINT_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font.
PRINT_FORMAT_DOUBLE_HIGH = Font double high uses two line space.

String1

First part of the invalidation string.

Length1

Length of the first part of the invalidation string.

Font2

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font.
PRINT_FORMAT_BOLD = Bold font.
PRINT_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font.
PRINT_FORMAT_DOUBLE_HIGH = Font double high uses two line space.

String2

Second part of the invalidation string.

Length2

Length of the second part of the invalidation string.

Font3

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font.
PRINT_FORMAT_BOLD = Bold font.
PRINT_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font.
PRINT_FORMAT_DOUBLE_HIGH = Font double high uses two line space.

String3

Third part of the invalidation string.

Length3

Length of the third part of the invalidation string.

Font4

Set the character style for the invalidation string,
PRINT_FORMAT_NORMAL = Normal font.
PRINT_FORMAT_BOLD = Bold font.
PRINT_FORMAT_NORMAL_15_CHAR = Normal 15 char for Inch font.
PRINT_FORMAT_DOUBLE_HIGH = Font double high uses two line space.

String4

Forth part of the invalidation string.

Length4

Length of the forth part of the invalidation string.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_FORMAT

Comments

Supported only by the peripheral LS models equipped with a backside 4 lines invalidation printer.

Each string provided in String1 String2 String3 String4 parameters can also not be NULL terminated.
Each line can have a different font .
If a line must not be printed, set the corresponding string to NULL.

4.21. LSReset

#include "LSApi.h"

```
Result API LSReset (    short     hConnect,
                        HWND      hWnd,
                        char      ResetType);
```

Description

Tests whether the cause of the error of the previous command has been removed and resets the unit.

The function is also used for cleaning the belt of the unit.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

ResetType

Type of reset done :

RESET_ERROR – Reset software of the equipment.
RESET_FREE_PATH – Reset with turn on of the motors.
RESET_BELT_CLEANING – Turn on of the motors for cleaning the belt.
RESET_COUNTER_CARTRIDGE – Reset the counter of the dot printed.
RESET_UNIT_RESERVE – Reset the reservation of the unit.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_INVALID_RESET_TYPE

Comments

The RESET_COUNTER_CARTRIDGE will be called when a cartridge is replaced.

4.22. LSConfigDoubleLeafingAndDocLength

#include "LSApi.h"

```
Result API LSConfigDoubleLeafingAndDocLength( short    hConnect,
                                               HWND     hWnd,
                                               long     Type,
                                               short    Value,
                                               long     DocMin,
                                               long     DocMax);
```

Description

Configures the device in case of Double Leafing (when two documents are picked together instead of one). It is possible to set the type of behavior, disabled or enabled, when enabled the unit will return a warning and continue to read the documents, or it will return a blocking error and stop the document processing loop with the document in the path. It is also possible to change the sensibility level and the length of the overlapped area before returning the error/warning.
The setting done via this function will be valid until a power off-on of the device or a next double leafing setting command.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages.

Type

The value accepted are the follows:
DOUBLE_LEAFING_ERROR – Error on Double leafing (default).
DOUBLE_LEAFING_WARNING – Warning on Double leafing.
DOUBLE_LEAFING_DISABLE – Double Leafing sensor disabled.

Value

The range of value accepted are from 0 to 100, the default is 50.

DocMin

Length in millimeter of the minimun doc handled :
Possible values for LS100 : grater than 100
Possible values for LS150 : grater than 80
Possible values for LS515 : grater than 150
Possible values for LS800 : grater than 130

DocMax

Length in millimeter of the maximun doc handled :
Possible values for LS100 : less than 216
Possible values for LS150 : less than 320
Possible values for LS515 : less than 216
Possible values for LS800 : less than 216

Return Value

LS_OKAY
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM

LS_INVALID_TYPE
LS_INVALID_VALUE

Comments

This function, replaces the functions *LSConfigDoubleLeafingEx()* and *LSDoubleLeafingSensibility()*. With the LS40 this function return always LS_OKAY for compatibility reason because it doesn't have the Double Leafing sensor.

4.23. LSSetLightIntensity

#include "LSApi.h"

```
Result API LSSetLightIntensity (short        hConnect,
                                HWND         hWnd,
                                short        Value);
```

Description

Available only for LS40 and LS150 model.
Changes in a volatile mode the intensity scanner light, when the device is powered Off the value is lost, and the original default value will be set.

Parameters

hConnect
Handle returned by LSConnect

hWnd
Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Value
The Range of Value accepted is from 0 to 30.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_VALUE

Comments

4.24. LSSetUnitSpeed

#include "LSApi.h"

```
Result API LSSetUnitSpeed (    short       hConnect,
                               HWND        hWnd,
                               char        UnitSpeed);
```

Description

Available only for LS150 model.

Changes in a volatile mode the Unit speed. (does not write in the flash memory of the device) The Unit can accept 2 types of speed, 75 doc per minute and 150 doc per minute.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

UnitSpeed

Value accepted :
SPEED_NORMAL – Set the default Unit speed. (150DPM)
SPEED_STAMP – Set for all type of ScanMode the stamp Unit speed. (75 DPM)

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_UNIT_SPEED

Comments

4.25. LSUnitStatus

#include "LSApi.h"

```
Result API LSUnitStatus ( short          hConnect,
                          HWND           hWnd,
                          UNITSTATUS     *lpStatus);
```

Description

This function can be used by the application to obtain detailed information about the LS device sensors status.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

LpStatus

Pointer to a structure **UNITSTATUS** that describe the status and the photo of the unit. The structure is described in the comments section.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_COMMAND_SEQUENCE_ERROR

Comments

Description UNITSTATUS structure :

```
typedef struct _UNITSTATUS
{
    int      Size;                         // Size of the structure
    int      UnitStatus;                   // Ls40, Ls100, Ls150,Ls5xx and Ls800.
                                           // This reflects the internal status of the device
                                           // as described hereafter

    BOOL     Photo_Feeder;                 // Ls40   Ls100  Ls150  Ls5xx  Ls800
    BOOL     Photo_Sorter;                 //        Ls100
    BOOL     Photo_MICR;                   //        Ls100  Ls150
    BOOL     Photo_Path_Ls100;             //        Ls100
    BOOL     Photo_Trigger;                // Ls40
    BOOL     Photo_Scanners;               //        Ls100  Ls150
    BOOL     Unit_Just_ON;                 // Ls40   Ls100  Ls150
    BOOL     Photo_Double_Leafing_Down;    //        Ls100  Ls150
    BOOL     Photo_Double_Leafing_Middle;  //               Ls150
    BOOL     Photo_Double_Leafing_Up;      //        Ls100  Ls150
    BOOL     Document_Retained;            // Ls40
    BOOL    Photo_Card;                 //              Ls150
    BOOL    Pockets_All_Full;           //              Ls150  Ls5xx
    BOOL    Photo_Stamp;                //                     Ls5xx
    BOOL    Photo_Exit;                 //                     Ls5xx
    BOOL    Pocket_1_Full;              //                     Ls5xx
    BOOL    Pocket_2_Full;              //                     Ls5xx
    BOOL    Photo_Path_Feeder;          //                            Ls800
    BOOL    Photo_Path_Module_Begin;    //                            Ls800
    BOOL    Photo_Path_Binary_Rigth;    //                            Ls800
    BOOL    Photo_Path_Binary_Left;     //                            Ls800
    BOOL    Photo_Path_Module_End;      //                            Ls800
    BOOL    Sorter_1_input_pocket_1;    //                            Ls800
    BOOL    Sorter_1_pocket_1_full;     //                            Ls800
    BOOL    Sorter_1_input_pocket_2;    //                            Ls800
    BOOL    Sorter_1_pocket_2_full;     //                            Ls800
    BOOL    Sorter_1_input_pocket_3;    //                            Ls800
    BOOL    Sorter_1_pocket_3_full;     //                            Ls800
    BOOL    Sorter_2_input_pocket_1;    //                            Ls800
    BOOL    Sorter_2_pocket_1_full;     //                            Ls800
    BOOL    Sorter_2_input_pocket_2;    //                            Ls800
    BOOL    Sorter_2_pocket_2_full;     //                            Ls800
    BOOL    Sorter_2_input_pocket_3;    //                            Ls800
    BOOL    Sorter_2_pocket_3_full;     //                            Ls800
    BOOL    Sorter_3_input_pocket_1;    //                            Ls800
    BOOL    Sorter_3_pocket_1_full;     //                            Ls800
    BOOL    Sorter_3_input_pocket_2;    //                            Ls800
    BOOL    Sorter_3_pocket_2_full;     //                            Ls800
    BOOL    Sorter_3_input_pocket_3;    //                            Ls800
    BOOL    Sorter_3_pocket_3_full;     //                            Ls800
    BOOL    Sorter_4_input_pocket_1;    //                            Ls800
    BOOL    Sorter_4_pocket_1_full;     //                            Ls800
    BOOL    Sorter_4_input_pocket_2;    //                            Ls800
    BOOL    Sorter_4_pocket_2_full;     //                            Ls800
    BOOL    Sorter_4_input_pocket_3;    //                            Ls800
    BOOL    Sorter_4_pocket_3_full;     //                            Ls800
    BOOL    Sorter_5_input_pocket_1;    //                            Ls800
    BOOL    Sorter_5_pocket_1_full;     //                            Ls800
    BOOL    Sorter_5_input_pocket_2;    //                            Ls800
    BOOL    Sorter_5_pocket_2_full;     //                            Ls800
    BOOL    Sorter_5_input_pocket_3;    //                            Ls800
    BOOL    Sorter_5_pocket_3_full;     //                            Ls800
    BOOL    Sorter_6_input_pocket_1;    //                            Ls800
    BOOL    Sorter_6_pocket_1_full;     //                            Ls800
    BOOL    Sorter_6_input_pocket_2;    //                            Ls800
    BOOL    Sorter_6_pocket_2_full;     //                            Ls800
    BOOL    Sorter_6_input_pocket_3;    //                            Ls800
    BOOL    Sorter_6_pocket_3_full;     //                            Ls800
    BOOL    Sorter_7_input_pocket_1;    //                            Ls800
    BOOL    Sorter_7_pocket_1_full;     //                            Ls800
    BOOL    Sorter_7_input_pocket_2;    //                            Ls800
    BOOL    Sorter_7_pocket_2_full;     //                            Ls800
    BOOL    Sorter_7_input_pocket_3;    //                            Ls800
    BOOL    Sorter_7_pocket_3_full;     //                            Ls800

} UNITSTATUS, *PUNITSTATUS;
```

Note:
The sensor status is equal to TRUE, then the sensor is covered otherwise is equal to FALSE then the sensor is uncovered.

Possible value assuming from the field *UnitStatus* :

0 H	No sense
2 H	Unit busy
3 H	Paper jam
4 H	Hardware error
5 H	Illegal request
6 H	Document not present
9 H	Double Leafing error
B H	Aborted command
40 H	Jam at MICR photo
41 H	Jam Document to long
42 H	Jam at scanner photo

4.26. LSPeripheralStatus

#include "LSApi.h"

```
Result API LSPeripheralStatus (  short          hConnect,
                                 HWND           hWnd,
                                 unsigned char  *SenseKey,
                                 unsigned char  *SensorStatus);
```

Description

This function can be used by the application to obtain detailed information about the LS device sensors status.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

SenseKey

Pointer to a byte variable that will contain the sense key status.
Possible values are :

Hex	Description
0 H	No sense
2 H	Unit busy
3 H	Paper jam
4 H	Hardware error
5 H	Illegal request
6 H	Document not present
9 H	Invalid command
40 H	Jam at MICR photo
41 H	Jam Document to long
42 H	Jam at scanner photo

SensorStatus

Pointer to a 16 bytes variable that will contain the sensor status.
The information returned takes on different meanings for different LS models.
The bits must be interpreted in this order:
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
For example if the sensor bit has a value of 0x0E the following bits will be set:
Bit0 = 0
Bit1 = 1
Bit2 = 1
Bit3 = 1
Bit4 = 0
Bit5 = 0
Bit6 = 0
Bit7 = 0

The meaning of each bit is described underneath.

For LS40 models

SensorStatus[byte 0]

Bit	Description
0	Document present in the leafer
1	Photo head MICR covered
2	n.u.
3	n.u.
4	n.u.
5	n.u.
6	Document Retained
7	Unit just ON

For LS100 models

SensorStatus[byte 0]

Bit	Description
0	Document present in the leafer
1	bit = 0 the value of the previous bit is not valid, bit = 1 yes
2	Document present in the bin
3	bit = 0 the value of the previous bit is not valid, bit = 1 yes
4	Photo head MICR covered
5	n.u.
6	n.u.
7	Unit just ON

For LS150 models

SensorStatus[byte 0]

Bit	Description
0	Document present in the leafer
1	Photo head MICR covered
2	Photo card covered
3	Photo scanner covered
4	BIN full
5	n.u.
6	n.u.
7	Unit just ON

SensorStatus[byte 1]

Bit	Description
0	Photo 1 Double leafing covered
1	Photo 2 Double leafing covered
2	Photo 3 Double leafing covered
3	n.u.
4	n.u.
5	n.u.
6	n.u.
7	n.u.

For LS200 models

SensorStatus[byte 0]

Bit	Description
0	Document present in the leafer
1	Photo scanner covered
2	Photo head MICR covered
3	Photo BIN covered
4	Photo stamp covered
5	n.u.
6	n.u.
7	Unit just ON

SensorStatus[byte 1]

Bit	Description
0	n.u.
1	Photo 0 double leafing covered
2	Photo 1 double leafing covered
3	Photo 2 double leafing covered
4	n.u.
5	n.u.
6	n.u.
7	n.u.

For LS5xx series models

SensorStatus[byte 0]

Bit	Description
0	Document present in the leafer
1	Photo head MICR covered
2	Photo stamp covered
3	Photo scanner covered
4	Document retained
5	not used
6	not used
7	Unit just ON

SensorStatus[byte 1]

Bit	Description
0	not used
1	First BIN full
2	Second BIN full
3	Double leafing occurred
4	not used
5	not used
6	not used
7	not used

For LS800 series model

The bytes from 11 to 15 are reserved for future enhancements.

SensorStatus[byte 0]

Bit	Description
0	Sorter 1 - photo input pocket 1 covered
1	Sorter 1 - pocket 1 full
2	Sorter 1 - photo input pocket 2 covered
3	Sorter 1 - pocket 2 full
4	Sorter 1 - photo input pocket 3 covered
5	Sorter 1 - pocket 3 full
6	n.u.
7	n.u.

SensorStatus[byte 1]

Bit	Description
0	Sorter 2 - photo input pocket 1 covered
1	Sorter 2 - pocket 1 full
2	Sorter 2 - photo input pocket 2 covered
3	Sorter 2 - pocket 2 full
4	Sorter 2 - photo input pocket 3 covered
5	Sorter 2 - pocket 3 full
6	n.u.

7	n.u.

SensorStatus[byte 2]

Bit	Description
0	Sorter 3 - photo input pocket 1 covered
1	Sorter 3 - pocket 1 full
2	Sorter 3 - photo input pocket 2 covered
3	Sorter 3 - pocket 2 full
4	Sorter 3 - photo input pocket 3 covered
5	Sorter 3 - pocket 3 full
6	n.u.
7	n.u.

SensorStatus[byte 3]

Bit	Description
0	Sorter 4 - photo input pocket 1 covered
1	Sorter 4 - pocket 1 full
2	Sorter 4 - photo input pocket 2 covered
3	Sorter 4 - pocket 2 full
4	Sorter 4 - photo input pocket 3 covered
5	Sorter 4 - pocket 3 full
6	n.u.
7	n.u.

SensorStatus[byte 4]

Bit	Description
0	Sorter 5 - photo input pocket 1 covered
1	Sorter 5 - pocket 1 full
2	Sorter 5 - photo input pocket 2 covered
3	Sorter 5 - pocket 2 full
4	Sorter 5 - photo input pocket 3 covered
5	Sorter 5 - pocket 3 full
6	n.u.
7	n.u.

SensorStatus[byte 5]

Bit	Description
0	n.u.
1	n.u.
2	n.u.
3	n.u.
4	n.u.
5	n.u.
6	n.u.
7	n.u.

SensorStatus[byte 6]

Bit	Description
0	n.u.
1	n.u.
2	n.u.
3	Photo Double Leafing 1 covered
4	Photo Double Leafing 2 covered
5	Photo Double Leafing 3 covered
6	n.u.
7	n.u.

SensorStatus[byte 7]

Bit	Description
0	if 0 - Documents present in the feeder. if 1 - Feeder empty

1	Photo at begin path base module covered
2	Photo path binary 1 covered
3	Photo path binary 2covered
4	Photo at end path base module covered
5	n.u.
6	n.u.
7	n.u.

SensorStatus[byte 8]
Reserved

SensorStatus[byte 9]

Bit	**Description**
0	Sorter 6 - photo input pocket 1 covered
1	Sorter 6 - pocket 1 full
2	Sorter 6 - photo input pocket 2 covered
3	Sorter 6 - pocket 2 full
4	Sorter 6 - photo input pocket 3 covered
5	Sorter 6 - pocket 3 full
6	n.u.
7	n.u.

SensorStatus[byte 10]

Bit	**Description**
0	Sorter 7 - photo input pocket 1 covered
1	Sorter 7 - pocket 1 full
2	Sorter 7 - photo input pocket 2 covered
3	Sorter 7 - pocket 2 full
4	Sorter 7 - photo input pocket 3 covered
5	Sorter 7 - pocket 3 full
6	n.u.
7	n.u.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_COMMAND_SEQUENCE_ERROR

Comments

More information about the sensor status may be found in the specific LS device manuals.

4.27. LSSetSorterCriteria

#include "LSApi.h"

```
Result API LSSetSorterCriteria (   short                  hConnect,
                                   HWND                   hwnd,
                                   PDATASORTERSELECT      *pCriteria,
                                   SHORT                  NrCriteria);
```

Description

Set the criteria to apply in sorting documents, it is possible to set a maximum of MAX_CRITERIA selection criteria. This command is available on LS510, LS515 models.

Parameters

hConnect

Handle returned by LSConnect

hwnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

pCriteria

Pointer to a DATASORTERSELECT structure (see Comments section below) that specifies the criteria to apply for sorting the documents and deciding the destination storage bin.

NrCriteria

Number of criteria structures in *pCriteria* . This value cannot be greater than MAX_CRITERIA

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_SYSTEM_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR

Comments

In case more than one sorting criteria is provided, make sure that the structures are all contiguous, one after the other.

The DATASORTERSELECT structure is organized as follows :

```
typedef struct _DATASORTERSELECT
{
        char    TypeCriteria;
        char    CharToStart;
        char    NrCharCheck;
        char    String1;
        char    String2;
        char    Bin;
} DATASORTERSELECT, *PDATASORTERSELECT;
```

Description :

TypeCriteria
Specify the selection criteria. Valid values are:

0x00 No Criteria.
0x01 Error in the codeline.
0x02 Codeline equal to string1.
0x03 Codeline different from string1.
0x04 Codeline greater than string1.
0x05 Codeline less than string1.
0x06 Codeline greater than string1 and less than string2.
0x07 Codeline minor string1 or greater string2.
0x08 Codeline equal string1 or equal string2.
0x09 Codeline different from string1 and string2.

CharToStart
Initial character in the codeline used as starting character of the codeline comparison string.

NrCharCheck
Number of characters to include in the comparison string.

String1
First reference string compared to the codeline comparison string.

String2
Second reference string compared to the codeline comparison string.

Bin
Destination Bin for documents that meet *TypeCriteria*

4.28. LSSetOpticalWindows

#include "LSApi.h"

```
Result API LSSetOpticalWindows (    short                    hConnect,
                                    HWND                     hWnd,
                                    PDATAOPTICALWINDOW       pDimWindows,
                                    SHORT                    NrWindows);
```

Description

Set the dimensions and character set to use by the decoding algorithm applied to the section of documents that will be processed.
This function must be invoked before **LSDocHandle**.
It works on the LS100 units which have the DSP option installed.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

pDimWindows

Pointer to a DATAOPTICALWINDOW structure that specifies the dimensions of the portions of document that must be read and the type of the character that must be read.

NrWindos

Number of windows that must be read. This value cannot be grater 1 for LS100 series.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_SYSTEM_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR

Comments

The DATAOPTICALWINDOW structure is as follows :

```
typedef struct _DATAOPTICALWINDOW
{
        unsigned char   TypeRead;
        unsigned char   Reserved;
        short           XRightBottom;
        short           YRightBottom;
        short           XLeftTop;
        short           YLeftTop;
} DATAOPTICALWINDOW, *PDATAOPTICALWINDOW;
```

Description:

TypeRead

Type of reading operation, this member can have the follows values :
READ_CODELINE_HW_OCRA for decoding OCRA characters.
READ_CODELINE_HW_OCRB_NUM for decoding OCRB number characters.
READ_CODELINE_HW_OCRB_ALFANUM for decoding OCRB Alpha-Numeric characters.
READ_CODELINE_HW_OCRB_ITALY for decoding a subset of OCRB Alpha-Numeric characters.
READ_CODELINE_HW_E13B for decoding E13B Optical characters.
READ_CODELINE_HW_E13B_X_OCRB start to decoding E13B Optical and X char switch OCRB characters.

Reserved

Must be set to 0.

XRightBottom

X coordinate in millimetre from the right bottom corner of the document.

YRightBottom

Y coordinate in millimetre from the right bottom corner of the document.

XLeftTop

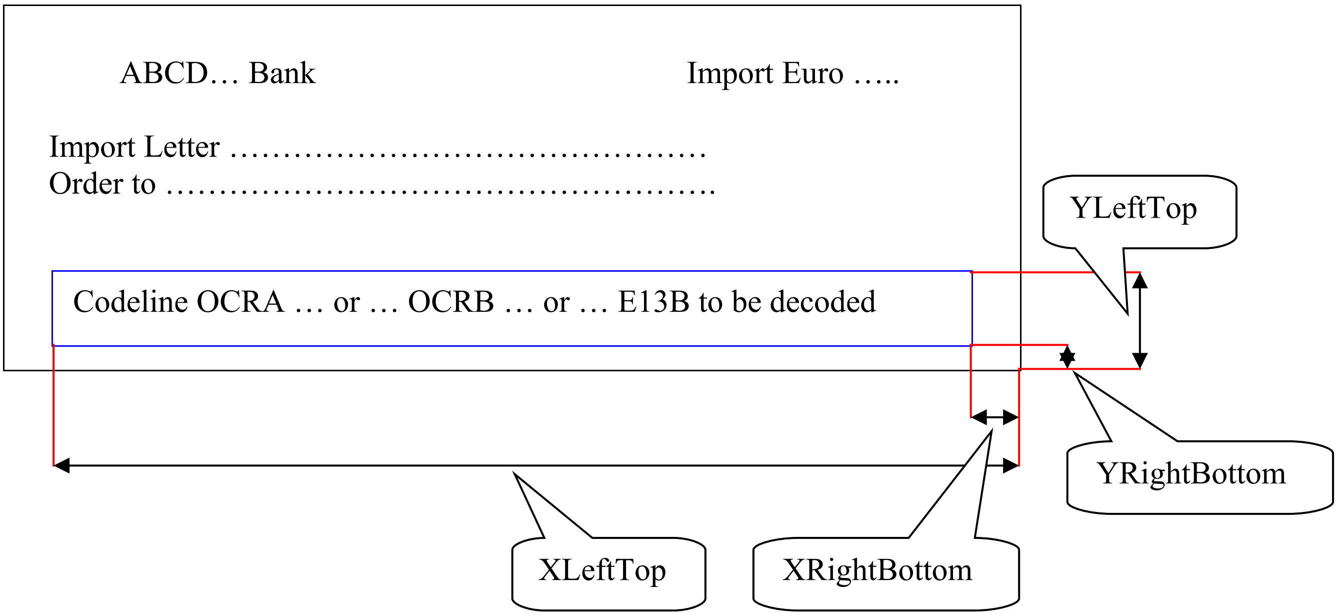
X coordinate in millimetre from the right bottom corner of the document.

YLeftTop

Y coordinate in millimetre from the right bottom corner of the document (this parameter accept only the value YRightBottom + 10).

The coordinate millimetres pair 0,0 refers to the bottom right hand corner of the document.

Example :


ABCD… Bank
Import Euro …..
Import Letter ……………………………………….
Order to ……………………………………………..
Codeline OCRA … or … OCRB … or … E13B to be decoded
YLeftTop
YRightBottom
XLeftTop
XRightBottom

4.29. LSDisableWaitDocument

```
#include "LsApi.h"

Result API LSDisableWaitDocument (    short     hConnect,
                                      HWND      hWnd,
                                      BOOL      Value);
```

Description

The function disable or enable the peripheral timeout on insert documents, by default the timeout is **enabled**.
Function available only with the LS100, LS150 and LS51x model.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Value

TRUE disable the timeout.
FALSE enable the timeout.

Return Value

LS_OKAY
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_OPEN_NOT_DONE

Comments

This performance on LS510 is available only from the firmware version 2.23 or later.

4.30. LSChangeStampPosition

#include "LSApi.h"

```
Result API LSChangeStampPosition(     short     hConnect,
                                      HWND      hwnd,
                                      short     Step,
                                      char      Reserved);
```

Description

Function available only with the LS51x and LS150 models.

Change the position of the stamps placed on the front of the processed document.

Parameters

hConnect

Handle returned by LSConnect

hwnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Step

This parameter specifies the advancement of the motor, in number of steps, before starting to stamp.

LS150 valid range is 0 to 75, each step unit corresponds to 1 mm.
0 – The stamping is done at the default position on the document.
75 – The stamping is done at the left of the document in length.

LS515 valid range is 0 to 25, each step unit corresponds to 2.5 mm.
0 – The stamping is done at the beginning of the document (default value).
25 – The stamping is done at the end of the document of 216 mm in length.

Reserved

This parameter must be set to 0.

Return Value

LS_OKAY
LS_PERIPHERAL_NOT_FOUND
LS_PAPER_JAM
LS_OPEN_NOT_DONE
LS_COMMAND_NOT_SUPPORTED
LS_COMMAND_SEQUENCE_ERROR
LS_INVALID_TYPE_COMMAND
LS_INVALID_FORMAT

Comments

This function is not applicable to LS100, LS5xx series.

4.31. LSUnitHistory

#include "LSApi.h"

```
Result API LSUnitHistory( short          hConnect,
                          HWND           hWnd,
                          UNITHISTORY    *sHistory);
```

Description

Retrieve the historical data stored inside the LS device non volatile memory.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

sHistory

Structure containing the historical data returned by the device.
The field **Size** of the structure **must** be set with the sizeof of the gived structure.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_COMMAND_SEQUENCE_ERROR

Comments

Description of the *UnitHistory* structure :

```
typedef struct _UNITHISTORY
{
      int               Size;                  // Size of the structure

      unsigned long     doc_sorted;            // Document sortered
      unsigned long     doc_retained;          // Nr. of document retained
      unsigned long     doc_retained_micr;     // Nr. documents retained after MICR header
      unsigned long     doc_retained_scan;     // Nr. documents retained after front scanning
      unsigned long     doc_ink_jet;           // Nr. of document printed
      unsigned long     doc_stamped;           // Nr. of document stamped

      unsigned long     tot_paper_jams;        // Totally of Paper jam
      unsigned long     jams_in_feeder;        // Nr. jam in the feeder
      unsigned long     jams_in_micr;          // Nr. jam during the MICR reading
      unsigned long     jams_scanner;          // Nr. jam between scanners
      unsigned long     jams_stamp;            // Nr. jam at stamp document
      unsigned long     jams_on_exit;          // Nr. jam after the film
      unsigned long     jams_card;             // Nr. jam in the card entry
      unsigned long     nr_double_leafing;     // Nr. double leafing occurs Ls800 only

      unsigned long     tot_doc_MICR_err;      // Totally MICR document, read with error
```

```
        unsigned long       doc_cmc7_err;           // Nr. of document CMC7, read with error
        unsigned long       doc_e13b_err;           // Nr. of document E13B, read with error
        unsigned long       doc_hw_barcode_err;     // Nr. of document Barcode, read from LS with error
        unsigned long       doc_hw_optic_err;       // Nr. of document OCR, read from LS with error

        unsigned long       num_turn_on;            // Nr. of power ON
        unsigned long       time_peripheral_on;     // Minutes peripheral time life

        // Section compiled only from Ls800
        unsigned long       jam_front_scanner;      // Jam in scanner front
        unsigned long       jam_track_left;         // Jam in the left track
        unsigned long       jam_track_right;        // Jam in the right track
        unsigned long       jam_back_scanner;       // Jam in scanner back
        unsigned long       jam_in_the_sorters;     // Jam in sorters track
        // Section compiled only from Ls800

        unsigned long       nr_drops_printed;       // Nr. of drops printed

    } UNITHISTORY, *PUNITHISTORY;
```

5. Advanced document handling functions

The functions described in this section allow documents handling in a cycle. For each document in the cycle it is possible to choose the resolution, to read the front or back image and/or the codeline, to print a validation string and to save the image to a file or to memory.

For the LS800 it is advisable that you use these functions, in order to achieve the maximum speed of the unit.
The same applies to all the LS family scanners models, if you use these functions multiple documents will be in the path at the same time, thus obtaining the maximum throughput out of the unit.
The speed of the machine will be completely independend of the application speed.

5.1. LSAutoDocHandle

#include "LSApi.h"

```
Result API LSAutoDocHandle(     short       hConnect,
                                HWND        hWnd,
                                short       Stamp,
                                short       Validate,
                                short       CodeLine,
                                short       ScanMode,
                                short       Feeder,
                                short       Sorter,
                                short       NumDocument,
                                short       ClearBlack,
                                char        Side,
                                short       ScanDocType,
                                short       SaveImage,
                                char        *DirectoryFile,
                                char        *BaseFilename,
                                float       pos_x,
                                float       pos_y,
                                float       sizeW,
                                float       sizeH,
                                short       OriginMeasureDoc,
                                short       OcrImageSide,
                                short       FileFormat,
                                int         Quality,
                                int         SaveMode,
                                int         PageNumber,
                                short       WaitTimeout,
                                short       Beep,
                                int         (*userfunc)(S_CODELINE_INFO *CodelineInfo),
                                LPVOID      Reserved2,
                                LPVOID      Reserved3);
```

Description

Handle documents in a cycle according to the options specified by the various parameters.
This function can be used to automatically handle documents on the LS40, LS100, LS150, LS515 and LS510S models. It must be called before the **LSGetDocData()** loop. To disable the wait timeout after the last document the **LSDisableWaitDocument()** function must be called prior to the LSAutoDocHandle().
Please also refer to the LSConfigDoubleLeafingAndDocLenght() function to configure the paper sensibility in the correct way to handle the double leafing functionality.

NOTE : LS100 model does NOT support the 300 dpi resolution.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Stamp

NO_STAMP = do not stamp document

FRONT_STAMP = stamp front side of document
REAR_STAMP = stamp rear side of document
FRONT_AND_REAR_STAMP = stamp both front and rear sides of document (this option is possible only with LS5xx models)

Validate

NO_PRINT_VALIDATE = do not print validation string
PRINT_VALIDATE = print validation string (the text of the validation string must be loaded using the ***LSLoadString*** command before invoking this command). The LS device must be equipped with the optional validation printer.
PRINT_DIGITAL_VALIDATE = print digital validation string on the image (the text of the validation string must be loaded using the ***LSLoadDigitalStringWithCounter*** command before invoking this command)

Codeline

Specify the type of codeline to read.
NO_READ_CODELINE = do not read codeline.

READ_CODELINE_MICR = read magnetic codeline, either CMC7 or E13B.
READ_CODELINE_E13B_MICR_AND_OCR = read a codeline E13B in magnetic / optic mode.
READ_CODELINE_CMC7_MICR_AND_OCR = read a codeline CMC7 in magnetic / optic mode.
READ_CODELINE_MICRO_HOLES = read a MicroHoles codeline present on the document. **IMPORTANT** if this define is set in order to obtain Micro codelines, then the application must call the function *LSGetDocDataMH()* instead of *LSGetDocData()*.
READ_CODELINE_MICR_AND_MICRO_HOLES = read a codeline CMC7 or E13B and the MicroHoles codeline present on the document. **IMPORTANT** if this define is set in order to obtain Micro codelines, then the application must call the function *LSGetDocDataMH()* instead of *LSGetDocData()*.
READ_CODELINE_CMC7_MOCR_AND_MICRO_HOLES = read a codeline CMC7 in magnetic / optic mode and the MicroHoles codeline present on the document. **IMPORTANT** if this define is set in order to obtain Micro codelines, then the application must call the function *LSGetDocDataMH()* instead of *LSGetDocData()*.
READ_CODELINE_SW_OCRA = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_NUM = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_ALFANUM = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_ITALY = read codeline from the position specified by X, Y and Size parameters. This font is a subset of the alfa-numeric font without the character ‘&’ and same separator.
READ_CODELINE_SW_E13B = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_CMC7 = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_ E13B_X_OCRB = read codeline from the position specified by X, Y and Size parameters. *The co-ordinates start from the bottom right hand corner of the document.*
READ_BARCODE_2_OF_5 = read in optical mode a 2_OF_ 5 barcode.
READ_BARCODE_CODE39 = the software decoding of a CODE39 barcode.

READ_BARCODE_CODE128 = the software decoding of a CODE128 barcode.
READ_2D_BARCODE_PDF417 = read in optical mode a PDF417 barcode.

ScanMode

This parameter set the resolution for the image to read Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 shades of grey, at 100 dpi
SCAN_MODE_16GR120 = 16 grey scale at 120 dpi
SCAN_MODE_16GR200 = 16 shades of grey, at 200 dpi
SCAN_MODE_16GR240 = 16 grey scale at 240 dpi
SCAN_MODE_16GR300 = 16 grey scale at 300 dpi
SCAN_MODE_256GR100 = 256 grey scale at 100 dpi
SCAN_MODE_256GR120 = 256 grey scale at 120 dpi
SCAN_MODE_256GR200 = 256 grey scale at 200 dpi
SCAN_MODE_256GR240 = 256 grey scale at 240 dpi
SCAN_MODE_256GR300 = 256 grey scale at 300 dpi
SCAN_MODE_256GR100_AND_UV = 256 gray scale at 100 dpi and Ultra Violet images
SCAN_MODE_256GR200_AND_UV = 256 gray scale at 200 dpi and Ultra Violet images
SCAN_MODE_256GR300_AND_UV = 256 gray scale at 300 dpi and Ultra Violet images
SCAN_MODE_COLOR_100 = Color 24 bit 100 dpi
SCAN_MODE_COLOR_200 = Color 24 bit 200 dpi
SCAN_MODE_COLOR_300 = Color 24 bit 300 dpi
SCAN_MODE_COLOR_100_AND_UV = 256 gray scale at 100 dpi and Ultra Violet images
SCAN_MODE_COLOR_200_AND_UV = 256 gray scale at 200 dpi and Ultra Violet images
SCAN_MODE_COLOR_300_AND_UV = 256 gray scale at 300 dpi and Ultra Violet images

SCAN_MODE_256GR100BN = 256 gray scale at 100 dpi Brutto and Netto images (LS515 model only)
SCAN_MODE_256GR200BN = 256 gray scale at 200 dpi Brutto and Netto images (LS515 model only)
SCAN_MODE_COLOR_AND_RED_100 = Color 24 bit at 100 dpi and Netto images (LS100 and LS515 model)
SCAN_MODE_COLOR_AND_RED_200 = Color 24 bit at 200 dpi and Netto images (LS100 and LS515 model)
SCAN_MODE_256GR100_ONLY_RED = 256 gray scale at 100 dpi Red images
SCAN_MODE_256GR200_ONLY_RED = 256 gray scale at 200 dpi Red images
SCAN_MODE_256GR300_ONLY_RED = 256 gray scale at 300 dpi Red images

Feeder

It specifies the source of the document
AUTO_FEED = document from feeder.
PATH_FEED = document from path. This value should be used when the document processing requires two or more passes. It requires that in the previous pass the value set for *Sorter* parameter is HOLD_DOCUMENT

Sorter

It specifies the destination of the processed document. The applicable values depend on the model of LS device being used :
HOLD_DOCUMENT = hold document. In the next pass the parameter *Feeder* must take the value PATHFEED
SORTER_BAY1 = document is stacked in sorting pocket 1.
SORTER_BAY2 = document is stacked in sorting pocket 2
SORTER_AUTOMATIC = send the document to the destination set by a previous LSSetSorterCriteria command.
SORTER_SWITCH_1_TO_2 = documents are stacked in Pocket 1, until it becomes full at which point stacking continues to Pocket 2. When Pocket 2 becomes full the service will return a LS_SORTERS_BOTH_FULL error code.
SORTER_ON_CODELINE_CALLBACK_WITH_PRINT = the application decides the destination pocket after reading the MICR codeline in a call-back function and the print of the stringa start always at the same point from the edge of

the documents (LS515-HS models only).

NumDocument

This parameter set the maximum number of documents to read.
0 = read all documents present in the feeder at intervals of 6 seconds one from the other.

ClearBlack

This sets the option for either cleaning or keeping the black area that may be present around the scanned image
NO_CLEAR_BLACK = no image cleaning, the black area is not removed.
CLEAR_ALL_BLACK = clean the document's image removing all the black around the image according to the level of filter specified by **LSSetThresholdClearBlack** command.
CLEAR_AND_ALIGN_IMAGE = removing all the black around the image and align the document's image. (deskew)

Side

It specifies which side(s) of the document to scan
SIDE_FRONT_IMAGE = Scan Front side.
SIDE_BACK_IMAGE = Scan Rear side.
SIDE_ALL_IMAGE = Scan Both sides.
SIDE_NONE_IMAGE = Do not scan document.

ScanDocType

Specifies the type of document to processed.
Accepted values are :
SCAN_PAPER_DOCUMENT : for paper.
SCAN_A4_DOCUMENT : for scan document A4 (only for LS150 G).

SaveImage

Specifies where to store the scanned image(s)
IMAGE_SAVE_ON_FILE = save the image on file
IMAGE_SAVE_HANDLE = save the image in memory
IMAGE_SAVE_BOTH = save the image in memory and on file
IMAGE_SAVE_NONE = do not save on file nor in memory.

DirectoryFile

Path of the Directory where to save the image(s), without file name.

BaseFilename

Root of the file name given to all stored images. For example, if this parameter is set to be equal to **AA** and the chosen File Format is JPEG, the first file created will be named **AA**0FF.jpg, with the FF postfix for front images and the BB postfix for rear images.

pos_X

Specify the x co-ordinates of the codeline. Reference point and unit of measure as of *OriginMeasureDoc* parameter.

pos_Y

Specify the y co-ordinates of the codeline, from the bottom margin of the document. Unit of measure as of *OriginMeasureDoc* parameter.

SizeW

Specify the size of the window that contained codeline in the image. Unit of measure as of *OriginMeasureDoc* parameter.

SizeH

Specify the height of the codeline.
For the moment accept only the values :
OCR_VALUE_IN_MM in case of measures in millimeters.

OCR_VALUE_IN_INCH in case of measures in inches.

OriginMeasureDoc

Specifies the reference point and unit of measure of the document from which measurements start. It can take the following values:
BOTTOM_LEFT_PIXEL = Start from bottom left hand corner. Measures expressed in pixels.
BOTTOM_RIGHT_MM = Start from bottom right hand corner. Measures expressed in millimeters.
BOTTOM_RIGHT_INCH = Start from bottom right hand corner. Measures expressed in inches.

OCR_Image_Side

This parameter specifies the document side to use to read the codeline
OCR_FRONT_IMAGE : use the front side image
OCR_BACK_IMAGE : use the back side image

FileFormat

This parameter specifies the file format used when storing the document's images.
The valid values are :
SAVE_JPEG : save the image in a JPEG format file.
SAVE_BMP : save the image in a DIB format file.
FILE_TIF : save the image in a Tagged Image File Format file.
SAVE_TIF_JPEG = Image TIFF saved in JPEG format.
FILE_CCITT : save the image in a TIFF CCITT file.
FILE_CCITT_GROUP3_1DIM : save the image in a CCITT Group3 one dimension file.
FILE_CCITT_GROUP3_2DIM : save the image in a CCITT Group3 two dimensions file.
FILE_CCITT_GROUP4 : save the image in a CCITT Group4 two dimensions file.

Quality

Integer from 2 to 255 that indicates the relationship between quality and compression:
2 = maximum quality.
255 = maximum compression.

SaveMode

SAVE_OVERWRITE = Overwrite the file. Applicable to all supported values of parameter *FileFormat*
SAVE_APPEND = Append the image in a multipage file images. Not applicable to JPEG and DIB file formats
SAVE_REPLACE = Replace a image in a multipage file images. Not applicable to JPEG and DIB file formats
SAVE_INSERT = Insert the image in a multipage file images. Not applicable to JPEG and DIB file formats

WaitTimeout

This parameter sets the behavior of the device when there are no more documents to process.
WAIT_YES = the device waits for approximately 7 seconds for a new document to process, when this internal timer expires and there is not a new document to process the command completes with return code LS_FEEDER_EMPTY .
WAIT_NO   = if no document is present, or after the last document in the feeder has been processed, the service will immediately complete with return code LS_FEEDER_EMPTY.
This parameter allow to wait the peripheral timeout after the last document in the leafer.

Beep

Specifies whether the internal beeper should emit an acoustical sound when an error occurs.
Accepted values are :
NO_BEEP : do not activate beeper
BEEP : activate beeper

PageNumber

An integer indicating the position inside a multipage file.

userfunc

Call-back function called from the library when the codeline is available. If this function is not provided (NULL) the application will retrieve the codeline string upon the LSGetDocData function call only. If the userfunc parameter is provided, the application will retrieve the codeline string as a parameter on the userfunc call and when calling the LSGetDocData function, as well. The output pocket is decided by the application by providing a valid sorter number during the callback function call. Refer to the structure described hereafter.
This function is available only with the ls515.

Reserved2

Reserved for future use. Must be set to NULL.

Reserved3

Reserved for future use. Must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_RESERVED_ERROR
LS_PAPER_JAM
LS_COMMAND_IN_EXECUTION_YET
LS_JPEG_ERROR

Comments

Note :
If the application is requesting to read MicroHole codeline, the function will force the resolution at 300 dpi. The image returned, however, will be the one requested by the application.

When the SaveImage parameter is set to IMAGE_SAVE_HANDLE or IMAGE_SAVE_BOTH, this function will allocate a memory image buffer for each document. After this function processing completes the application must call the LSGetDocData to retrieve the image handles, display the images and then free the memory buffer.

Description Codeline structure :

```
typedef struct _S_CODELINE_INFO
{
// Parameter compiled by  LsApi.dll
short          Size;                            // Size of the struct
unsigned long  NrDoc;                           // Progessive document number
char           CodelineRead[CODE_LINE_LENGTH]; // Codeline returned
short          NrBytes;                         // Length of the codeline
unsigned long  Reserved;                        // Reserved for future use

// Parameter compiled by Application
short          Sorter;                    // Sorter where put the document
char           FormatString;              // Set from application NORMAL or BOLD
char           StringToPrint[80];         // String to print rear of the document
} S_CODELINE_INFO, *LPS_CODELINE_INFO;
```

5.2. *LSAutoDocHandleVB*

#include "LSApi.h"

```
Result API LSAutoDocHandleVB(   short         hConnect,
                                HWND          hWnd,
                                short         Stamp,
                                short         Validate,
                                short         CodeLine,
                                short         ScanMode,
                                short         Feeder,
                                short         Sorter,
                                short         NumDocument,
                                short         ClearBlack,
                                char          Side,
                                short         ScanDocType,
                                short         SaveImage,
                                char          *DirectoryFile,
                                char          *BaseFilename,
                                float         pos_x,
                                float         pos_y,
                                float         sizeW,
                                float         sizeH,
                                short         OriginMeasureDoc,
                                short         OcrImageSide,
                                short         FileFormat,
                                int           Quality,
                                int           SaveMode,
                                int           PageNumber,
                                short         WaitTimeout,
                                short         Beep,
                                int           (*userfunc)(S_CODELINE_INFO_VB
                                *CodelineInfo),
                                LPVOID        Reserved2,
                                LPVOID        Reserved3);
```

Description

Handle documents in a cycle according to the options specified by the various parameters.
This function was expecially developed for Visual Basic programming where callback functions must be defined as Standard Call.
Refer to LsAutoDocHandle for parameters and description.

Parameters

See the *LSAutoDocHandle()* function.

Return Value

See the *LSAutoDocHandle()* function.

Comments

When the SaveImage parameter is set to IMAGE_SAVE_HANDLE or IMAGE_SAVE_BOTH, this function will allocate a memory image buffer for each document. After this function processing completes the application must call the LSGetDocData to retrieve the image handles, display the images and then free the memory buffer.

Description Codeline structure :

```
typedef struct _S_CODELINE_INFO_VB
{
// Parameter compiled from LsApi.dll
short           Size;                              // Size of the struct
unsigned long   NrDoc;                             // Progessive document number
WCHAR           CodelineRead[CODE_LINE_LENGTH]; // Codeline returned
short           NrBytes;                           // Length of the codeline
unsigned long   Reserved;                          // Reserved for future use

// Parameter compiled from Application
short           Sorter;                    // Sorter where put the document
short           FormatString;              // Set from application NORMAL or BOLD
WCHAR           StringToPrint[80];         // String to print rear of the document
} S_CODELINE_INFO_VB, *LPS_CODELINE_INFO_VB;
```

5.3. LS800AutoDocHandle

#include "LsApi.h"

```
Result API LS800AutoDocHandle(  short     hConnect,
                                HWND      hWnd,
                                char      Validate,
                                short     CodeLine,
                                char      Side,
                                short     ScanModeFront,
                                short     ScanModeBack,
                                short     ClearBlack,
                                short     NumDocument,
                                short     SaveImage,
                                char      *DirectoryFile,
                                char      *BaseFilename,
                                short     UnitMeasure,
                                float     pos_x,
                                float     pos_y,
                                float     sizeW,
                                float     sizeH,
                                short     OCR_Image_Side,
                                short     FileFormat,
                                int       Quality,
                                int       SaveMode,
                                int       PageNumber,
                                short     Beep,
                                int       (*userfunc1)(S_CODELINE_INFO_LS800 *CodelineInfo),
                                int       (*userfunc2)(S_IMAGE_INFO_LS800 *ImageInfo),
                                int       (*userfunc3)(S_IMAGE_INFO_LS800 *ImageInfo),
                                LPVOID    Reserved1,
                                LPVOID    Reserved2,
                                LPVOID    Reserved3);
```

Description

This function handles documents in a cycle according to the options specified by the various parameters.

The sorting and the ink-jet of the documents are decided by the application in the post-routine. This can be done or after having read the codeline in the associated call-back function, or after having read the front image in the associated call-back function.

The Post-Routine must be a very short piece of code that must decide the pocket according to the MICR codeline returned by the peripheral or some information retrieved from the image. This information can be for example a OCR line, obtained with the function *LSCodelineReadFromBitmap()*.

Note that all the information of each single document are returned all together in a second time on the *LSGetDocData()* function call.

Only one call-back function address can be provided at the same time.

Please refer to the LSConfigDoubleLeafingEx() function to configure the paper sensibility in the correct way to handle the double leafing functionality.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved**

for future use).

Validate

NO_PRINT_VALIDATE = do not print validation string.
PRINT_VALIDATE = print validation string (the text of the validation string must be loaded using the ***LSLoadString*** command before invoking this command). The LS device must be equipped with the optional validation printer.

Codeline

Specify the type of codeline to read.
NO_READ_CODELINE = do not read codeline.
READ_CODELINE_MICR = read magnetic codeline, either CMC7 or E13B.

Software read :
READ_BARCODE_PDF417 = read in optical mode a PDF417 barcode.
READ_BARCODE_2_OF_5 = read in optical mode a 2_OF_ 5 barcode.
READ_BARCODE_CODE39 = read in optical mode a CODE 39 barcode.
READ_BARCODE_CODE128 = read in optical mode a CODE 128 barcode.
READ_CODELINE_SW_OCRA = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document*.
READ_CODELINE_SW_OCRB_NUM = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_ALFANUM = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_ITALY = read codeline from the position specified by X, Y and Size parameters. This font is a subset of the alfa-numeric font without the character '&' and same separator. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_E13B = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_CMC7 = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_ E13B_X_OCRB = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*

Side

It specifies which side(s) of the document to scan
SIDE_FRONT_IMAGE = Scan Front side
SIDE_BACK_IMAGE = Scan Rear side
SIDE_ALL_IMAGE = Scan Both sides
SIDE_NONE_IMAGE = Do not scan document

ScanModeFront

This parameter set the resolution for the image to read. Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 shades of grey, at 100 dpi
SCAN_MODE_16GR200 = 16 shades of grey, at 200 dpi
SCAN_MODE_256GR100 = 256 shades of grey, at 100 dpi
SCAN_MODE_256GR200 = 256 shades of grey, at 200 dpi
SCAN_MODE_COLOR_100 = colour at 100 dpi
SCAN_MODE_COLOR_200 = colour at 200 dpi

ScanModeBack

This parameter set the resolution for the image to read Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 shades of grey, at 100 dpi

SCAN_MODE_16GR200 = 16 shades of grey, at 200 dpi
SCAN_MODE_256GR100 = 256 shades of grey, at 100 dpi
SCAN_MODE_256GR200 = 256 shades of grey, at 200 dpi
SCAN_MODE_COLOR_100 = Color 24 bit 100 dpi
SCAN_MODE_COLOR_200 = Color 24 bit 200 dpi

ClearBlack

This sets the option for either cleaning or keeping the black area that may be present around the scanned image
NO_CLEAR_BLACK = no image cleaning, the black area is not removed.
CLEAR_ALL_BLACK = clean the document's image removing all the black around the image according to the level of filter specified by **LSSetThresholdClearBlack** command.
CLEAR_AND_ALIGN_IMAGE = removing all the black around the image and align the document's image.

NumDocument

This parameter set the maximum number of documents to read.
0 = read all documents present in the feeder at intervals of 6 seconds one from the other.

SaveImage

Specifies where to store the scanned image(s)
IMAGE_SAVE_HANDLE = save the image in memory.
IMAGE_SAVE_ON_FILE = save the image on file.
IMAGE_SAVE_BOTH = save the image in memory and on file.
IMAGE_SAVE_NONE = do not save on memory nor on file.

DirectoryFile

Path of the Directory where to save the image(s), without file name.

BaseFilename

Root of the file name given to all stored images. For example, if this parameter is set to be equal to **AA** and the chosen File Format is JPEG, the first file created will be named **AA**0FF.jpg, with the FF postfix for front images and the BB postfix for rear images.

UnitMeasure

Specify whether the *Start_X, Start_Y* and *SizeW* measures are expressed in millimeters or in inches.
The possible values are either **UNIT_MM** or **UNIT_INCH**.

Start_X

Specify the x co-ordinate from right hand margin of the document. The value must be consistent with the *UnitMeasure*.

Start_Y

Specify the y co-ordinate from bottom margin of the document. The value must be consistent with the *UnitMeasure*.

SizeW

Specify the width of the window on the image bitmap that will be processed by the decoding software. The value must be consistent with the *UnitMeasure*.

SizeH

Specify the height of the window on the image bitmap that will be processed by the decoding software.
Valid values are :
OCR_VALUE_IN_MM for measures expressed in millimeters.
OCR_VALUE_IN_INCH for measures expressed in inches.

OCR_Image_Side

This parameter specifies the document side to use to read the codeline
OCR_FRONT_IMAGE : use the front side image
OCR_BACK_IMAGE : use the back side image

FileFormat

This parameter specifies the file format used when storing the document's images.
The valid values are :
SAVE_JPEG : save the image in a JPEG format file.
SAVE_BMP : save the image in a DIB format file.
FILE_TIF : save the image in a Tagged Image File Format file.
SAVE_TIF_JPEG = Image TIFF saved in JPEG format.
FILE_CCITT : save the image in a TIFF CCITT file.
FILE_CCITT_GROUP3_1DIM : save the image in a CCITT Group3 one dimension file.
FILE_CCITT_GROUP3_2DIM : save the image in a CCITT Group3 two dimensions file.
FILE_CCITT_GROUP4 : save the image in a CCITT Group4 two dimensions file.

Quality

Integer from 2 to 255 that indicates the relationship between quality and compression:
2 = maximum quality.
255 = maximum compression.

SaveMode

SAVE_OVERWRITE = Overwrite the file. Applicable to all supported values of parameter *FileFormat.*
SAVE_APPEND = Append the image in a multi-page file images. Not applicable to JPEG and DIB file formats.
SAVE_REPLACE = Replace a image in a multi-page file images. Not applicable to JPEG and DIB file formats.
SAVE_INSERT = Insert the image in a multi-page file images. Not applicable to JPEG and DIB file formats.

PageNumber

An integer indicating the position inside a multi-page file.

Beep

Specifies whether the internal beeper should emit an acoustical sound when an error occurs.
Accepted values are :
BEEP_NO : do not activate beeper.
BEEP_YES : activate beeper.

userfunc1

Call-back function called from the library then the codeline is available.

userfunc2

Call-back function called from the library then the front image is available.

userfunc3

Call-back function called from the library then the back image is available (not available at this moment).

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2

Reserved for future use. Must be set to NULL.

Reserved3

Reserved for future use. Must be set to NULL.

Return Value

LS800_OKAY
LS800_SYSTEM_ERROR
LS800_USB_ERROR
LS800_PERIPHERAL_NOT_FOUND
LS800_HARDWARE_ERROR
LS800_PAPER_JAM
LS800_INVALID_COMMAND
LS800_COMMAND_IN_EXECUTION_YET

Comments

When the SaveImage parameter is set to IMAGE_SAVE_HANDLE or IMAGE_SAVE_BOTH, this function will allocate a memory image buffer for each document. After this function processing completes the application must call the LS800_GetDocData to retrieve the image handles, process the images and then free the memory buffer.
The images buffers must be released by the application with the function *LS800_FreeImage()*.

Description of the structures returned as parameter in the call-back functions.

Description Codeline structure :

```
typedef struct _S_CODELINE_INFO_LS800
{
    // Parameter filled by LsApi
    short           Size;                                  // Size of the struct
    unsigned long   NrDoc;                                 // Progessive document number
    char            CodelineRead[CODE_LINE_LENGTH];        // Codeline returned
    short           NrBytes;                               // Length of the codeline
    unsigned long   Reserved;                              // Reserved for future use

    // Parameter compiled from Application
    short           Sorter;                 // Sorter where put the document
    char            FormatString1;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint1[80];     // String line 1 to print rear of the document
    char            FormatString2;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint2[80];     // String line 2 to print rear of the document
    char            FormatString3;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint3[80];     // String line 3 to print rear of the document
    char            FormatString4;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint4[80];     // String line 4 to print rear of the document

} S_CODELINE_INFO_LS800, *LPS_CODELINE_INFO_LS800;
```

Description Image structure :

```
typedef struct _S_IMAGE_INFO_LS800
{
	// Parameter filled by LsApi
	short		Size;					// Size of the struct
	unsigned long	NrDoc;					// Progressive document number
	HANDLE		hImage;					// Image handle
	int		ImageSize;				// Image size bytes
	int		Width;					// Image width
	int		Height;					// Image height
	int		Resolution;				// Image resolution
	int		BitCount;				// Image bit count (level of grey)
	char		CodelineRead[CODE_LINE_LENGTH];		// Codeline OCR or MICR returned
	short		NrBytes;				// Length of the codeline
	unsigned long	Reserved;				// Reserved for future use

	// Parameter compiled from Application
	short		Sorter;			// Sorter where put the document
	char		FormatString1;		// Set from application NORMAL, NORMAL 15 or
						// BOLD
	char		StringToPrint1[80];	// String line 1 to print rear of the document
	char		FormatString2;		// Set from application NORMAL, NORMAL 15 or
						// BOLD
	char		StringToPrint2[80];	// String line 2 to print rear of the document
	char		FormatString3;		// Set from application NORMAL, NORMAL 15 or
						// BOLD
	char		StringToPrint3[80];	// String line 3 to print rear of the document
	char		FormatString4;		// Set from application NORMAL, NORMAL 15 or
						// BOLD
	char		StringToPrint4[80];	// String line 4 to print rear of the document

} S_IMAGE_INFO_LS800, *LPS_IMAGE_INFO_LS800;
```

5.4. LS800AutoDocHandleVB

#include "LSApi.h"

```
Result API LS800AutoDocHandleVB( short       hConnect,
                                 HWND        hWnd,
                                 char        Validate,
                                 short       CodeLine,
                                 char        Side,
                                 short       ScanModeFront,
                                 short       ScanModeBack,
                                 short       ClearBlack,
                                 short       NumDocument,
                                 short       SaveImage,
                                 char        *DirectoryFile,
                                 char        *BaseFilename,
                                 short       UnitMeasure,
                                 float       pos_x,
                                 float       pos_y,
                                 float       sizeW,
                                 float       sizeH,
                                 short       OCR_Image_Side,
                                 short       FileFormat,
                                 int         Quality,
                                 int         SaveMode,
                                 int         PageNumber,
                                 short       Beep,
                                 int    (*userfunc1)(S_CODELINE_INFO_VB_LS800
                                 *CodelineInfo),
                                 int    (*userfunc2)(S_IMAGE_INFO_VB_LS800 *ImageInfo),
                                 int    (*userfunc3)(S_IMAGE_INFO_VB_LS800 *ImageInfo),
                                 LPVOID      Reserved1,
                                 LPVOID      Reserved2,
                                 LPVOID      Reserved3);
```

Description

This function is similar to the *LS800_AutoDocHandle()* function but is expecially developed for Visual Basic programming Language.
The main differences are the structures given as parameter on the callback functions.
In these structures some fields are defined as type UNICODE instead of ASCII.

Parameters

See the *LS800_AutoDocHandle()* function.

Return Value

See the *LS800_AutoDocHandle()* function.

Comments

Description of the structures provided as parameter in the call-back functions.

Description Codeline structure :

```
typedef struct _S_CODELINE_INFO_VB_LS800
{
    // Parameter filled by LsApi
    short           Size;                                // Size of the struct
    unsigned long   NrDoc;                               // Progressive document number
    WCHAR           CodelineRead[CODE_LINE_LENGTH];      // Codeline returned
    short           NrBytes;                             // Length of the codeline
    unsigned long   Reserved;                            // Reserved for future use

    // Parameter compiled from Application
    short           Sorter;                  // Sorter where put the document
    short           FormatString1;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint1[80];      // String line 1 to print rear of the document
    short           FormatString2;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint2[80];      // String line 2 to print rear of the document
    short           FormatString3;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint3[80];      // String line 3 to print rear of the document
    short           FormatString4;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint4[80];      // String line 4 to print rear of the document

} S_CODELINE_INFO_VB_LS800, *LPS_CODELINE_INFO_VB_LS800;
```

Description Image structure :

```
typedef struct _S_IMAGE_INFO_VB_LS800
{
    // Parameter filled by LsApi
    short           Size;                                // Size of the struct
    unsigned long   NrDoc;                               // Progressive document number
    HANDLE          hImage;                              // Image handle
    int             ImageSize;                           // Image size bytes
    int             Width;                               // Image width
    int             Height;                              // Image height
    int             Resolution;                          // Image resolution
    int             BitCount;                            // Image bit count (level of grey)
    WCHAR           CodelineRead[CODE_LINE_LENGTH];      // Codeline OCR or MICR returned
    short           NrBytes;                             // Length of the codeline
    unsigned long   Reserved;                            // Reserved for future use

    // Parameter compiled from Application
    short           Sorter;                  // Sorter where put the document
    short           FormatString1;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint1[80];      // String line 1 to print rear of the document
    short           FormatString2;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint2[80];      // String line 2 to print rear of the document
    short           FormatString3;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint3[80];      // String line 3 to print rear of the document
    short           FormatString4;           // Set from application NORMAL, NORMAL 15 or
                                             // BOLD
    WCHAR           StringToPrint4[80];      // String line 4 to print rear of the document
} S_IMAGE_INFO_VB_LS800, *LPS_IMAGE_INFO_VB_LS800;
```

Visual basic structure definition example :

```
Public Type TCODELINE_INFO
   '// Parameter filled by LsApi
   Size As Integer                      '// Size of the struct
   NrDoc As Long                        '// Progessive document number
   CodelineRead As String * CODE_LINE_LENGTH      '// Codeline returned
   NrBytes As Integer                   '// Length of the codeline
   Reserved As Long                     '// Reserved for future use

   '// Parameter filled by Application
   Sorter As Integer                    '// Sorter where put the document
   FormatString1 As Byte                '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint1 As String * 80        '// String line 1 to print rear of the document
   FormatString2 As Byte                '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint2 As String * 80        '// String line 2 to print rear of the document
   FormatString3 As Byte                '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint3 As String * 80        '// String line 3 to print rear of the document
   FormatString4 As Byte                '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint4 As String * 80        '// String line 4 to print rear of the document
End Type

Public Type TIMAGE_INFO
   '// Parameter filled by LsApi
   Size As Integer                      '// Size of the struct
   NrDoc As Long                        '// Progessive document number
   hImage As Long                       '// Image handle
   ImageSize As Long                    '// Image size bytes
   Width  As Long                       '// Image width
   Height As Long                       '// Image height
   Resolution As Long                   '// Image resolution
   BitCount As Long                     '// Image bit count (level of grey)
   CodelineRead As String * CODE_LINE_LENGTH '// Codeline OCR or MICR returned
   NrBytes As Integer                   '// Length of the codeline
   Reserved As Long                     '// Reserved for future use

   '// Parameter filled by  Application
   Sorter As Integer                    '// Sorter where put the document
   FormatString1 As Integer             '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint1 As String * 80        '// String line 1 to print rear of the document
   FormatString2 As Integer             '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint2 As String * 80        '// String line 2 to print rear of the document
   FormatString3 As Integer             '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint3 As String * 80        '// String line 3 to print rear of the document
   FormatString4 As Integer             '// Set from application NORMAL, NORMAL 15 or BOLD
   StringToPrint4 As String * 80        '// String line 4 to print rear of the document
End Type
```

5.5. *LS800AutoDocHandleWithAllCallback*

#include "LsApi.h"

```
Result API LS800AutoDocHandleWithAllCallback (  short    hConnect,
                                               HWND     hWnd,
                                               char     Validate,
                                               short    CodeLine,
                                               char     Side,
                                               short    ScanModeFront,
                                               short    ScanModeBack,
                                               short    ClearBlack,
                                               short    NumDocument,
                                               short    SaveImage,
                                               char     *DirectoryFile,
                                               char     *BaseFilename,
                                               short    UnitMeasure,
                                               float    pos_x,
                                               float    pos_y,
                                               float    sizeW,
                                               float    sizeH,
                                               short    OCR_Image_Side,
                                               short    FileFormat,
                                               int      Quality,
                                               int      SaveMode,
                                               int      PageNumber,
                                               short    Beep,
                                               short    SortOnChoice,
                                               int    (*userfunc1)(S_CODELINE_INFO_LS800 *CodelineInfo),
                                               int    (*userfunc2)(S_IMAGE_INFO_LS800 *ImageInfo),
                                               int    (*userfunc3)(S_IMAGE_INFO_LS800 *ImageInfo),
                                               LPVOID   Reserved1,
                                               LPVOID   Reserved2,
                                               LPVOID   Reserved3);
```

Description

This function handles documents in a cycle according to the options specified by the various parameters.
The sorting and the ink-jet of the documents are decided by the application in the post-routine. This can be done or after having read the codeline in the associated call-back function, or after having read the front image in the associated call-back function.
The Post-Routine must be a very short piece of code that must decide the pocket according to the MICR codeline returned by the peripheral or some information retrieved from the image. This information can be for example a OCR line, obtained with the function *LSCodelineReadFromBitmap()*.
Note that all the information of each single document are returned all together in a second time on the *LSGetDocData()* function call.
Only one call-back function address can be provided at the same time.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Validate

NO_PRINT_VALIDATE = do not print validation string
PRINT_VALIDATE = print validation string (the text of the validation string must be loaded using the ***LSLoadString*** command before invoking this command). The LS device must be equipped with the optional validation printer.

Codeline

Specify the type of codeline to read.
NO_READ_CODELINE = do not read codeline.
READ_CODELINE_MICR = read magnetic codeline, either CMC7 or E13B.

Software read :
READ_BARCODE_PDF417 = read in optical mode a PDF417 barcode.
READ_BARCODE_2_OF_5 = read in optical mode a 2_OF_ 5 barcode.
READ_BARCODE_CODE39 = read in optical mode a CODE 39 barcode.
READ_BARCODE_CODE128 = read in optical mode a CODE 128 barcode.
READ_CODELINE_SW_OCRA = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document*.
READ_CODELINE_SW_OCRB_NUM = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_ALFANUM = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_OCRB_ITALY = read codeline from the position specified by X, Y and Size parameters. This font is a subset of the alfa-numeric font without the character '&' and same separator. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_E13B = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*
READ_CODELINE_SW_ E13B_X_OCRB = read codeline from the position specified by X, Y and Size parameters. *The co-ordinate start from bottom right corner of the document.*

Side

It specifies which side(s) of the document to scan
SIDE_FRONT_IMAGE = Scan Front side
SIDE_BACK_IMAGE = Scan Rear side
SIDE_ALL_IMAGE = Scan Both sides
SIDE_NONE_IMAGE = Do not scan document

ScanModeFront

This parameter set the resolution for the image to read. Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 shades of grey, at 100 dpi
SCAN_MODE_16GR200 = 16 shades of grey, at 200 dpi
SCAN_MODE_256GR100 = 256 shades of grey, at 100 dpi
SCAN_MODE_256GR200 = 256 shades of grey, at 200 dpi
SCAN_MODE_COLOR_100 = colour at 100 dpi
SCAN_MODE_COLOR_200 = colour at 200 dpi

ScanModeBack

This parameter set the resolution for the image to read Accepted values are :
SCAN_MODE_BW = black and white at 200 dpi
SCAN_MODE_16GR100 = 16 shades of grey, at 100 dpi
SCAN_MODE_16GR200 = 16 shades of grey, at 200 dpi
SCAN_MODE_256GR100 = 256 shades of grey, at 100 dpi
SCAN_MODE_256GR200 = 256 shades of grey, at 200 dpi
SCAN_MODE_COLOR_100 = Color 24 bit 100 dpi

SCAN_MODE_COLOR_200 = Color 24 bit 200 dpi

ClearBlack

This sets the option for either cleaning or keeping the black area that may be present around the scanned image
NO_CLEAR_BLACK = no image cleaning, the black area is not removed.
CLEAR_ALL_BLACK = clean the document's image removing all the black around the image according to the level of filter specified by **LSSetThresholdClearBlack** command.
CLEAR_AND_ALIGN_IMAGE = removing all the black around the image and align the document’s image.

NumDocument

This parameter set the maximum number of documents to read.
0 = read all documents present in the feeder at intervals of 6 seconds one from the other.

SaveImage

Specifies where to store the scanned image(s)
IMAGE_SAVE_HANDLE = save the image in memory.
IMAGE_SAVE_ON_FILE = save the image on file.
IMAGE_SAVE_BOTH = save the image in memory and on file.
IMAGE_SAVE_NONE = do not save on memory nor on file.

DirectoryFile

Path of the Directory where to save the image(s), without file name.

BaseFilename

Root of the file name given to all stored images. For example, if this parameter is set to be equal to **AA** and the chosen File Format is JPEG, the first file created will be named **AA**0FF.jpg, with the FF postfix for front images and the BB postfix for rear images.

UnitMeasure

Specify whether the *Start_X, Start_Y* and *SizeW* measures are expressed in millimeters or in inches.
The possible values are either **UNIT_MM** or **UNIT_INCH**.

Start_X

Specify the x co-ordinate from right hand margin of the document. The value must be consistent with the *UnitMeasure*.

Start_Y

Specify the y co-ordinate from bottom margin of the document. The value must be consistent with the *UnitMeasure*.

SizeW

Specify the width of the window on the image bitmap that will be processed by the decoding software. The value must be consistent with the *UnitMeasure*.

SizeH

Specify the height of the window on the image bitmap that will be processed by the decoding software.
Valid values are :
OCR_VALUE_IN_MM for measures expressed in millimeters.
OCR_VALUE_IN_INCH for measures expressed in inches.

OCR_Image_Side

This parameter specifies the document side to use to read the codeline
OCR_FRONT_IMAGE : use the front side image
OCR_BACK_IMAGE : use the back side image

FileFormat

This parameter specifies the file format used when storing the document's images.
The valid values are :
SAVE_JPEG : save the image in a JPEG format file.
SAVE_BMP : save the image in a DIB format file.
FILE_TIF : save the image in a Tagged Image File Format file.
FILE_CCITT : save the image in a TIFF CCITT file.
FILE_CCITT_GROUP3_1DIM : save the image in a CCITT Group3 one dimension file.
FILE_CCITT_GROUP3_2DIM : save the image in a CCITT Group3 two dimensions file.
FILE_CCITT_GROUP4 : save the image in a CCITT Group4 two dimensions file.

Quality

Integer from 2 to 255 that indicates the relationship between quality and compression:
2 = maximum quality.
255 = maximum compression.

SaveMode

SAVE_OVERWRITE = Overwrite the file. Applicable to all supported values of parameter *FileFormat.*
SAVE_APPEND = Append the image in a multi-page file images. Not applicable to JPEG and DIB file formats.
SAVE_REPLACE = Replace a image in a multi-page file images. Not applicable to JPEG and DIB file formats.
SAVE_INSERT = Insert the image in a multi-page file images. Not applicable to JPEG and DIB file formats.

PageNumber

An integer indicating the position inside a multi-page file.

Beep

Specifies whether the internal beeper should emit an acoustical sound when an error occurs.
Accepted values are :
BEEP_NO : do not activate beeper.
BEEP_YES : activate beeper.

SortOnChoice

Specifies which is call-back function valid to select the pocket.
Accepted values are :
SORT_ON_MICR : Sort selection on MICR call-back function.
SORT_ON_FRONT_IMAGE : Sort selection on Front image call-back function.

userfunc1

Call-back function called from the library then the codeline is available.

userfunc2

Call-back function called from the library then the front image is available.

userfunc3

Call-back function called from the library then the back image is available.

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2

Reserved for future use. Must be set to NULL.

Reserved3

Reserved for future use. Must be set to NULL.

Return Value

LS800_OKAY
LS800_SYSTEM_ERROR
LS800_USB_ERROR
LS800_PERIPHERAL_NOT_FOUND
LS800_HARDWARE_ERROR
LS800_PAPER_JAM
LS800_INVALID_COMMAND
LS800_COMMAND_IN_EXECUTION_YET

Comments

When the SaveImage parameter is set to IMAGE_SAVE_HANDLE or IMAGE_SAVE_BOTH, this function will allocate a memory image buffer for each document. After this function processing completes the application must call the LS800_GetDocData to retrieve the image handles, process the images and then free the memory buffer.
The images buffers must be released by the application with the function *LS800_FreeImage()*.

Description of the structures returned as parameter in the call-back functions.

Description Codeline structure :

```
typedef struct _S_CODELINE_INFO_LS800
{
    // Parameter filled by LsApi
    short           Size;                                   // Size of the struct
    unsigned long   NrDoc;                                  // Progessive document number
    char            CodelineRead[CODE_LINE_LENGTH];         // Codeline returned
    short           NrBytes;                                // Length of the codeline
    unsigned long   Reserved;                               // Reserved for future use

    // Parameter compiled from Application
    short           Sorter;                 // Sorter where put the document
    char            FormatString1;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint1[80];     // String line 1 to print rear of the document
    char            FormatString2;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint2[80];     // String line 2 to print rear of the document
    char            FormatString3;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint3[80];     // String line 3 to print rear of the document
    char            FormatString4;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint4[80];     // String line 4 to print rear of the document

} S_CODELINE_INFO_LS800, *LPS_CODELINE_INFO_LS800;
```

Description Image structure :

```
typedef struct _S_IMAGE_INFO_LS800
{
    // Parameter filled by LsApi
    short           Size;                                   // Size of the struct
    unsigned long   NrDoc;                                  // Progressive document number
    HANDLE          hImage;                                 // Image handle
    int             ImageSize;                              // Image size bytes
    int             Width;                                  // Image width
    int             Height;                                 // Image height
    int             Resolution;                         // Image resolution
    int             BitCount;                           // Image bit count (level of grey)
    char            CodelineRead[CODE_LINE_LENGTH];     // Codeline OCR or MICR returned
    short           NrBytes;                            // Length of the codeline
    unsigned long   Reserved;                           // Reserved for future use

    // Parameter compiled from Application
    short           Sorter;                 // Sorter where put the document
    char            FormatString1;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint1[80];     // String line 1 to print rear of the document
    char            FormatString2;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint2[80];     // String line 2 to print rear of the document
    char            FormatString3;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint3[80];     // String line 3 to print rear of the document
    char            FormatString4;          // Set from application NORMAL, NORMAL 15 or
                                            // BOLD
    char            StringToPrint4[80];     // String line 4 to print rear of the document

} S_IMAGE_INFO_LS800, *LPS_IMAGE_INFO_LS800;
```

5.6. LSGetDocData

#include "LSApi.h"

```
Result API LSGetDocData (   short            hConnect,
                            HWND             hWnd,
                            unsigned long    *NrDoc,
                            LPSTR            FilenameFront,
                            LPSTR            FilenameBack,
                            LPSTR            Reserved1,
                            LPSTR            Reserved2,
                            LPHANDLE         *FrontImage,
                            LPHANDLE         *RearImage,
                            LPHANDLE         *FrontImage2,
                            LPHANDLE         *RearImage2,
                            LPSTR            CodelineSW,
                            LPSTR            CodelineHW,
                            LPSTR            Barcode,
                            LPVOID           CodelinesOptical,
                            SHORT            *DocToRead,
                            LONG             *NrPrinted,
                            LPVOID           Reserved5,
                            LPVOID           Reserved6);
```

Description

This function should be used to retrieve the information about the documents processed by LSAutoDocHandle. If no document has been processed the function returns a LS_FEEDER_EMPTY completion code.
If the LsAutoDocHandle is called with NumDocument parameter set to 0, the application must call the LSGetDocData function in a loop until LS_FEEDER_EMPTY completion code or an error code is returned.
When the function return a warning of destination sorter full, the application must continue call the function until return LS_NO_OTHER_DOCUMENT.
Note that if a double leafing occurs, this error code is returned by the LSGetDocData function itself, it is very important that the application handles this situation in the correct way.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

NrDoc

Progressive sequence number identifying the document whose data is retrieved by this function.

FilenameFront

Full pathname of the file where the front image is stored. (max 128)

FilenameBack

Full pathname of the file where the back image is stored. (max 128)

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2
Reserved for future use. Must be set to NULL.

**FrontImage*
Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in DIB format.

**RearImage*
Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in DIB format.

**FrontImage2*
Pointer to a handle where will be returned the handle of memory buffer containing the **Ultra Violet** front side image of the requested document, in DIB format.

**RearImage2*
Reserved for future use. Must be set to NULL.

CodelineSW
This field will return the codeline data read using the software interpretation algorithm

CodelineHW
This field will return the codeline data read using the MICR reader . This buffer will be filled also in case Magnetic and Optical is requested.

Barcode
This field will return the barcode codeline data, either PDF417 or 2_OF_5, read using the barcode reader.

CodelinesOptical
This field will return the OCR codeline data read using the OCR reader.

**DocToRead*
Not used.

**NrPrinted*
Variable where it will be returned the progressive count number physically endorsed on the documents. A previous *LSLoadStringWithCounterEx()* call has to be done.

**Reserved5*
Reserved for future use. Must be set to NULL.

**Reserved6*
Reserved for future use. Must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_INVALID_COMMAND
LS_FEEDER_EMPTY
LS_DOUBLE_LEAFING_WARNING
LS_DOUBLE_LEAFING_ERROR
LS_LOOP_INTERRUPTED
LS_REPLACE_CARTRIDGE

Comments

The image handle retrieved by this function must be released by the application.
In case more than one Optic Codeline is returned they are separated by a space character.

5.7. LSGetDocDataMH

#include "LSApi.h"

```
Result API LSGetDocDataMH ( short              hConnect,
                            HWND               hWnd,
                            unsigned long      *NrDoc,
                            LPSTR              FilenameFront,
                            LPSTR              FilenameBack,
                            LPSTR              Reserved1,
                            LPSTR              Reserved2,
                            LPHANDLE           *FrontImage,
                            LPHANDLE           *RearImage,
                            LPHANDLE           *FrontImage2,
                            LPHANDLE           *RearImage2,
                            LPSTR              CodelineSW,
                            LPSTR              CodelineHW,
                            LPSTR              Barcode,
                            LPVOID             CodelinesOptical,
                            SHORT              *DocToRead,
                            LONG               *NrPrinted,
                            BOOL               VerifyHole,
                            short              UnitMeasure,
                            short              nrRegions,
                            MICROHOLE_STRUCT   stMicroHole);
```

Description

This function is same to *LSGetDocData()* function, but return also the Micro codelines if the parameter Codeline of the function LSAutoDocHandle() is set to **READ_CODELINE_MICR_AND_MICRO_HOLES** or **READ_CODELINE_CMC7_MOCR_AND_MICRO_HOLES**.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

NrDoc

Progressive sequence number identifying the document whose data is retrieved by this function.

FilenameFront

Full pathname of the file where the front image is stored. (max 128)

FilenameBack

Full pathname of the file where the back image is stored. (max 128)

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2

Reserved for future use. Must be set to NULL.

**FrontImage*

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in DIB format.

**RearImage*

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in DIB format.

**FrontImage2*

Pointer to a handle where will be returned the handle of memory buffer containing the **Ultra Violet** front side image of the requested document, in DIB format.

**RearImage2*

Reserved for future use. Must be set to NULL.

CodelineSW

This field will return the codeline data read using the software interpretation algorithm

CodelineHW

This field will return the codeline data read using the MICR reader . This buffer will be filled also in case Magnetic and Optical is requested.

Barcode

This field will return the barcode codeline data, either PDF417 or 2_OF_5, read using the barcode reader.

CodelinesOptical

This field will return the OCR codeline data read using the OCR reader.

**DocToRead*

Not used.

**NrPrinted*

Variable where it will be returned the progressive count number physically endorsed on the documents. A previous *LSLoadStringWithCounterEx()* call has to be done.

VerifyHole

Specify to verify if the hole of the codeline are micro perforated.

UnitMeasure

Specify whether the *Start_X, Start_Y, SizeW* and *SizeH* measures are expressed in millimeters or in inches.
The possible values are either **UNIT_MM** or **UNIT_INCH**.

nrRegions

Number of codeline to decode.

stMicroHole

Array of struct of parameter of each region, described in **Comments** section of the function *LSReadMicroHolesCodelines()*.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM

LS_INVALID_COMMAND
LS_FEEDER_EMPTY
LS_DOUBLE_LEAFING_WARNING
LS_DOUBLE_LEAFING_ERROR
LS_LOOP_INTERRUPTED
LS_REPLACE_CARTRIDGE

Comments

The image handle retrieved by this function must be released by the application.
In case more than one Optic Codeline is returned they are separated by a space character.

5.8. LSGetDocDataEx

#include "LSApi.h"

```
Result API LSGetDocDataEx( short            hConnect,
                           HWND             hWnd,
                           int              CompressionPlace,
                           unsigned long    *NrDoc,
                           short            DpiImagesJPEG,
                           HANDLE           *FrontImageJPEG,
                           int              *SizeFrontImageJPEG,
                           HANDLE           *RearImageJPEG,
                           int              *SizeRearImageJPEG,
                           short            DpiImagesTIFF,
                           HANDLE           *FrontImageTIFF,
                           int              *SizeFrontImageTIFF,
                           HANDLE           *RearImageTIFF,
                           int              *SizeRearImageTIFF,
                           short            DpiImagesBMP,
                           HANDLE           *FrontImageBMP,
                           HANDLE           *RearImageBMP,
                           LPSTR            CodelineSW,
                           LPSTR            CodelineHW,
                           LPSTR            Barcode,
                           LONG             *NrPrinted,
                           SHORT            *Reserved1,
                           LPVOID           Reserved2,
                           LPVOID           Reserved3);
```

Description

This function should be used to retrieve the information about the documents processed by LSAutoDocHandle. If no document has been processed the function returns a LS_FEEDER_EMPTY completion code.
If the LsAutoDocHandle is called with NumDocument parameter set to 0, the application must call the LSGetDocData function in a loop until LS_FEEDER_EMPTY completion code or an error code is returned.
When the function return a warning of destination sorter full, the application must continue call the function until return LS_NO_OTHER_DOCUMENT.
Note that if a double leafing occurs, this error code is returned by the LSGetDocData function itself, it is very important that the application handles this situation in the correct way.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

CompressionPlace

Valid parameter ONLY in case of LSConnect connection.
Place where the compression will be done, the value accepted are :
COMPRESSION_ON_PC : the image will be sent from the LSConnect to PC in RAW mode and the relative conversion will be done on the PC (fast way).
COMPRESSION_ON_LSCONNECT : the compression will be done on board on the LSConnect and after the images will be sent to the PC (slow way).

NrDoc

Progressive sequence number identifying the document whose data is retrieved by this function.

DpiImagesJPEG

Specified the DPI resolution of the JPEG images returned, the value accepted are :
IMAGES_RESOLUTION_100_DPI : Images at 100 dpi.
IMAGES_RESOLUTION_200_DPI : Images at 200 dpi.
IMAGES_RESOLUTION_300_DPI : Images at 300 dpi.

**FrontImageJPEG*

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in JPEG format. If this parameter is set to NULL the images will not returned.

**SizeFrontImageJPEG*

Pointer to an integer where will be returned the size of the Front image in JPEG format.

**pRearImageJPEG*

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in JPEG format. If this parameter is set to NULL the images will not returned.

**SizeRearImageJPEG*

Pointer to an integer where will be returned the size of the Rear image in JPEG format.

DpiImagesTIFF

Specified the DPI resolution of the TIFF images returned, the value accepted are :
IMAGES_RESOLUTION_100_DPI : Images at 100 dpi.
IMAGES_RESOLUTION_200_DPI : Images at 200 dpi.
IMAGES_RESOLUTION_300_DPI : Images at 300 dpi.

**FrontImageTIFF*

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in TIFF format. If this parameter is set to NULL the images will not returned.

**SizeFrontImageTIFF*

Pointer to an integer where will be returned the size of the Front image in TIFF format.

**pRearImageTIFF*

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in TIFF format. If this parameter is set to NULL the images will not returned.

**SizeRearImageTIFF*

Pointer to an integer where will be returned the size of the Rear image in TIFF format.

DpiImagesBMP

Specified the DPI resolution of the BMP images returned, the value accepted are :
IMAGES_RESOLUTION_100_DPI : Images at 100 dpi.
IMAGES_RESOLUTION_200_DPI : Images at 200 dpi.
IMAGES_RESOLUTION_300_DPI : Images at 300 dpi.

**FrontImageBMP*

Pointer to a handle where will be returned the handle of memory buffer containing the front side image of the requested document, in DIB format. If this parameter is set to NULL the images will not returned.

**RearImageBMP*

Pointer to a handle where will be returned the handle of memory buffer containing the rear side image of the requested document, in DIB format. If this parameter is set to NULL the images will not returned.

CodelineSW

This field will return the codeline data read using the software interpretation algorithm.

CodelineHW

This field will return the codeline data read using the MICR reader . This buffer will be filled also in case Magnetic and Optical is requested.

Barcode

This field will return the barcode codeline data, either PDF417 or 2_OF_5, read using the barcode reader.

**NrPrinted*

Variable where it will be returned the progressive count number physically endorsed on the documents. A previous *LSLoadStringWithCounterEx()* call has to be done.

**Reserved1*

Reserved for future use. Must be set to NULL.

**Reserved2*

Reserved for future use. Must be set to NULL.

**Reserved3*

Reserved for future use. Must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_PAPER_JAM
LS_INVALID_COMMAND
LS_FEEDER_EMPTY
LS_DOUBLE_LEAFING_WARNING
LS_DOUBLE_LEAFING_ERROR
LS_LOOP_INTERRUPTED
LS_REPLACE_CARTRIDGE

Comments

The type of compression for the JPEG format is taken from the *Quality* parameter set with the *LSAutoDocHandle()* call.
The function doesn’t return a resolution higher than the resolution provided in the *ScanMode* parameter set with the *LSAutoDocHandle()* call.
The TIFF images are returned in TIFF Gr. 4 format ONLY.
The images handles retrieved by this function must be released by the application.

5.9. LSStopAutoDocHandle

#include "LSApi.h"

Result API LSStopAutoDocHandle(**short** *hConnect,*
HWND *hWnd*);

Description

Stop the loop of automatic documents handling started with the *LSAutoDocHandle()*.
If you had started a loop of LSAutoDocHandle and you want to stop it, you must accept that some documents still need to be processed before the stop is performed. The more the application is slow in reading images and codelines, the more documents will pass before the stop is done.
Even with a very fast application which reads images as they are available, it is always possible that in the meantime a new document is fed before the LSStopAutoDocHandle is called.
This means that the stop is not istantaneous.
The documents already present in the path at the time of the *LSSTopAutoDocHandle()* are moved up to the sorter.

So the application should do the following:
LSAutoDochandle ()
Loop of LSGetDocData() until Feeder is empty or error (like paper jam).
When reply code is equal to LS_LOOP_INTERRUPTED it means that no further items will be fed but still items in the path have to be handled.
Application must continue the loop of LSGetDocData until reply code is different from :
LS_FEEDER_EMPTY or LS_NO_OTHER_DOCUMENT

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE

Comments

Remember to call the function *LSGetDocData()* until the reply is different from *LS_OKAY*, otherwise the unit remain in a pending state.

6. Magnetic stripe reader functions

The following API return the magnetic tracks encoded in a card in the form of three different strings, provided that the device has this option installed.

There are two different ways of working . The first one is to issue the read command and wait indefinitely for the card insertion.
The second one is to send a read command with a given timeout, so either the command completes with the resulting strings or with a timeout error.

Make sure to work with the magnetic stripe commands only if no other commands are outstanding to the device. No more than one outstanding command can be issued to the device at the same time.

If you issued a magnetic stripe reading command and you want to cancel it, you can call a *LSResetEx()* function with the ResetType parameter set to **RESET_ERROR** from a parallel thread.

6.1. LSReadBadge

#include "LSApi.h"

```
Result API LSReadBadge (     short       hConnect,
                             HWND        hWnd,
                             CHAR        Format,
                             SHORT       MaxLength,
                             LPSTR       String,
                             SHORT       *Length);
```

Description

This command may be used with LS devices equipped with a magnetic stripe card reader.
The function terminate when one of the following events occur :
A badge is read or the function *LSReset()* is called.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use).**

Format

Specifies which track(s) to read. Accepted values are :
FORMAT_IATA = IATA, usually associated to track 1
FORMAT_ABA = ABA, usually associated to track 2
FORMAT_MINTS = MINTS, usually associated to track 3
FORMAT_IATA_ABA = IATA + ABA, both tracks 1 and 2
FORMAT_ABA_MINTS = ABA + MINTS, both tracks 2 and 3
FORMAT_IATA _ABA_MINTS = IATA + ABA + MINTS, tracks 1, 2 and 3

MaxLength

Length of the buffer where data read from the card will be returned.

String

Buffer where data read from the card will be returned.

Length

Length of the data returned in *string*.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_DATA_TRUNCATED

Comments

The start of data of each of the card tracks read will be identified by a leading ASCII character 't' .

6.2. LSReadBadgeWithTimeout

#include "LsApi.h"

```
Result API LSReadBadgeWithTimeout(   short    hConnect,
                                      HWND    hWnd,
                                     char     Format,
                                     short    MaxLength,
                                     LPSTR    String,
                                     short    *Length,
                                     long     Timeout);
```

Description

This command may be used with LS devices equipped with a magnetic stripe card reader.
Reads badge traces, the difference with the *LSReadBadge()* is the timeout associated.
The function terminate when one of the following events occur :
A badge is read or the timeout is expired or the function *LSReset()* is called.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use).**

Format

FORMAT_IATA = IATA
FORMAT_ABA = ABA
FORMAT_MINTS = MINTS
FORMAT_IATA_ABA = IATA + ABA
FORMAT_ABA_MINTS = ABA + MINTS
FORMAT_IATA_ABA_MINTS = IATA + ABA + MINTS

MaxLength

Length of buffer where data read from badge will be returned.

String

Buffer where data read from badge will be returned.

Length

Length of data read from badge.

Timeout

Timeout value expressed in milliseconds, this parameter accept as minimum value **MIN_TIMEOUT** (500 ms).

Return Value

LS_OKAY
LS_SERVERSYSERR
LS_USBERR
LS_PERIFNOTFOUND
LS_HARDERR
LS_PERIFOFFON
LS_INVALID_TYPE_CMD

LS_INVALID_BADGE_TRACK
LS_INVALID_BADGE_TIMEOUT
LS_DATATRUNC

Comments

The traces string returned start with the character 't'.
In order to cancel a *LSReadBadgeWithTimeout()* function execution, call the *LSReset() function*.

7. Software Read Codelines functions

The following functions retrieve the codeline printed on the document with the aid of the image.
The image provided to the functions must be in BMP format.

In some cases the function requires the user to provide the window coordinates for the decoding. In other cases (like for instance with the barcode) the engine itself is able to locate the correct window and decode it.
The following support libraries are needed in case you want the availability of the Software Read Codelines functions:

CtsDecod.dll
BarDecode.dll
CtsPDF.dll
CtsPdfDriverLicenze.dll
CtsDataMatrix.dll
CtsQRCode.dll
CtsMicroHole.dll

CtsBarcodeLocate.dll
CtsRS.dll
RotateImagelib.dll

7.1. *LSCodelineReadFromBitmap*

#include "LSApi.h"

```
Result API LSCodelineReadFromBitmap(   HWND          hWnd,
                                       HANDLE        hImage,
                                       char          *CodelineType,
                                       short         UnitMeasure,
                                       float         x,
                                       float         y,
                                       float         sizeW,
                                       float         sizeH,
                                       READOPTIONS   *Option,
                                       LPSTR         Codeline,
                                       UINT          *Length);
```

Description

This function should be used to read the codeline data by means of software decoding. In this case the codeline may be located anywhere in the document.
At present the decoding engine works only on bitmaps of 100, 200 or 300 dpi.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the scanned image, in DIB format, from which to read the codeline.

CodelineType

Specify the type of codeline font to read and decode. The coordinates' starting point is the bottom right margin of the document.
READ_CODELINE_SW_OCRA = read codeline from the position specified by X, Y and Size parameters.
READ_CODELINE_SW_OCRB_NUM = read codeline from the position specified by X, Y and Size parameters.
READ_CODELINE_SW_OCRB_ALFANUM = read codeline from the position specified by X, Y and Size parameters.
READ_CODELINE_SW_OCRB_ITALY = read codeline from the position specified by X, Y and Size parameters. This font is a subset of the alfa-numeric font without the character '&' and same separator.
READ_CODELINE_SW_E13B = read codeline from the position specified by X, Y and Size parameters.
READ_CODELINE_SW_CMC7 = read codeline from the position specified by X, Y and Size parameters. The co-ordinate start from bottom right corner of the document.

UnitMeasure

Specify whether the *Start_X, Start_Y, SizeW* and *SizeH* measures are expressed in millimeters or in inches.
The possible values are either **UNIT_MM** or **UNIT_INCH**.

Start_X

Specify the x co-ordinate from right hand margin of the document. The value must be consistent with the *UnitMeasure*.

Start_Y

Specify the y co-ordinate from bottom margin of the document. The value must be consistent with the *UnitMeasure*.

SizeW

Specify the width of the window on the image bitmap that will be processed by the decoding software. The value must be consistent with the *UnitMeasure*.

SizeH

Specify the height of the window on the image bitmap that will be processed by the decoding software.

CodelineOpt

Pointer to a structure of type READOPTIONS (refer to **Comments**) that contains further options for codeline decoding.

Codeline

Pointer to the user buffer where the decoded codeline data will be copied.
If the parameter *CodelineType* is set to **READ_CODELINE_SW_OCRA** the data will start with a leading ASCII character '**A**'.

If the parameter *CodelineType* is set to either **READ_CODELINE_SW_OCRB_NUM, READ_CODELINE_SW_OCRB_ALFANUM** or **READ_CODELINE_SW_OCRB_ITALY** the data will start with a leading ASCII character '**B**'.

If the parameter *CodelineType* is set to either **READ_CODELINE_SW_E13B** or is returned a codeline E13BxOCRB the data will start with a leading ASCII character '**E**'.

If the parameter *CodelineType* is set to **READ_CODELINE_SW_CMC7** the data will start with a leading ASCII character '**H**'.

Length

Pointer to a variable that in input specifies the size of the buffer supplied by the application and pointed by *Codeline* parameter, and in output will specify the actual number of returned characters.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_INVALID_CODELINE_TYPE
LS_UNIT
LS_MISSING_IMAGE
LS_INVALID_SIZEH_VALUE
LS_OPEN_NOT_DONE

LS_DECODE_FONT_NOT_PRESENT
LS_DECODE_INVALID_COORDINATE
LS_DECODE_INVALID_OPTION
LS_DECODE_INVALID_CODELINE_TYPE
LS_DECODE_SYSTEM_ERROR
LS_DECODE_DATA_TRUNC
LS_DECODE_INVALID_BITMAP
LS_DECODE_ILLEGAL_USE

Comments

Description structure READOPTIONS:

```
typedef struct _ReadOption
{
        BOOL     PutBlanks;      // TRUE or FALSE
        char     TypeRead;       // Possible value :
                                 // READ_ONE_CODELINE_TYPE
                                 // READ_CODELINE_SW_E13B_X_OCRB
                                 // READ_CODELINE_SW_MULTI_READ
}READOPTIONS, *LPREADOPTIONS;
```

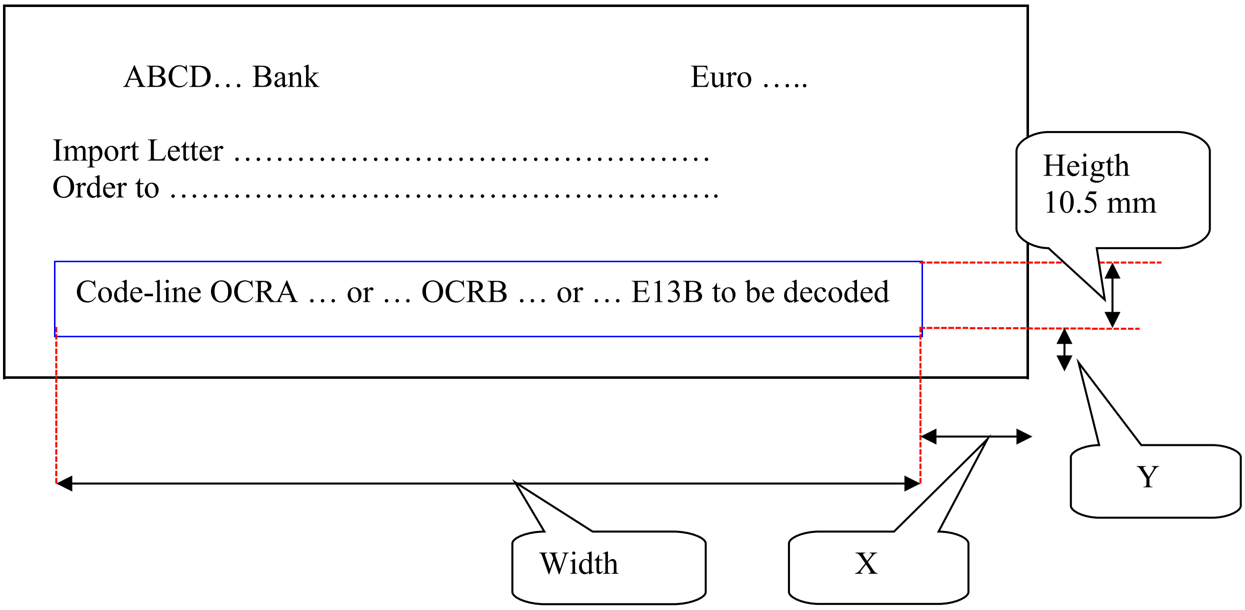
PutBanks

When equal to TRUE causes the addition of a space character between different parts of codeline data. When equal to FALSE no space character is added.

TypeRead

To decode only one type of font the parameter must be set to READ_ONE_CODELINE_TYPE.
To decode a codeline E13B x OCRB the parameter must be to set READ_CODELINE_SW_E13B_X_OCRB and the parameter *CodelineType* of the function must be set to :
CodelineType[0] = READ_CODELINE_SW_E13B
CodelineType[1] = READ_CODELINE_SW_OCRB_NUM
CodelineType[2] = '\0'

Example for take the coordinate to give at the function :


ABCD… Bank
Euro …..
Import Letter ………………………………………
Order to ……………………………………………
Code-line OCRA … or … OCRB … or … E13B to be decoded
Heigth
10.5 mm
Y
Width
X

7.2. LSReadBarcodeFromBitmap

#include "LSApi.h"

Result API LSReadBarcodeFromBitmap(	**HWND**	*hWnd*,
	HANDLE	*hImage*,
	char	*TypeBarcode*,
	int	*pos_x*,
	int	*pos_y*,
	int	*sizeW*,
	int	*sizeH*,
	LPSTR	*Codeline*,
	UINT	**Length)*;

Description

This function should be used to read barcodes 1 and 2D, by means of software decoding. In this case the barcode may be located anywhere in the document.
For decoding barcodes 1D this function work only if Bardecode.dll is available.
For decoding barcode PDF417 this function work only if CTSPdf.dll is available.
For decoding barcode DATAMATRIX this function work only if CtsDataMatrix.dll is available.
For decoding barcode QRCODE this function work only if CtsQRCode.dll is available.
There’s also the possibility to locate the barcode in automatic mode on the given image, if the parameters for the coordinate are all set to 0 (zero).
If on the document are present more then one barcode the function return only one barcode, in order to get the other call the function *LSGetNextBarcode()*.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the ***16 or 256 shades grey or color*** bitmap that contains the data to read.
PDF417 decode only on bitmap at ***16 or 256 shades of grey, at 200 or 300 dpi*** types.

TypeBarcode

Type of barcode supported :
READ_BARCODE_2_OF_5 = the library decoding a 2_OF_5 barcode.
READ_BARCODE_CODE39 = the library decoding a CODE39 barcode.
READ_BARCODE_CODE128 = the library decoding a CODE128 barcode.
READ_2D_BARCODE_PDF417 = the library decoding a PDF417 barcode.
READ_2D_BARCODE_DATAMATRIX = the library decoding a DATAMATRIX barcode.
READ_2D_BARCODE_QRCODE = the library decoding a QRCode barcode.

pos_X

Specify the co-ordinate (**in millimeters**) from right margin of the document.

pos_Y

Specify the y position (**in millimeters**) from bottom margin of the document.

SizeW

Specify the width (**in millimeters**) of the window on the image bitmap that will be processed by the decoding software to search for and decode the barcode.

SizeH

Specify the height (**in millimeters**) of the window on the image bitmap that will be processed by the decoding software to search for and decode the barcode.

Codeline

Pointer to the application buffer where the interpreted barcode data will be copied.

Length

Pointer to a variable that in input specifies the size of the application buffer pointed by *Codeline* parameter, and in output will specify the actual number of returned characters.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_MISSING_IMAGE
LS_INVALID_BARCODE_TYPE
LS_INVALID_COORDINATE
LS_OPEN_NOT_DONE

LS_BARCODE_GENERIC_ERROR
LS_BARCODE_NOT_DECODABLE
LS_BARCODE_OPENFILE_ERROR
LS_BARCODE_READBMP_ERROR
LS_BARCODE_MEMORY_ERROR
LS_BARCODE_START_NOTFOUND
LS_BARCODE_STOP_NOTFOUND

LS_PDF_NOT_DECODABLE
LS_PDF_READBMP_ERROR
LS_PDF_BITMAP_FORMAT_ERROR
LS_PDF_MEMORY_ERROR
LS_PDF_START_NOTFOUND
LS_PDF_STOP_NOTFOUND
LS_PDF_LEFTIND_ERROR
LS_PDF_RIGHTIND_ERROR
LS_PDF_OPENFILE_ERROR

Comments

7.3. LSGetNextBarcode

#include "LSApi.h"

Result API LSGetNextBarcode(**HWND** *hWnd*,
LPSTR *Codeline*,
long **Length)*;

Description

This function should be used to get the next barcode present on the document, re-call the function until return the reply LS_NO_MORE_DATA.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Codeline

Pointer to the application buffer where the interpreted barcode data will be copied.

Length

Pointer to a variable that in input specifies the size of the application buffer pointed by *Codeline* parameter, and in output will specify the actual number of returned characters.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_NO_MORE_DATA
LS_OPEN_NOT_DONE

Comments

7.4. LSReadMicroHolesCodelines

#include "LSApi.h"

```
Result API LSReadMicroHolesCodelines(  HWND              hWnd,
                                       HANDLE            hImageFront,
                                       HANDLE            hImageRear,
                                       BOOL              Reserved,
                                       short             UnitMeasure,
                                       short             nrRegions,
                                       MICROHOLE_STRUCT  stMicroHole);
```

Description

Used to read Micro Holes codelines present on the front and/or rear of the document, by means of software decoding.
At the moment this function doesn't verify the true hole.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImageFront

Handle of the scanned front image, in DIB format, that contains the data to read.

hImageRear

Handle of the scanned rear image, in DIB format, that contains the data to read.
If not available set to NULL

Reserved

Not used, set to FALSE.

UnitMeasure

Specify whether the *Start_X, Start_Y, SizeW* and *SizeH* measures are expressed in millimeters or in inches.
The possible values are either **UNIT_MM** or **UNIT_INCH**.

nrRegions

Number of codeline to decode.

stMicroHole

Array of struct of parameter of each region, described in **Comments** section.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_MISSING_IMAGE
LS_INVALID_COORDINATE
LS_OPEN_NOT_DONE

Comments

The MICROHOLE_STRUCT structure is organized as follows :

```
typedef struct _MicroHoleStruct
{
        float     x;
        float     y;
        float     width;
        float     height;
        BOOL      TrueHole;
        char      *strFront;
        short     dimStrFront;
        char      *strRear;
        short     dimStrRear;
} MICROHOLE_STRUCT, *PMICROHOLE_SCRUCT;
```

Description :

x
Specify the co-ordinate from left margin of the document.

y
Specify the co-ordinate from left margin of the document.

width
Specify the width of the window on the image bitmap that will be processed by the decoding software.

height
Specify the height of the window on the image bitmap that will be processed by the decoding software.

TrueHole
Return TRUE o FALSE if the holes are verified or no.

strFront
String decoded on the front image.

dimStrFront
Specify the dimension in byte of the *strFront* parameter.

strRear
String decoded on the rear image.

dimStrRear
Specify the dimension in byte of the *strRear* parameter.

7.5. LSReadBarcodeDriverLicense

#include "LSApi.h"

```
Result API LSReadBarcodesDriverLicense (  HWND        hWnd,
                                          HANDLE      hImage,
                                          short       encodeBase,
                                          LPSTR       Codeline_2D,
                                          int         *Length_2D,
                                          short       *ErrorRate,
                                          int         TypeBarcode_1D,
                                          LPSTR       Codeline_1D,
                                          int         *Length_1D,
                                          int         Reserved1,
                                          int         Reserved2,
                                          int         Reserved3,
                                          int         Reserved4);
```

Description

This function allows to read US Driver licenses . In those cards the following symbols can be present: PDF417 (2D barcode) and/or mono-dimensional barcodes (like code 128)
Those symbols can be present in the document in all position of this.
This function is available only if the following DLLs are present: CtdPDFDriverLicense.dll and bardecode.dll.
It is also necessary that the devices are sold with the DL option enabled.
At present the function works only on bitmaps of 300 dpi.
The output of the function will be composed of two separate strings (one for 1D and another for 2D barcodes) . Those results can be provided as normal strings or strings coded in 64 base .
The function will also be able to provide the following information:

- Whether the card was passed in the correct direction
- Whether the card was passed upside down
- Whether the card was passed correctly but the barcodes cannot be decoded
- Whether no barcodes were found in the image provided.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the ***16 shades of grey, 300 dpi*** bitmap that contains the data to be read. (not advised)
Handle of the ***256 shades of grey, 300 dpi*** bitmap that contain the data to be read.
Handle of **color, *300 dpi*** bitmap that contain the data to be read.

encodeBase

Possible values:
ENCODE_NO
ENCODE_BASE_64

Codeline_2D

Pointer to the user buffer where the decoded 2D barcode data will be copied.

Length_2D

Pointer to a variable that in input specifies the size of the buffer supplied by the application and pointed by *Codeline_2D* parameter, and in output will specify the actual number of returned characters. In case the card does not include 2D barcodes, the Length value returned is 0.

ErrorRate

It returns the number of characters corrected by the decoding algorithm.

TypeBarcode_1D

It specifies the type of 1D barcode to be read. Possible values:
READ_BARCODE_2_OF_5
READ_BARCODE_CODE39
READ_BARCODE_CODE128 (default)

Codeline_1D

Pointer to the user buffer where the decoded 1D barcode data will be copied.

Length_1D

Pointer to a variable that in input specifies the size of the buffer supplied by the application and pointed by *Codeline_1D* parameter, and in output will specify the actual number of returned characters. In case the card does not include 1D barcodes, the Length value returned is 0.

Reserved1

Reserved for future improvement, must be set to NULL.

Reserved2

Reserved for future improvement, must be set to NULL.

Reserved3

Reserved for future improvement, must be set to NULL.

Reserved4

Reserved for future improvement, must be set to NULL.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_MISSING_IMAGE
LS_OPEN_NOT_DONE

LS_PDF_NOT_DECODABLE
LS_PDF_READBMP_ERROR
LS_PDF_BITMAP_FORMAT_ERROR
LS_PDF_MEMORY_ERROR
LS_PDF_START_NOTFOUND
LS_PDF_STOP_NOTFOUND
LS_PDF_LEFTIND_ERROR
LS_PDF_RIGHTIND_ERROR
LS_PDF_OPENFILE_ERROR
LS_PDF_LOCALIZATION_ERROR
LS_PDF_TOO_MANY_ERROR

Comments

Output of the function will be if the barcode :

LS_OKAY: in this case the image passed to the function includes the barcodes (rear side). The application can use front image to show it on the screen. Depending on the type of DL, it is possible that not all of the barcodes are present in the image. The application can check *Length_2D and Length_1D.*

LS_PDF417_UPSIDE_DOWN: a warning message indicating that barcode was detected from the image but the card is upside down. **<u>The application must rotate the front image 180°</u> before showing it on the screen.**

LS_PDF417_NOT_DECODABLE : this means that the 2D barcode can be located in the image but it is not decodable. In this case the application can show the front image on the screen as it is. (this function is under development)

LS_PDF417_NOT_DECODABLE_UPSIDE_DOWN : this means that the 2D barcode can be located in the image but it is not decodable and the card is passed upside-down. IN this case the application can show the front image on the screen rotating it 180°.
(this function is under development)

LS_BARCODE_NOT_PRESENT : this means that on the image neither the 2D barcode nor the 1D barcode are present, it is possible that the card is passed in the wrong direction. The application must call the function again passing the front image handle, then rely on the next reply code.

8. Image manipulation functions

The functions described in this section provide additional image processing capabilities to the application.

The following functions are available for image processing.

LSSaveJPEG
LSSaveDIB
LSSaveTIFFEx
LSDisplayImage
LSUpdateImage
LSConvertImageToBW
LSConvertToJPEG
LSConvertToTIFF
LSEnableImageCorrection
LSFreeImage
LSRotateImage
LSConvertImage200To100dpi
LSImageBrightness
LSImageContrast
LSCutImage

NOTE: although Windows allows users to create pathnames longer than this, there is the limitation that the pathname of the file to be saved with CTS Image manipulation functions cannot exceed 128 bytes.

8.1. LSSaveJPEG

#include "LSApi.h"

```
Result API LSSaveJPEG(    HWND        hWnd,
                          HANDLE      hImage,
                          int         Quality,
                          LPSTR       filename );
```

Description

This function allows to save the scanned image in JPEG format.

Parameters

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

hImage

Handle of the image to save in **JPEG** format.

Quality

Indicates the relationship between quality and compression range 2 to 255 that :
2 = max quality.
255 = max compression.

Filename

Full pathname of the file where the image will be saved. (max 128)

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_NO_LIBRARY_LOAD
LS_INVALID_QUALITY
LS_MISSING_IMAGE
LS_JPEG_ERROR

Comments

8.2. LSSaveTIFFEx

```
#include "LSApi.h"

Result API LSSaveTIFFEx(HWND      hWnd,
                        HANDLE    hImage,
                        LPSTR     filename,
                        int       Type,
                        int       Quality,
                        int       SaveMode,
                        int       PageNumber );
```

Description

This function allows to save the scanned image in TIF format (included TIFF_JPEG).

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the image to save in **TIF** format.

Filename

Name of the file where the image will be saved. (max 128)

Type

Specify the TIF format
FILE_TIF = Tagged Image File Format
SAVE_TIF_JPEG = Image TIFF saved in JPEG format.
FILE_CCITT = TIFF CCITT, only b/w image
FILE_CCITT_GROUP3_1DIM = CCITT Group3 one dimension, only b/w image
FILE_CCITT_GROUP3_2DIM = CCITT Group3 two dimensions, only b/w image
FILE_CCITT_GROUP4 = CCITT Group4 two dimensions, only b/w image.

Quality

Valid only with the parameter *Type* set to **SAVE_TIF_JPEG**, indicates the relationship between quality and compression range 2 to 255 that :
2 = max quality.
255 = max compression.

SaveMode

SAVE_OVERWRITE = Overwrite the file.
SAVE_APPEND = Append the image in a multi-page image file.
SAVE_REPLACE = Replace a image in a multi-page image file.
SAVE_INSERT = Insert the image in a multi-page image file.

PageNumber

Number indicating the position in a multi-page file.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE

LS_INVALID_TYPE
LS_INVALID_QUALITY
LS_INVALID_SAVEMODE
LS_INVALID_PAGE_NUMBER
LS_TIFF_ERROR

Comments

8.3. LSSaveDIB

#include "LSApi.h"

```
Result API LSSaveDIB(    HWND        hWnd,
                         HANDLE      hImage,
                         LPSTR       filename );
```

Description

This function allows to save the scanned image in Bitmap format.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the image to save in **BMP** format.

Filename

Name of the file where the image will be saved. (max 128)

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_BMP_ERROR

Comments

8.4. LSConvertToJPEG

#include "LSApi.h"

```
Result API LSConvertToJPEG(   HWND     hWnd,
                              HANDLE   hImage,
                              int      Quality,
                              HANDLE   *JpegImage,
                              long     *ImageSize);
```

Description

This function allows to convert the scanned image in JPEG format in a memory buffer.

Parameters

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

hImage

Handle of the BMP image to convert in **JPEG** format.

Quality

Indicates the relationship between quality and compression range 2 to 255 that :
2 = max quality.
255 = max compression.

JpegImage

Pointer to a handle where will be returned the handle of a memory buffer containing the converted image in JPEG format.

ImageSize

Pointer to a variable where will be returned the size of the converted image.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_NO_LIBRARY_LOAD
LS_INVALID_QUALITY
LS_MISSING_IMAGE
LS_JPEG_ERROR

Comments

The image (in BMP format) to be converted must be supplied by the application. ***The JPEG image returned by this function must be released by the application*** using the ***LSFreeImage*** function.

8.5. LSConvertToTIFF

#include "LSApi.h"

```
Result API LSConvertToTIFF( HWND          hWnd,
                            HANDLE        hImage,
                            int           Type,
                            int           Quality,
                            int           SaveMode,
                            int           PageNumber
                            HANDLE        *TiffImage,
                            long          *ImageSize );
```

Description

This function allows to convert the scanned image in TIF format in a memory buffer.

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the image to be converted in **TIF** format.

Type

Specify the TIF format
FILE_TIF = Tagged Image File Format
SAVE_TIF_JPEG = Image TIFF saved in JPEG format.
FILE_CCITT = TIFF CCITT, only b/w image
FILE_CCITT_GROUP3_1DIM = CCITT Group3 one dimension, only b/w image
FILE_CCITT_GROUP3_2DIM = CCITT Group3 two dimensions, only b/w image
FILE_CCITT_GROUP4 = CCITT Group4 two dimensions, only b/w image.

Quality

Valid only with the parameter *Type* set to **SAVE_TIF_JPEG**, indicates the relationship between quality and compression range 2 to 255 that :
2 = max quality.
255 = max compression.

SaveMode

SAVE_OVERWRITE = Overwrite the file.
SAVE_APPEND = Append the image in a multi-page image file.
SAVE_REPLACE = Replace a image in a multi-page image file.
SAVE_INSERT = Insert the image in a multi-page image file.

PageNumber

Number indicating the position in a multi-page file.

TiffImage

Pointer to a handle where will be returned the handle of a memory buffer containing the converted image in TIFF format.

ImageSize

Pointer to a variable where will be returned the size of the converted image.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE
LS_INVALID_TYPE
LS_INVALID_QUALITY
LS_INVALID_SAVEMODE
LS_INVALID_PAGE_NUMBER
LS_TIFF_ERROR

Comments

The image (in BMP format) to be converted must be supplied by the application. ***The TIFF image returned by this function must be released by the application*** using the ***LSFreeImage*** function.

8.6. LSConvertImageToBW

#include "LSApi.h"

```
Result API LSConvertImageToBW(     HWND         hWnd,
                                   SHORT        Method,
                                   HANDLE       GrayImage,
                                   LPHANDLE     BwImage,
                                   short        Parameter1,
                                   float        Threshold);
```

Description

This function can be used by client application to convert an image from 16 or 256 shades of grey to black and white.

Parameters

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

Method

Specify the conversion algorithm to apply :
ALGORITHM_CTS : Use a proprietary CTS algorithm.
ALGORITHM_CTS_2 : Use a proprietary CTS algorithm.
ALGORITHM_CTS_3 : Use a proprietary CTS algorithm.
ALGORITHM_CTS_TOP_IMAGE : Use a proprietary CTS algorithm, need CtsTopImage.dll library.
ALGORITHM_CTS_IMAGE_PRO : Use a proprietary CTS algorithm, need CtsImagePro.dll library.
ALGORITHM_CTS_CLEAR_PIX : Use a proprietary CTS algorithm, need CtsCLearPIX.dll library.
ALGORITHM_NODITHERING : Use nearest color matching.
ALGORITHM_FLOYDSTEINDITHERING : Use Floyd-Steinberg dithering.
ALGORITHM_STUCKIDITHERING : Use Stucki dithering.
ALGORITHM_BURKESDITHERING : Use Burkes dithering.
ALGORITHM_SIERRADITHERING : Use Sierra dithering.
ALGORITHM_STEVENSONARCEDITHERING : Use Stevenson Arce dithering.
ALGORITHM_JARVISDITHERING : Use Jarvis dithering.

GrayImage

Handle of memory buffer containing an image, in DIB format, supplied by the application for conversion. The handle of the converted image will be returned into BwImage.

BwImage

Pointer to a handle where will be returned the handle of a memory buffer containing the converted b/w image of the document, in DIB format.

Parameter1

This parameter is related to the *Method* parameter.
When *Method* is equal to :

ALGORITHM_CTS The valid range is from 50 to 600, the default is 450 (**DEFAULT_POLO_FILTER**). For documents with a clear background the value must be in the low range (50..450), for documents with a darkened background the value must be in the higher range (450..600).

ALGORITHM_CTS_3 The valid range is from 0 to 15, the default is 8 (**DEFAULT_CTS_3_TRHESHOLD**).

Threshold

This parameter is meaningful only if ***ALGORITHM_CTS*** is chosen as *Method*. The valid range is from 0.50 to 1.
The default value when this parameter is set to 0 is 0.90.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_NO_LIBRARY_LOAD
LS_INVALID_METHOD
LS_INVALID_POLO_FILTER

Comments

The grey scale image (in DIB format) to be converted must be supplied by the application. ***The BW image returned by this function must be released by the application*** using the ***LSFreeImage*** function. This function can be used only after a successful **LSConnect**.

8.7. LSConvertImageToBWWithReport

#include "LSApi.h"

```
Result API LSConvertImageToBWWithReport( HWND        hWnd,
                                         SHORT       Method,
                                         HANDLE      GrayImage,
                                         LPHANDLE    BwImage,
                                         short       Parameter1,
                                         float       Threshold,
                                         int         *histogram,
                                         int         *Noise,
                                         int         *WhitePixel,
                                         int         *BlackPixel);
```

Description

This function can be used by client application to convert an image from 16 or 256 shades of grey to black and white. In output the function return some parameters which can be used by the application to evaluate the characteristics of the resulting bitonal image.

Parameters

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

Method

Specify the conversion algorithm to apply :
ALGORITHM_CTS : Use a proprietary CTS algorithm.
ALGORITHM_CTS_2 : Use a proprietary CTS algorithm.
ALGORITHM_CTS_3 : Use a proprietary CTS algorithm.
ALGORITHM_CTS_TOP_IMAGE : Use a proprietary CTS algorithm, need CtsTopImage.dll library.
ALGORITHM_CTS_IMAGE_PRO : Use a proprietary CTS algorithm, need CtsImagePro.dll library.
ALGORITHM_CTS_CLEAR_PIX : Use a proprietary CTS algorithm, need CtsClearPIX.dll library.
ALGORITHM_NODITHERING : Use nearest color matching.
ALGORITHM_FLOYDSTEINDITHERING : Use Floyd-Steinberg dithering.
ALGORITHM_STUCKIDITHERING : Use Stucki dithering.
ALGORITHM_BURKESDITHERING : Use Burkes dithering.
ALGORITHM_SIERRADITHERING : Use Sierra dithering.
ALGORITHM_STEVENSONARCEDITHERING : Use Stevenson Arce dithering.
ALGORITHM_JARVISDITHERING : Use Jarvis dithering.

GrayImage

Handle of memory buffer containing an image, in DIB format, supplied by the application for conversion. The handle of the converted image will be returned into BwImage.

BwImage

Pointer to a handle where will be returned the handle of a memory buffer containing the converted b/w image of the document, in DIB format.

Parameter1

This parameter is related to the *Method* parameter.
When *Method* is equal to :
ALGORITHM_CTS The valid range is from 50 to 600, the default is 450

(**DEFAULT_POLO_FILTER**). For documents with a clear background the value must be in the low range (50..450), for documents with a darkened background the value must be in the higher range (450..600).

ALGORITHM_CTS_3 The valid range is from 0 to 15, the default is 8 (**DEFAULT_CTS_3_TRHESHOLD**).

Threshold

This parameter is meaningful only if *ALGORITHM_CTS* is chosen as *Method*. The valid range is from 0.50 to 1.
The default value when this parameter is set to 0 is 0.90.

histogram

Pointer to an array of 16 or 256 intergers values where it will be returned the number of pixels found for each gray level.

Noise

Pointer to an integer where it will be returned the number of spots 2x2 found after the convertion.

WhitePixel

Pointer to an integer where it will be returned the total number of white pixels present in the bitmap after the convertion.

BlackPixel

Pointer to an integer where it will be returned the total number of black pixels present in the bitmap after the convertion.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_NO_LIBRARY_LOAD
LS_INVALID_METHOD
LS_INVALID_POLO_FILTER

Comments

The grey scale image (in DIB format) to be converted must be supplied by the application. ***The BW image returned by this function must be released by the application*** using the ***LSFreeImage*** function. This function can be used only after a successful **LSConnect**.

8.8. *LSSetBinarizationParameters*

#include "LSApi.h"

```
Result API LSSetBinarizationParameters (short      hConnect,
                                        HWND       hWnd,
                                        Short      Method,
                                        short      Threshold,
                                        float      Reserved);
```

Description

The function set the Method and the threshold used to convert the images to BW from the functions *LSAutoDocHandle()* and *LSDocHandle()* when the parameter ScanMode is set to SCAN_MODE_BW. The method and threshold value is used in the CTS Black and White convertion algorithm which is used by default.
If this function is not explicity called from the application the BW convertion is done with *method* = ALGORITHM_CTS and *Threshold* = 450.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Method

The valid value are :
ALGORITHM_CTS : Use a proprietary CTS algorithm. (Set by **default**)
ALGORITHM_CTS_3 : Use a proprietary CTS algorithm.
ALGORITHM_CTS_TOP_IMAGE : Use a proprietary CTS algorithm, need CtsTopImage.dll library.
ALGORITHM_CTS_IMAGE_PRO : Use a proprietary CTS algorithm, need CtsImagePro.dll library.
ALGORITHM_CTS_CLEAR_PIX : Use a proprietary CTS algorithm, need CtsClearPIX.dll library.

Threshold

This parameter is related to the *Method* parameter.
When *Method* is equal to :

ALGORITHM_CTS The valid range is from 50 to 600, the default is 450 (**DEFAULT_POLO_FILTER**). For documents with a clear background the value must be in the low range (50..450), for documents with a darkened background the value must be in the higher range (450..600).

ALGORITHM_CTS_3 The valid range is from 0 to 15, the default is 8.

Reserved

Reserved for future improvement, must be set to NULL.

Return Value

LS_OKAY
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_INVALID_THRESHOLD

Comments

8.9. *LSConvertImageResolution*

#include "LSApi.h"

```
Result API LSConvertImageResolutioni( HWND      hWnd,
                                      HANDLE    hImage,
                                      int       NewResolution,
                                      HANDLE    *pImage);
```

Description

Convert a BW, grey or color image from 100, 200 or 300 dpi to another requested resolution.

Parameters

hWnd

Handle of the application windows which will receive the notification messages. **(Reserved for future use)**

hImage

Handle of memory buffer containing an image, in DIB format, supplied by client for conversion. The handle of the converted image will be returned into pImage.

NewResolution

Resolution required.

pImage

Pointer to a handle where will be returned the handle of memory buffer containing the converted image of the document in DIB format.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE
LS_RESIZE_ERROR

Comments

The image returned from this function must be released by the application with LsFreeImage() function.

8.10. LSConvertImageColorTo256Gray

#include "LSApi.h"

```
Result API LSConvertImageColorTo256Grayi( HWND      hWnd,
                                          HANDLE    hImage,
                                          HANDLE    *pImage);
```

Description

Convert a color image to 256 gray.

Parameters

hWnd

Handle of the application windows which will receive the notification messages. **(Reserved for future use)**

hImage

Handle of memory buffer containing an image, in DIB format, supplied by client for conversion. The handle of the converted image will be returned into pImage.

pImage

Pointer to a handle where will be returned the handle of memory buffer containing the converted image of the document in DIB format.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE

Comments

This function can be used by client application to convert a *color* image to gray image. The color scale image (in DIB format) to be converted must be supplied by the application. ***The image returned from this function must be released by the application with LsFreeImage() function.***

8.11. LSConvertImage256To16Gray

#include "LSApi.h"

```
Result API LSConvertImage256To16Grayi(  HWND        hWnd,
                                        HANDLE      hImage,
                                        HANDLE      *pImage);
```

Description

Convert a image of 256 gray to 16 gray scale.

Parameters

hWnd

Handle of the application windows which will receive the notification messages. **(Reserved for future use)**

hImage

Handle of memory buffer containing an image, in DIB format, supplied by client for conversion. The handle of the converted image will be returned into pImage.

pImage

Pointer to a handle where will be returned the handle of memory buffer containing the converted image of the document in DIB format.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE

Comments

This function can be used by client application to convert a *256 gray* image to *16 gray* image. The 256 gray scale image (in DIB format) to be converted must be supplied by the application. ***The image returned from this function must be released by the application with LsFreeImage() function.***

8.12. LSEnableImageCorrection

#include "LSApi.h"

Result API LSEnableImageCorrection(**HWND** *hWnd*,
BOOL *fDo*);

Description

This function either enables or disables a software filter used to give a lighter rendering of the scanned image.

Parameters

hWnd

Handle of the application windows which will receive the notification messages.

fDo

TRUE enable the filter
FALSE disable the filter.

Return Value

LS_OKAY
LS_OPEN_NOT_DONE

Comments

The filter is disabled by default.

8.13. LSFreeImage

#include "LSApi.h"

Result API LSFreeImage (	**HWND**	*hWnd*,
	LPHANDLE	*hImage*);

Description

Free memory allocated by **LSReadImage()**, **LS_ReadImagePiece()**, **LSGetDocData()**, **LSConvertImageToBW(),LSConvertImage200To100dpi, LSRotateImage, LSImageBrightness, LSImageContrast, LSCutImage** functions**.**

Parameters

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

hImage

Pointer to handle of image to be released.

Return Value

LS_OKAY
LS_INVALID_HANDLE
LS_NO_LIBRARY_LOAD

Comments

Use this function also to release all image memory returned from the APIs function.

8.14. LSRotateImage

#include "LSApi.h"

Result API LSRotateImage(	**HWND**	*hWnd*,
	HANDLE	*hImage,*
	int	*degree,*
	HANDLE	**pImage)*;

Description

The function returns a copy of the image passed in input rotated by the specified number of degrees. The application must free (release) the memory areas allocated for both input and output images. Freeing one image area does not affect the other.

Parameters

hWnd

Handle of the application windows which will receive the notification messages.

hImage

Handle of the image returned by L*SReadImage()*.

degree

Number of degrees to rotate (+/-). This can be a number from 1 to 360. A positive value will rotate the image in a clockwise rotation, while a negative value will rotate the image in a counter-clockwise rotation.

pImage

Pointer to a handle where will be returned the handle of the memory buffer containing the rotated image, in DIB (BMP) format.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_INVALID_TYPE_COMMAND
LS_EXECCMD
LS_CMDSEQUENCEERROR
LS_INVALID_DEGREE
LS_MISSING_IMAGE
LS_OPEN_NOT_DONE

Comments

The memory areas allocated for both images must be released with the function *LSFreeImage()*.

8.15. LSImageBrightness

#include "LSApi.h"

```
Result API LSImageBrightness( HWND      hWnd,
                              HANDLE    hImage,
                              int       nChange,
                              HANDLE    *hNewImage);
```

Description

The function changes the intensity (brightness) of the image in a bitmap, it returns a copy of the given image.

Parameters

hWnd

Handle of the application windows which will receive the notification messages.

hImage

Handle of the image returned from L*SReadImage()*.

nChange

Amount to change the intensity. The intensity ranges from -1000 to 1000. A positive value increases (or lightens) the brightness of the bitmap image. A negative values decreases (or darkens) the brightness of the bitmap image.

hNewImage

Pointer to a handle where will be returned the handle of the memory buffer containing the image modified in DIB format.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_INVALID_TYPE_COMMAND
LS_EXECCMD
LS_CMDSEQUENCEERROR
LS_INVALID_NCHANGE
LS_MISSING_IMAGE
LS_OPEN_NOT_DONE
LS_BRIGHTNESS_ERROR

Comments

The images must be free with the function *LSFreeImage()*.

8.16. LSImageContrast

#include "LSApi.h"

```
Result API LSImageContrast( HWND      hWnd,
                            HANDLE    hImage,
                            int       nChange,
                            HANDLE    *hNewImage);
```

Description

The function increases or decreases the contrast of the image in a bitmap, it return a copy of the given image.

Parameters

hWnd

Handle of the application windows which will receive the notification messages.

hImage

Handle of the image returned from *LSReadImage()*.

nChange

Amount of contrast change. The contrast ranges from -1000 to 1000. A positive value increases the contrast of the bitmap image. A negative values decreases the contrast of the bitmap image.

hNewImage

Pointer to a handle where will be returned the handle of the memory buffer containing the image modified in DIB format.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_INVALID_TYPE_COMMAND
LS_EXECCMD
LS_CMDSEQUENCEERROR
LS_INVALID_NCHANGE
LS_MISSING_IMAGE
LS_OPEN_NOT_DONE
LS_CONTRAST_ERROR

Comments

The images must be released with the function *LSFreeImage()*.

8.17. LSCutImage

#include "LSApi.h"

```
Result API LSCutImage( HWND      hWnd,
                       HANDLE    hImage,
                       short     UnitMeasure,
                       float     Start_x,
                       float     Start_y,
                       float     sizeW,
                       float     sizeH,
                       HANDLE    *hNewImage);
```

Description

The function return a portion of the given image according to the rectangle provided.

Parameters

hWnd

Handle of the application windows which will receive the notification messages.

hImage

Handle of the image returned from *LSReadImage()*.

UnitMeasure

Specify if the *Start_X, Start_Y* and *SizeW* measures are give in millimeter or in inch.
The values are **UNIT_MM** and **UNIT_INCH**.

Start_X

Specify the x co-ordinate from right margin of the document.

Start_Y

Specify the y co-ordinate from bottom margin of the document.

SizeW

Specify the width of the window to be cut in the document bitmap.

SizeH

Specify the height of the window to be cut in the document bitmap.

hNewImage

Pointer to a handle where will be returned the handle of the memory buffer containing the portion of the image in DIB format.

Return Value

LS_OKAY
LS_NO_LIBRARY_LOAD
LS_INVALID_TYPE_COMMAND
LS_EXECCMD
LS_CMDSEQUENCEERROR
LS_UNIT_PARAM
LS_MISSING_IMAGE
LS_INVALID_COORDINATE
LS_OPEN_NOT_DONE

Comments

The images must be released with the function *LSFreeImage()*.

8.18. LSDisplayImage

#include "LSApi.h"

```
Result API LSDisplayImage (   HWND      hWnd,
                              HANDLE    Hinstance
                              CHAR      *FilenameFront,
                              CHAR      *FilenameBack,
                              INT       Xfront,
                              INT       Yfront,
                              INT       XBack,
                              INT       YBack,
                              INT       FrontWidth,
                              INT       FrontHeight,
                              INT       BackWidth,
                              INT       BackHeight,
                              HWND      *RetHwndFront,
                              HWND      *RetHwndBack);
```

Description

This function should be used to display front and back images read from a file.

Parameters

hWnd

Handle of the application windows which will receive the notification messages ***(Reserved for future use)***.

HInstance

Handle of the application instance.

FilenameFront

Pathname of the file containing the front image to display.

FilenameBack

Pathname of the file containing the rear image to display.

Xfront

The *x* parameter is the initial x co-ordinate of the window's upper-left corner, in screen co-ordinates, of the front image.

Yfront

The Y parameter is the initial Y co-ordinate of the window's upper-left corner, in screen co-ordinates of the front image.

Xback

The *x* parameter is the initial x co-ordinate of the window's upper-left corner, in screen co-ordinates of the back image.

Yback

The Y parameter is the initial Y co-ordinate of the window's upper-left corner, in screen co-ordinates of the back image.

FrontWidth

Width of front image.

FrontHeight

Height of front image.

BackWidth

Width of back image.

BackHeight

Height of back image.

RetHwndFront

Handle of the new window that contain the front image.

RetHwndBack

Handle of the new window that contain the back image.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_INVALID_COMMAND
LS_NO_LIBRARY_LOAD
LS_INVALID_HANDLE

Comments

8.19. LSUpdateImage

#include "LSApi.h"

Result API LSUpdateImage (	**HWND**	*hWnd*,
	CHAR	**FilenameFront*,
	CHAR	**FilenameBack*,
	HWND	**RetHwndFront*,
	HWND	**RetHwndBack*);

Description

This function displays front and back images read from file. **It must be used after a LSDisplayImage command.**

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use).**

FilenameFront

Pathname of the front image to display.

FilenameBack

Pathname of the back image to display.

RetHwndFront

Handle of the window that contain the new front image.

RetHwndBack

Handle of the window that contain the new back image.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_INVALID_COMMAND
LS_NO_LIBRARY_LOAD
LS_INVALID_HANDLE

Comments

9. Ultra Violet Image functions

These functions are available for the LS150-UV version.

The documents must be handled with the usual functions described in the ***Basic functions*** and in the ***Advanced Documents handling*** sections of this manual.

In both cases the *Scanmode* parameter must be set to **SCAN_MODE_256GR100_AND_UV** or **SCAN_MODE_256GR200_AND_UV.**

The device returns per each scanned document two different images:

1. a normal grey scale image
2. an UV image

The application can also request a third image which is the combination of the grey scale image + the UV image. This is done via API with the **LSMergeImageGrayAndUV()** function.

Before feeding the documents, the application can set some parameters for the UV.
This is done with the **LSModifyPWMUltraViolet()** function where it is possible to change the following :

1. increase or decrease the UV intensity light from the standard calibration of the device (in percentage). This is particularly useful in checks that for example have a very low intensity in the UV ink.
2. set the device in such a way that it gives a lower contrast mantaining part of the background of the check. In alternative it can be set in such a way that the background is completely removed, only the UV ink is captured and the resulting image has only two values: light pixels values for the UV and black pixels where there is not presence of UV ink. In this way of working the application can make use of the **LSMergeImageGrayAndUV()** function to get a combination of grey scale + UV image to get an image which can be displayed or visually analyzed.

The UV image obtained by the scanner can also be converted in bitonal vi athe function **LSConvertUVtoBWEx**()
This function allows you to specify whether the UV patterns have to be represented in the bitmap in white or in black color.

9.1. LSModifyPWMUltraViolet

#include "LSApi.h"

```
Result API LSModifyPWMUltraViolet (   short      hConnect,
                                      HWND       hWnd,
                                      short      PWMValue,
                                      BOOL       HighContrast,
                                      short      Reserved);
```

Description

Available only for the LS150 model.
Modify the default PWM value optained with the scanner Ultra Violet calibration.
When the unit is powered off and on the PWM value is set to the original value set with the calibration procedure of the scanner.
This function can be used to increase or decrease the UV intensity light and also to set two different way of working: one returns the UV image with a high contrast, while the other option gives a lower contrast keeping part of the background of the check.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

PWMValue

Modify the value of PWM in percentage, the range acceptable is from 1 to 200, the value of 100 set the value obtained with the calibration.

HighContrast

Exalt the contrast between the normal and the ultra violet ink.

Reserved

Reserved for future use, must be set to NULL.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_HARDWARE_ERROR
LS_INVALID_PWM_VALUE

Comments

9.2. LSMergeImageGrayAndUV

#include "LSApi.h"

```
Result API LSMergeImageGrayAndUV( HWND      hWnd,
                                  HANDLE    hImageGray,
                                  HANDLE    hImageUV,
                                  float     Reserved1,
                                  float     Reserved2,
                                  HANDLE    *pImage);
```

Description

Merge the two image given in input.

Parameters

hWnd

Handle of the application windows which will receive the notification messages. **(Reserved for future use)**

hImageGray

Handle of memory buffer containing a gray scale image at 256 colors returned from the device to be merged with the UV image.

hImageUV

Handle of memory buffer containing a Ultra Violet image at 256 colors returned from the device for merged.

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2

Reserved for future use. Must be set to NULL.

pImage

Pointer to a handle where will be returned the handle of memory buffer containing the merged image of the document in DIB format.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE
LS_IMAGE_NOT_256_COLOR

Comments

This function can be used by client application to merge a *Gray* image with to *Ultra Violet* image. The gray and the Ultra Violet images (in DIB format) to be mergered must be supplied by the application. ***The image returned from this function must be released by the application with LsFree function.***

9.3. LSMergeImageColorAndUV

#include "LSApi.h"

```
Result API LSMergeImageColorAndUV( HWND      hWnd,
                                   HANDLE    hImageColor,
                                   HANDLE    hImageUV,
                                   float     Reserved1,
                                   float     Reserved2,
                                   HANDLE    *pImage);
```

Description

Merge the two image given in input.

Parameters

hWnd

Handle of the application windows which will receive the notification messages. **(Reserved for future use)**

hImageColor

Handle of memory buffer containing a color scale image returned from the device to be merged with the UV image.

hImageUV

Handle of memory buffer containing a Ultra Violet image at 256 colors returned from the device for merged.

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2

Reserved for future use. Must be set to NULL.

pImage

Pointer to a handle where will be returned the handle of memory buffer containing the merged image of the document in DIB format.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE
LS_INVALID_BIT_DEPTH

Comments

This function can be used by client application to merge a *Corol* image with to *Ultra Violet* image. The color and the Ultra Violet images (in DIB format) to be mergered must be supplied by the application. ***The image returned from this function must be released by the application with LsFree function.***

9.4. LSConvertUVtoBWEx

#include "LSApi.h"

```
Result API LSConvertUVtoBWEx( HWND      hWnd,
                              HANDLE    hImageGray,
                              HANDLE    hImageUV,
                              float     Reserved1,
                              float     Reserved2,
                              int       UVImageType
                              HANDLE    *pImage);
```

Description

Convert the image UV in bitonal.

Parameters

hWnd

Handle of the application windows which will receive the notification messages. **(Reserved for future use)**

hImageGray

Not used set to NULL.
Future enanchement - Handle of memory buffer containing a gray scale image at 256 colors returned from the device.

hImageUV

Handle of memory buffer containing a Ultra Violet image at 256 color, returned from the device.

Reserved1

Reserved for future use. Must be set to NULL.

Reserved2

Reserved for future use. Must be set to NULL.

UVImageType

The accepted values are :
UV_IMAGE_NORMAL : Return the image with UV in white on a black background. (**default**)
UV_IMAGE_REVERSE : Return the image with UV in black on a white background.

pImage

Pointer to a handle where will be returned the handle of memory buffer containing the image binarizated in DIB format.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_MISSING_IMAGE
LS_IMAGE_NOT_256_COLOR

Comments

The image returned from this function must be released by the application with LsFree

function.

10. Digital Print functions

In case where the ink-jet option is not present on the device and the image needs to be endorsed, then the digital Print functions can be used.
These functions can use any kind of Windows font and print in any place of the image an invalidation string.
This invalidation string is inserted electronically in the image and can be saved afterwards in the file for archiving.

10.1. LSDigitalPrint

#include "LSApi.h"

```
Result API LSDigitalPrint( HWND          hWnd,
                          HANDLE        hImage,
                          char          *String,
                          short          Length,
                          char          *fontName,
                          int           fontSize,
                          BOOL          bold,
                          BOOL          italic,
                          BOOL          underline,
                          long          TextColor,
                          short         UnitMeasure,
                          float         pos_x,
                          float         pos_y);
```

Description

Prints a digital invalidation string, in a given image (in BMP format)

Parameters

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

hImage

Handle of the scanned image, in DIB format, where to print the invalidation *String*.

String

Invalidation string.

Length

Length of the string give in the *String* parameter.

fontName

Name of the window font selected.

fontSize

Size of the character font.

bold

Set character Bold stile.
TRUE = Stile selected.
FALSE = Stile unselected

italic

Set character Italic stile.
TRUE = Stile selected.
FALSE = Stile unselected

underline

Set character Underline stile.
TRUE = Stile selected.
FALSE = Stile unselected

TextColor

Gray tone of the character printed, the range value is from 0 (black) to 255 (white).

UnitMeasure

Specify whether the *pos_x,* and *pos_y* measures are expressed in millimeters or in inches. The possible values are either **UNIT_MM** or **UNIT_INCH**.

pos_x

Specify the x co-ordinate from left margin of the document. The value must be consistent with the *UnitMeasure*.

pos_y

Specify the y co-ordinate from top margin of the document. The value must be consistent with the *UnitMeasure*.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_SYSTEM_ERROR
LS_INVALID_UNIT

Comments

10.2. LSLoadDigitalStringWithCounter

#include "LSApi.h"

```
Result API LSLoadDigitalStringWithCounter( short          hConnect,
                                           HWND           hWnd,
                                           char           Side,
                                           char           *String,
                                           short           Length,
                                           unsigned long StartNumber,
                                           short           Step,
                                           char           *fontName,
                                           int            fontSize,
                                           BOOL           bold,
                                           BOOL           italic,
                                           BOOL           underline,
                                           long           TextColor,
                                           short          UnitMeasure,
                                           float          pos_x,
                                           float          pos_y);
```

Description

Loads the digital invalidation string, which may be printed onto the front and/or rear side of the document. It is possible to specify a starting number that will be automatically incremented or decremented for each new document. **At the moment avalible only for LS150 model**

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Side

Specifies which of the document's printed sides.
SIDE_FRONT_IMAGE = Front side image
SIDE_BACK_IMAGE = Rear side image
SIDE_ALL_IMAGE = Both Front and Rear side images

String

Invalidation string having the same syntax of the C-language PRINTF function, if in the string it is not specified the %d or %ld number, the parameter *StartNumber* and *Step* are meaningless.

Length

Length of the string given in the *String* parameter.

StartNumber

Starting number used in the invalidation print numeration.

Step

When greater than zero it sets the increment step applied to *StartNumber* .
When less than zero it sets the decrement step applied to *StartNumber* .

fontName

Name of the window font selected.

fontSize

Size of the character font.

bold

Set character Bold stile.
TRUE = Stile selected.
FALSE = Stile unselected

italic

Set character Italic stile.
TRUE = Stile selected.
FALSE = Stile unselected

underline

Set character Underline stile.
TRUE = Stile selected.
FALSE = Stile unselected

TextColor

Gray tone of the character printed, the range value is from 0 (black) to 255 (white).

UnitMeasure

Specify whether the *pos_x,* and *pos_y* measures are expressed in millimeters or in inches. The possible values are either **UNIT_MM** or **UNIT_INCH**.

pos_x

Specify the x co-ordinate from left margin of the document. The value must be consistent with the *UnitMeasure*.

pos_y

Specify the y co-ordinate from top margin of the document. The value must be consistent with the *UnitMeasure*.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_OPEN_NOT_DONE
LS_SYSTEM_ERROR
LS_INVALID_SIDE
LS_INVALID_UNIT

Comments

11.Security functions

These functions are made available in order to implement data integrity and be sure that the image returned by the scanner has not been altered after capture.

This is achieved with encrypted HASH codes which are inserted inside the image.
The image in any case can be visible with normal imaging tools.

At the moment the signature is done only in JPEG and TIFF Gr.4 images.

The integrity can be checked by the function *CTSCheckSignature()*.

11.1. LSSetSignatureKey

#include "LSApi.h"

```
Result API LSSetSignatureKey (  short           hConnect,
                                HWND            hWnd,
                                unsigned char   *Key,
                                short           lenKey,
                                BOOL            fUseSerialID);
```

Description

Set a User Key to protect the images returned from the unit from alterations.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

Key

A sequence of hexadecimal bytes.
If the first 3 bytes are set to zero, the key and the check are disabled.
Max Key length is **LEN_SIGNATURE_KEY.**

lenKey

Length in bit of the Key, value accepted :
KEY_LENGHT_128
KEY_LENGHT_256
KEY_LENGHT_512

fUseSerialID

Use the internal serial ID to digitally sign the image.
TRUE – Use the Serial ID.
FALSE – Not use the Serial ID.

Return Value

LS_OKAY
LS_SYSTEM_ERROR
LS_USB_ERROR
LS_PERIPHERAL_NOT_FOUND
LS_INVALID_KEY_LENGTH

Comments

11.2. CTSCheckImageSignature

#include "LsApi.h"

```
Result API CTSCheckImageSignature(   unsigned char    *ImageFilename,
                                     unsigned char    *Key,
                                     short            lenKey,
                                     unsigned char    LsSerialID);
```

Description

Check the images integrity from alterations.

Parameters

ImageFilename

Pointer at the image path filename to be checked.

Key

A sequence of hexadecimal bytes. This is the same *Key* used to sign the image.

lenKey

Length in bit of the Key, value accepted :
KEY_LENGHT_128
KEY_LENGHT_256
KEY_LENGHT_512

LsSerialID

Pointer to the internal serial ID of the unit that have filmed the image or NULL if the flag *fUseSerialID* was set to FALSE when function *LSSetSignatureKey()* was called.

Return Value

LS_OKAY
LS_INVALID_KEY_LENGTH
LS_IMAGE_CORRUPTED
LS_IMAGE_NOT_SIGNED

Comments

11.3. CTSCheckSignature

#include "LsApi.h"

```
Result API CTSCheckSignature(  unsigned char     *pImage,
                               long              lenImage,
                               unsigned char     *Key,
                               short             lenKey,
                               unsigned char     LsSerialID);
```

Description

Check the images integrity from alterations.

Parameters

pImage

Pointer at the image to be checked.

lenImage

Dimension in bytes of the image.

Key

A sequence of hexadecimal bytes. This is the same *Key* used to sign the image.

lenKey

Length in bit of the Key, value accepted :
KEY_LENGHT_128
KEY_LENGHT_256
KEY_LENGHT_512

LsSerialID

Pointer to the internal serial ID of the unit that have filmed the image or NULL if the flag *fUseSerialID* was set to FALSE when function *LSSetSignatureKey()* was called.

Return Value

LS_OKAY
LS_INVALID_KEY_LENGTH
LS_IMAGE_CORRUPTED
LS_IMAGE_NOT_SIGNED

Comments

11.4. LSWriteCertificate

```
#include "LsApi.h"

Result API LSWriteCertificate(  short           hConnect,
                                HWND            hWnd,
                                unsigned char   *pCertificate,
                                long            nrBytesCert);
```

Description

Store a certificate p12 in the unit flash.

For now available only for the LS40 and LS100 model.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

pCertificate

Pointer at the certificate.

nrBytesCert

Dimension in bytes of the certificate.

Return Value

LS_OKAY
LS_INVALID_PARAMETER

Comments

11.5. LSReadCertificate

#include "LsApi.h"

```
Result API LSReadCertificate(   short             hConnect,
                                HWND              hWnd,
                                unsigned char     *pCertificate,
                                long              nrBytesCert);
```

Description

Get a certificate p12 stored in the unit flash.
For now available only for the LS40 and LS100 model.

Parameters

hConnect

Handle returned by LSConnect

hWnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

pCertificate

Pointer at the buffer where will be returned the certificate.

nrBytesCert

Dimension in bytes of the certificate.

Return Value

LS_OKAY
LS_INVALID_PARAMETER

Comments

12. Debug and Download functions

The functions described in this section have the purpose of helping the application developer in the debugging stages of application development. They should be used in the development phase only and afterwards removed from the production application code.

The standard DLL delivered in the base package can trace into a file if a folder named **CtsTrace** is created. This folder must be created in the same folder as where the LsApi and the program are, and must have write rights for the user.

LsApi, when loaded, write a file when the folder is present named LsApiTrace.txt

The log files reach a maximum 4Mb bytes of size. When this happens the current file is saved with old extension and a new file is created.

Thence you must have 32M bytes of free disk space.

Latest LsApi versions (from 1.1.0.4 on) are also able to create tracefiles per transaction if the folder **CtsTrace_batch** is created.
Files named LsApiTrace_xxxxxxxxxx.txt (where xxxxxxxxxx is a timing reference) will be created.
ATTENTION: in this mode you must make sure to delete the files or remove the CtsTrace_batch folder otherwise your hard disk will become full.

Trace files include useful information during application debug phase .
This file is usually requested by CTS technical support group should an issue occur during the application development.

It is possible to create a tracefile in a path which is defined by the user in LsApi.ini file in Logging section. This logs modality work as the batch trace.

12.1. LSViewOCRRectangle

#include "LSApi.h"

```
Result API LSViewOCRRectangle(   HWND        hwnd,
                                  BOOL        fView );
```

Description

This function causes the drawing of a red rectangle around the document area that will be OCR decoded with the function *LSCodelineReadFromBitmap()*.

Parameters

hwnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

fView

TRUE = enable the drawing of the rectangle.
FLASE = disable the drawing of the rectangle.

Return Value

LS_OKAY

Comments

12.2. LSDownloadFirmware

#include "LSApi.h"

```
Result API LSDownloadFirmware(     short      hConnect,
                                   HWND       hwnd,
                                   char       *FileFw,
                                   int        (*userfunc1)(char *Item) );
```

Description

The function send the new firmware at the peripheral, one item for time.

Parameters

hConnect

Handle returned by LSConnect

Hwnd

Handle of the application windows which will receive the notification messages **(Reserved for future use)**.

FileFw

Full pathname of the file that contains the firmware to download.

*(*userfunc1)(char *Item)*

Address of an application defined call-back function that will get in *Item* the same string of firmware code sent to the device. This mechanism is provided for download monitoring purpose from the application.
This parameter can be also set to NULL.

Return Value

LS_OKAY
LS_COMMAND_IN_EXECUTION_YET
LS_COMMAND_SEQUENCE_ERROR
LS_MISSING_FILENAME
LS_OPEN_NOT_DONE
LS_OPEN_FILE_ERROR_OR_NOT_FOUND

Comments

13. Application Program Guideline

This is a brief description on how to use LS functions to develop an application program.
The logical flow of the commands should be structured as in the following three examples :

1. Manual document handling (LS40, LS100, LS150, LS5xx) :

```
LSConnect                /* Logical connection to the device */
LSIdentify               /* Device identification */
. . . . .
LSReadBadge              /* Badge read  */
. . . . .
LSLoadString             /* Load invalidation string */
LSConfigDoubleLeafingAndDocLength /* Set double leafing parameters */
LSDocHandle              /* Execute physical document handling */
LSReadCodeline           /* Read document codeline data */
LSReadImage              /* Read document image */
LSFreeImage              /* Free memory buffers */
LSDisconnect             /* Disconnect from device */
```

2. Advanced document handling (LS40, LS100, LS150, LS5xx):

```
LSConnect                /* Logical connection to the device*/
LSIdentify               /* Device identification */
. . . . .
LSReadBadge              /* Badge read  */
. . . . .
LSLoadString             /* Load invalidation string */
LSConfigDoubleLeafingAndDocLength /* Set double leafing parameters */
LSAutoDocHandle          /* Execute automatic document handling */
LSGetDocData             /* Retrieve the documents information */
LSDisconnect             /* Disconnect from device */
```

3. Manual document handling for retained document (LS40, LS100) :

```
LSConnect                /* Logical connection to the devicel */
LSIdentify               /* Device identification */
. . . . .
LSReadBadge              /* Badge read  */
. . . . .
LSLoadString             /* Load invalidation string */
LSConfigDoubleLeafingAndDocLength /* Set double leafing parameters */
LSDocHandle              /* Execute physical document handling with retain parameter */
LSReadCodeline           /* Read document codeline data if needed */
LSReadImage              /* Read document image if needed */
LSFreeImage              /* Free memory buffers if needed */
LSDocHandle              /* Execute physical document handling with sorter parameter */
LSReadImage              /* Read document image if needed */
LSFreeImage              /* Free memory buffers if needed */
LSDisconnect             /* Disconnect from device */
```

4. Advanced document handling (LS800):

```
LSConnect                /* Logical connection to the device*/
LSIdentify               /* Device identification */
```

```
. . . . .
LSConfigDoubleLeafingAndDocLength /* Set double leafing parameters */
LS800AutoDocHandle  /* Execute automatic document handling */

Decide ENDORSEMENT and pocket on callback function

LSGetDocData            /* Retrieve the documents information */
LSDisconnect            /* Disconnect from device */
```

Note :

- **LSIdentify** allows the application to get specific information about device configuration and capabilities.
- Document handling functions ***must be used*** only inside a document handling session.
- To release the image memory allocated by the service use **LSFreeImage.**
- **LSConvertImageToBW** can be used only for *manual document handling.*
- To use **LSSaveJpeg, LSDisplayImage, LSUpdateImage** it is necessary to install the ***complete software distribution kit***.

ATTENTION :

When the application calls LSConnect, the LS service tries to load the additional module IMG_UTIL.DLL. If this module is not present the service disables the use of the additional functions. If IMG_UTIL.DLL is found then the following additional DLL's are also required :

LFBMP13N.DLL
LFCMP13N.DLL
LFFAX13N.DLL
LTDIS13N.DLL
LTDLG13N.DLL
LTFIL13N.DLL
LTIMG13N.DLL
LTKRN13N.DLL
LTEFX13N.DLL
otherwise the system will display a "Unable to locate DLL" message box.

14. Sample code

14.1. Sample code in C language for the LS515 extracted from LSW5 demo code.

There are two different functions, the first one is for the manual handling of documents, the second one for the automatic one.

14.1.1. DoSingleDocHandle()

```
int DoSingleDocHandle(HWND hWnd)
{
    int     Reply, ReplyDH;
    short   X, Y, Size;
    char    dirBase[_MAX_DIR];
    char    Filename[_MAX_FNAME];
    char  BufCodelineHW[CODE_LINE_LENGTH];
    char  BufBarcode[CODE_LINE_LENGTH];
    char  BufCodelineSW[CODE_LINE_LENGTH];
    short llCodeline;
    short len_barcode;
    char    CodelineType;
    short   Sorter;
    short   StopAfterCodeline;
    READOPTIONS ro;
    unsigned char SenseKey;
    unsigned char StatusByte[4];
    char  SideToFilm;


    NrCheque ++;


    //-----------LoadString--------------------------------------
    if( (stParDocHandle.Validate == PRINT_VALIDATE) && (strlen(stParDocHandle.szValidateText)) )
    {
        if( strstr(stParDocHandle.szValidateText, "%d") )
            Reply = LSLoadStringWithCounter(hLS,
                                hWnd,
                                (char)(stParDocHandle.PrintBold ? FORMAT_BOLD :
FORMAT_NORMAL),
                                stParDocHandle.szValidateText,
                                8,
                                3);
        else
            Reply = LSLoadString(hLS,
                                hWnd,
                                 (char)(stParDocHandle.PrintBold ? FORMAT_BOLD :
FORMAT_NORMAL),
                                 (short)strlen(stParDocHandle.szValidateText),
                                 stParDocHandle.szValidateText);
        if (Reply != LS_OKAY)
        {
            if (CheckReply(hWnd, Reply, "LSLoadString"))
            {
```

```
                        return Reply;
                    }
            }
    }


    // Send the command of draw red rectangle
    LSViewOCRRectangle(hWnd, fViewOCRRectangle);


    // Send the command for handle the image
    LSEnableImageCorrection(hLS, hWnd, fImageCorrection);


    if( TypeLS == TYPE_LS510S || TypeLS == TYPE_LS510D ||
            TypeLS == TYPE_LS515S || TypeLS == TYPE_LS515D )
    {
            //----------- Set sorter criteria, if required
            if( stParDocHandle.Sorter == SORTER_AUTOMATIC )
            {
                    Reply = LSSetSorterCriteria(hLS,
                                                hWnd,
                                                stParDocHandle.DataSorter,
                                                MAX_CRITERIA);
                    if( Reply != LS_OKAY )
                    {
                            if( CheckReply(hWnd, Reply, "LSSetSorterCriteria") )
                            {
                                    return 0;
                            }
                    }
            }

            //----------- Set double leafing sensibility -----------
            Reply = LSDoubleLeafingSensibility(hLS, hWnd, 0, stParDocHandle.DoubleLeafingLevel);
            if (Reply != LS_OKAY)
            {
                    if (CheckReply(hWnd, Reply, "LSDuobleLeafingSensibility"))
                    {
                            return 0;
                    }
            }

            //----------- Set Block Document if double leafing -----------
            Reply = LSConfigDoubleLeafing(hLS, hWnd,
                                        (short)(stParDocHandle.DoubleLeafingBlock ?
                            DOUBLE_LEAFING_ERROR : DOUBLE_LEAFING_WARNING));
            if (Reply != LS_OKAY)
            {
                    if( CheckReply(hWnd, Reply, "LSConfigDoubleLeafing"))
                    {
                            return 0;
                    }
            }
    }

    // set fixed params for the LS500_DocHandle
    sprintf(dirBase, "%s%s", PathAppl, SAVE_DIRECTORY_IMAGE);
    strcpy(Filename, NAME_IMAGE);
    if( (stParDocHandle.CodelineType == READ_CODELINE_SW_OCRA) ||
        (stParDocHandle.CodelineType == READ_CODELINE_SW_OCRB_NUM) )
    {
        CodelineType = NO_READ_CODELINE;

        X    = 16;
        Y    = 16;
        Size = -1;

        if( stParDocHandle.ScanMode == SCAN_MODE_16GR200 )
        {
            X *= 2;
            Y *= 2;
        }
    }
    else
        CodelineType = stParDocHandle.CodelineType;


    // in case of CheckCodeline I force the film of the image
    if( (stParDocHandle.DoCheckCodeline) &&
        (stParDocHandle.Side == SIDE_NONE_IMAGE) )
        SideToFilm = SIDE_FRONT_IMAGE;
    else
        SideToFilm = stParDocHandle.Side;


    //-----------DocHandle----------------------------------
    ReplyDH = LSDocHandle(hLS,
                        hWnd,
                        stParDocHandle.Stamp,
                        stParDocHandle.Validate,
                        CodelineType,
                        SideToFilm,
                        stParDocHandle.ScanMode,
                        (short)(stParDocHandle.LinearEntry == TRUE ? PATH_FEED :
AUTO_FEED),
                        stParDocHandle.Sorter,
                        stParDocHandle.WaitTimeout,
                        stParDocHandle.BeepOnError,
                        NULL,
                        0,
                        0);

    if( ReplyDH != LS_OKAY && ReplyDH != LS_KEEP_DOC_ON_CODELINE_ERROR )
    {
        if( ReplyDH != LS_DOUBLE_LEAFING_WARNING )
            if( CheckReply(hWnd, ReplyDH, "LSDocHandle"))
            {
                return ReplyDH;
            }
    }


    //
    // Read image and codeline
    //

    // free  bitmaps
```

```
if( BufBackNettoImage )
{
            GlobalFree( BufBackNettoImage );
            BufBackNettoImage = 0;
}
if( BufBackImage )
{
            GlobalFree( BufBackImage );
            BufBackImage = 0;
}
if( BufFrontNettoImage )
{
            GlobalFree( BufFrontNettoImage );
            BufFrontNettoImage = 0;
}
if( BufFrontImage )
{
            GlobalFree( BufFrontImage );
            BufFrontImage = 0;
}

memset((char *)BufCodelineHW,  0, sizeof(BufCodelineHW));
memset((char *)BufBarcode,                      0, sizeof(BufBarcode));
memset((char *)BufCodelineSW,  0, sizeof(BufCodelineSW));

llCodeline = CODE_LINE_LENGTH;
len_barcode = CODE_LINE_LENGTH;

//------------Read Codeline hardware--------------------------
if( (!stParDocHandle.LinearEntry) &&
            (stParDocHandle.CodelineType == READ_CODELINE_MICR ||
             stParDocHandle.CodelineType == READ_BARCODE_HW ||
             stParDocHandle.CodelineType == READ_MICR_AND_BARCODE_HW) )
{
            Reply = LSReadCodeline(hLS,
                                                            hWnd,
                                                            BufCodelineHW,
                                                            &llCodeline,
                                                            BufBarcode,
                                                            &len_barcode,
                                                            NULL,
                                                            NULL);

            if( Reply != LS_OKAY)
            {
                        if( CheckReply(hWnd, Reply, "LSReadCodeline"))
                        {
                                    return Reply;
                        }
            }
}

StopAfterCodeline = FALSE;

// Read images only if DocHandle is OK
if( ((ReplyDH == LS_OKAY) || (ReplyDH == LS_DOUBLE_LEAFING_WARNING)) &&
            (Reply == LS_OKAY) )
{
            //-----------ReadImage------------------------------------
```

```
if( SideToFilm != SIDE_NONE_IMAGE )
{
    Reply = LSReadImage(hLS,
            hWnd,
            stParDocHandle.ClearBlack,
            SideToFilm,
            stParDocHandle.ReadMode,
            0,
            &BufFrontImage,
            &BufBackImage,
            &BufFrontNettoImage,
            &BufBackNettoImage);

    if (Reply != LS_OKAY)
    {
        if((Reply == LS_IMAGE_NOT_PRESENT ) &&
            (stParDocHandle.Sorter == SORTER_AUTOMATIC))
        {

            StopAfterCodeline = TRUE;
        }
        else
        {
            if( CheckReply(hWnd, Reply, "LSReadImage"))
            {
                return Reply;
            }
        }
    }

    if( stParDocHandle.TypeOfDecod & DECODE_BARCODE )
    {
        Reply = LSReadBarcodeFromBitmap(hWnd,
                        BufFrontImage,
                        stParDocHandle.Barcodetype,
                        (int)stParDocHandle.Barcode_Sw_x,
                        (int)stParDocHandle.Barcode_Sw_y,
                        (int)stParDocHandle.Barcode_Sw_w,
                        (int)stParDocHandle.Barcode_Sw_h,
                        BufBarcode,
                        (UINT *)&len_barcode);
        if (Reply != LS_OKAY)
        {
            CheckReply(hWnd, Reply, "LS500_ReadBarcodeFromBitmap");
        }
    }
}

if( (StopAfterCodeline == FALSE) &&
    ((stParDocHandle.Side == SIDE_ALL_IMAGE) ||
    (stParDocHandle.Side == SIDE_FRONT_IMAGE)) )
{
    if( ((stParDocHandle.CodelineType == READ_CODELINE_SW_OCRA) ||
         (stParDocHandle.CodelineType == READ_CODELINE_SW_OCRB_NUM))
         && (hOCRLibrary) )
    {
        ro.PutBlanks = TRUE;
        ro.TypeRead = 'N';
```

```
                Reply = LSReadCodelineFromBitmap(hWnd,
                                BufFrontImage,
                                (char *)&stParDocHandle.CodelineType,
                                X,
                                Y,
                                Size,
                                MAX_PIXEL_HEIGHT,
                                &ro,
                                BufCodelineSW,
                                (UINT *)&llCodeline);
                if (Reply != LS_OKAY)
                {
                    if( CheckReply(hWnd, Reply, "LSReadCodelineFromBitmapEx"))
                    {
                        return Reply;
                    }
                }
            }
        }


        if(!StopAfterCodeline)
        {
            if( !stParDocHandle.ViewOnlyLastImage )
            {
                if( (fOptionViewImage == TRUE) && (!stParDocHandle.DoCheckCodeline) )
                {
                    ShowCodelineAndImage(ReplyDH,
                                NrCheque,
                                stParDocHandle.Side,
                                (unsigned char *)BufFrontImage,
                                (unsigned char *)BufBackImage,
                                (unsigned char *)BufFrontNettoImage,
                                (unsigned char *)BufBackNettoImage,
                                BufCodelineSW,
                                BufCodelineHW,
                                BufBarcode,
                                "");
                } // if fOptionViewImage
            } // if( !stParDocHandle.ViewOnlyLastImage )
        }


        if(stParDocHandle.SaveImage == IMAGE_SAVE_BOTH)
        {
            // richiesto salvataggio file
            switch(stParDocHandle.SaveFormat)
            {
            case SAVE_JPEG:
                if( BufBackImage )
                {
                    // build filename
                    sprintf(SaveFile, "%s\\%s%dBB.jpg", SAVE_DIRECTORY_IMAGE,
NAME_IMAGE, NrFileJPEG);
                    Reply = LSSaveJPEG(hWnd, BufBackImage, stParDocHandle.Qual,
SaveFile );
                }

                if( BufFrontImage )
                {
                    // build filename
                    sprintf(SaveFile, "%s\\%s%dFF.jpg", SAVE_DIRECTORY_IMAGE,
NAME_IMAGE, NrFileJPEG);
                    Reply = LSSaveJPEG(hWnd, BufFrontImage, stParDocHandle.Qual,
SaveFile );
                }

                NrFileJPEG ++;
                break;

            case SAVE_BMP:
                if( BufBackImage )
                {
                    // build filename
                    sprintf(SaveFile, "%s\\%s%dBB.bmp", SAVE_DIRECTORY_IMAGE,
NAME_IMAGE, NrFileBMP);
                    Reply = LSSaveDIB(hWnd, BufBackImage, SaveFile );
                }

                if( BufFrontImage )
                {
                    // build filename
                    sprintf(SaveFile, "%s\\%s%dFF.bmp", SAVE_DIRECTORY_IMAGE,
NAME_IMAGE, NrFileBMP);
                    Reply = LSSaveDIB(hWnd, BufFrontImage, SaveFile );
                }

                NrFileBMP ++;
                break;

            case FILE_TIF:
            case FILE_CCITT:
            case FILE_CCITT_GROUP3_1DIM:
            case FILE_CCITT_GROUP3_2DIM:
            case FILE_CCITT_GROUP4:

                if( BufBackImage )
                {
                    // build filename
                    sprintf(SaveFile, "%s\\%s%dBB.tiff", SAVE_DIRECTORY_IMAGE,
NAME_IMAGE, NrFileTIFF);
                    Reply = LSSaveTIFF(hWnd, BufBackImage, SaveFile,
stParDocHandle.SaveFormat,SAVE_OVERWRITE,1);
                }

                if( BufFrontImage )
                {
                    // build filename
                    sprintf(SaveFile, "%s\\%s%dFF.tiff", SAVE_DIRECTORY_IMAGE,
NAME_IMAGE, NrFileTIFF);
                    Reply = LSSaveTIFF(hWnd, BufFrontImage, SaveFile,
stParDocHandle.SaveFormat,SAVE_OVERWRITE, 1);
                }

                NrFileTIFF ++;
                break;
            }
        }
        //save images
        if( Save_BufBackNettoImage )
            GlobalFree( Save_BufBackNettoImage );
        Save_BufBackNettoImage = BufBackNettoImage;
        BufBackNettoImage = 0;

        if( Save_BufBackImage )
            GlobalFree( Save_BufBackImage );
        Save_BufBackImage = BufBackImage;
        BufBackImage = 0;

        if( Save_BufFrontNettoImage )
            GlobalFree( Save_BufFrontNettoImage );
        Save_BufFrontNettoImage = BufFrontNettoImage;
        BufFrontNettoImage = 0;

        if( Save_BufFrontImage )
            GlobalFree( Save_BufFrontImage );
        Save_BufFrontImage = BufFrontImage;
        BufFrontImage = 0;

        //save codelines
        memcpy(Save_BufCodelineSW, BufCodelineSW, sizeof(Save_BufCodelineSW));
        memcpy(Save_BufCodelineHW, BufCodelineHW, sizeof(Save_BufCodelineHW));
        memcpy(Save_BufBarcode, BufBarcode, sizeof(Save_BufBarcode));
    }


    memset(CodelineRead, 0, sizeof(CodelineRead));
    if( BufCodelineHW[0] )
        strcpy(CodelineRead, BufCodelineHW);
    else if( BufBarcode[0] )
        strcpy(CodelineRead, BufBarcode);
    else if( BufCodelineSW[0] )
        strcpy(CodelineRead, BufCodelineSW);


    if( (stParDocHandle.DoCheckCodeline ) &&
            (stParDocHandle.CodelineType != NO_READ_CODELINE) )
        {
            CheckCodeline(hWnd, CodelineRead, TRUE, (char *)BufFrontImage);
        }

    if( (stParDocHandle.Sorter == HOLD_DOCUMENT) ||
        (fDocRetained && (ReplyDH == LS_KEEP_DOC_ON_CODELINE_ERROR)) ||
        (StopAfterCodeline) )
    {
        Sorter = (short)DialogBox(hInst, MAKEINTRESOURCE(IDD_SORTER), hWnd, SorterDlgProc);

        //-----------LoadString--------------------------------------
        if( (stParDocHandle.SorterValidate == PRINT_VALIDATE) &&
(strlen(stParDocHandle.SorterszValidateText)) )
        {
            Reply = LSLoadString(hLS,
                        hWnd,
                        (char)(stParDocHandle.SorterPrintBold ? FORMAT_BOLD :
FORMAT_NORMAL),
                        (short)strlen(stParDocHandle.SorterszValidateText),
                        stParDocHandle.SorterszValidateText);
            if( Reply != LS_OKAY) //&& (Reply != LS_KEEP_DOC_ON_CODELINE_ERROR) )
            {
                if (CheckReply(hWnd, Reply, "LSLoadString"))
                {
                    return Reply;
                }
            }
        }

        //-----------DocHandle----------------------------------
        Reply = LSDocHandle(hLS,
                            hWnd,
                            stParDocHandle.SorterStamp,
                            stParDocHandle.SorterValidate,
                            NO_READ_CODELINE,
                            stParDocHandle.SorterSide,
                            stParDocHandle.ScanMode,
                            PATH_FEED,
                            Sorter,
                            stParDocHandle.WaitTimeout,
                            stParDocHandle.BeepOnError,
                            NULL,
                            0,
                            0);
        if( Reply != LS_OKAY )
        {
            if( CheckReply(hWnd, Reply, "LSDocHandle") )
            {
                return Reply;
            }
        }


        //
        // read image and codeline
        //

        if( BufBackNettoImage )
        {
            GlobalFree( BufBackNettoImage );
            BufBackNettoImage = 0;
        }
        if( BufBackImage )
        {
            GlobalFree( BufBackImage );
            BufBackImage = 0;
        }
        if( BufFrontNettoImage )
        {
            GlobalFree( BufFrontNettoImage );
            BufFrontNettoImage = 0;
        }
        if( BufFrontImage )
        {
            GlobalFree( BufFrontImage );
            BufFrontImage = 0;
        }
```

```
        if( Reply == LS_OKAY )
        {
            //-----------ReadImage------------------------------------
            if( stParDocHandle.SorterSide != SIDE_NONE_IMAGE )
            {
                Reply = LSReadImage(hLS,
                                    hWnd,
                                    stParDocHandle.ClearBlack,
                                    stParDocHandle.SorterSide,
                                    stParDocHandle.ReadMode,
                                    0,
                                    &BufFrontImage,
                                    &BufBackImage,
                                    &BufFrontNettoImage,
                                    &BufBackNettoImage);

                if (Reply != LS_OKAY)
                {
                    if( CheckReply(hWnd, Reply, "LSReadImage"))
                    {
                        return Reply;
                    }
                }
            }

            if( (stParDocHandle.SorterSide == SIDE_ALL_IMAGE) ||
                (stParDocHandle.SorterSide == SIDE_FRONT_IMAGE) )
            {
                if( ((stParDocHandle.CodelineType == READ_CODELINE_SW_OCRA) ||
                     (stParDocHandle.CodelineType ==
READ_CODELINE_SW_OCRB_NUM))
                     && (hOCRLibrary) )
                {
                    ro.PutBlanks = TRUE;
                    ro.TypeRead = 'N';

                    LSReadCodelineFromBitmap(hWnd,
                                BufFrontImage,
                                (char *)&stParDocHandle.CodelineType,
                                X,
                                Y,
                                Size,
                                MAX_PIXEL_HEIGHT,
                                &ro,
                                BufCodelineSW,
                                (UINT *)&llCodeline);
                    if (Reply != LS_OKAY)
                    {
                        if( CheckReply(hWnd, Reply, "LSReadCodelineFromBitmapEx"))
                        {
                            return Reply;
                        }
                    }
                }
            }

            if( fOptionViewImage == TRUE )
            {
                ShowCodelineAndImage(Reply,
                            NrCheque,
                            stParDocHandle.SorterSide,
                            (unsigned char *)BufFrontImage,
                            (unsigned char *)BufBackImage,
                            (unsigned char *)BufFrontNettoImage,
                            (unsigned char *)BufBackNettoImage,
                            BufCodelineSW,
                            BufCodelineHW,
                            BufBarcode,
                            "");

            } // if fOptionViewImage

            if(stParDocHandle.SaveImage == IMAGE_SAVE_BOTH)
            {
                // richiesto salvataggio file
                switch(stParDocHandle.SaveFormat)
                {
                case SAVE_JPEG:
                    if( BufBackImage )
                    {
                        // build filename
                        sprintf(SaveFile, "%s\\%s%dBB.jpg",
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileJPEG);
                        Reply = LSSaveJPEG(hWnd, BufBackImage,
stParDocHandle.Qual, SaveFile );
                    }

                    if( BufFrontImage )
                    {
                        // build filename
                        sprintf(SaveFile, "%s\\%s%dFF.jpg",
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileJPEG);
                        Reply = LSSaveJPEG(hWnd, BufFrontImage,
stParDocHandle.Qual, SaveFile );
                    }

                    NrFileJPEG ++;
                    break;

                case SAVE_BMP:
                    if( BufBackImage )
                    {
                        // build filename
                        sprintf(SaveFile, "%s\\%s%dBB.bmp",
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileBMP);
                        Reply = LSSaveDIB(hWnd, BufBackImage, SaveFile );
                    }

                    if( BufFrontImage )
                    {
                        // Costruisco nome file completo
                        sprintf(SaveFile, "%s\\%s%dFF.bmp",
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileBMP);
                        Reply = LSSaveDIB(hWnd, BufFrontImage, SaveFile );
                    }

                    NrFileBMP ++;
                    break;

                case FILE_TIF:
                case FILE_CCITT:
                case FILE_CCITT_GROUP3_1DIM:
                case FILE_CCITT_GROUP3_2DIM:
                case FILE_CCITT_GROUP4:

                    if( BufBackImage )
                    {
                        // build filename
                        sprintf(SaveFile, "%s\\%s%dBB.tiff",
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileTIFF);
                        Reply = LSSaveTIFF(hWnd, BufBackImage, SaveFile,
stParDocHandle.SaveFormat,SAVE_OVERWRITE,1);
                    }

                    if( BufFrontImage )
                    {
                        // build filename
                        sprintf(SaveFile, "%s\\%s%dFF.tiff",
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileTIFF);
                        Reply = LSSaveTIFF(hWnd, BufFrontImage, SaveFile,
stParDocHandle.SaveFormat,SAVE_OVERWRITE, 1);
                    }

                    NrFileTIFF ++;
                    break;
                }
            }
        } // if( Reply == LS_OKAY )
    }

    // more docs?
    if( Reply == LS_OKAY )
    {
        if( LSPeripheralStatus(hLS, hWnd, &SenseKey, StatusByte) == LS_OKAY )
        {
            if( !(StatusByte[0] & FEEDER_EMPTY) )
                Reply = LS_FEEDER_EMPTY;
        }
    }

    return Reply;
}  // End DoSingleDocHandle
```

14.1.2. DoAutoDocHandle()

```
int DoAutoDocHandle(HWND hWnd)
{
    int             Reply;
    char    dirBase[_MAX_DIR];
    char    Filename[_MAX_FNAME];

    char    BufFrontFile[_MAX_PATH];
    char    BufFrontNettoFile[_MAX_PATH];
    char    BufBackFile[_MAX_PATH];
```

```
        char     BufBackNettoFile[_MAX_PATH];

        char     BufCodelineSW[CODE_LINE_LENGTH];
        char     BufCodelineHW[CODE_LINE_LENGTH];
        char     BufBarcode[CODE_LINE_LENGTH];
        READOPTIONS ro;
        short len_codeline, len_barcode = CODE_LINE_LENGTH;
        char     WaitCom;
        MSG              msg;
        short    NumDocRemain;
        short    NrDocToProcess;
        int              ReplyFunz = 0;
        short    CodelineType;
        char     SideToFilm;
        char   TypeOfDecod;
        char     nOutGrade;

        FILE     *fhCodeline = NULL;
        float C_x,C_y,C_w,C_h;


        if( stParDocHandle.NumDoc )
                 NrDocToProcess = stParDocHandle.NumDoc;
        else
                 NrDocToProcess = -1;

        if(( stParDocHandle.TypeOfDecod & DECODE_OCR )&& (stParDocHandle.Side ==
SIDE_NONE_IMAGE ))
                 SideToFilm =  SIDE_FRONT_IMAGE;

        //-----------LoadString--------------------------------------
        if( (stParDocHandle.Validate == PRINT_VALIDATE) && (strlen(stParDocHandle.szValidateText)))
        {
                 if( strstr(stParDocHandle.szValidateText, "%d") )
                          Reply = LSLoadStringWithCounter(hLS,
                                                                                          hWnd,
                                                                                          (char)
(stParDocHandle.PrintBold ? FORMAT_BOLD : FORMAT_NORMAL),

        stParDocHandle.szValidateText,
                                                                                          8,
                                                                                          3);
                 else
                          Reply = LSLoadString(hLS,
                                                     hWnd,
                                                    (char)(stParDocHandle.PrintBold ? FORMAT_BOLD :
FORMAT_NORMAL),
                                                     (short)strlen(stParDocHandle.szValidateText),
                                                    stParDocHandle.szValidateText);
                 if (Reply != LS_OKAY)
                 {
                          if (CheckReply(hWnd, Reply, "LSLoadString"))
                          {
                                   return 0;
                          }
                 }
        }
```

```
    // Send the command of draw red rectangle
    LSViewOCRRectangle(hWnd, fViewOCRRectangle);

    // Send the command for handle the image
    LSEnableImageCorrection(hLS, hWnd, fImageCorrection);


    if( TypeLS == TYPE_LS505 || TypeLS == TYPE_LS510S || TypeLS == TYPE_LS510D ||
        TypeLS == TYPE_LS515S || TypeLS == TYPE_LS515D )
    {
        //----------- Change stamp position -----------
        Reply = LSChangeStampPosition(hLS, hWnd, stParDocHandle.StampPosition, 0);
    }

    if( TypeLS == TYPE_LS510S || TypeLS == TYPE_LS510D ||
        TypeLS == TYPE_LS515S || TypeLS == TYPE_LS515D )
    {
        Reply = LSDoubleLeafingSensibility(hLS, hWnd, 0, stParDocHandle.DoubleLeafingLevel);
        if (Reply != LS_OKAY)
        {
            if (CheckReply(hWnd, Reply, "LSDoubleLeafingSensibility"))
            {
                return 0;
            }
        }

        //----------- Set Block Document if double leafing -----------
        Reply = LSConfigDoubleLeafing(hLS, hWnd, (short)(stParDocHandle.DoubleLeafingBlock ?
DOUBLE_LEAFING_ERROR : DOUBLE_LEAFING_WARNING));
        if (Reply != LS_OKAY)
        {
            if( CheckReply(hWnd, Reply, "LSConfigDoubleLeafing"))
            {
                return 0;
            }
        }

        //----------- Set sorter criteria if needed
        if( stParDocHandle.Sorter == SORTER_AUTOMATIC )
        {
            Reply = LSSetSorterCriteria(hLS,
                        hWnd,
                        stParDocHandle.DataSorter,
                        MAX_CRITERIA);
            if( Reply != LS_OKAY )
            {
                if( CheckReply(hWnd, Reply, "LSSetSorterCriteria") )
                {
                    return 0;
                }
            }
        }
    }


    sprintf(dirBase, "%s%s", PathAppl, SAVE_DIRECTORY_IMAGE);
    strcpy(Filename, NAME_IMAGE);

    TypeOfDecod = stParDocHandle.TypeOfDecod;
```

```
	CodelineType = NO_READ_CODELINE;
	if( stParDocHandle.TypeOfDecod & DECODE_MICR )
	{
		CodelineType = READ_CODELINE_MICR;
		TypeOfDecod -= DECODE_MICR;
	}
	else if( stParDocHandle.TypeOfDecod & DECODE_OCR )
	{
		CodelineType = NO_READ_CODELINE;
		C_x = stParDocHandle.Codeline_Sw_x;
		C_y = stParDocHandle.Codeline_Sw_y;
		C_w = stParDocHandle.Codeline_Sw_w;
		C_h = stParDocHandle.Codeline_Sw_h;

		TypeOfDecod -= DECODE_OCR;
	}
	else if( stParDocHandle.TypeOfDecod & DECODE_BARCODE )
	{
		CodelineType = NO_READ_CODELINE;
		C_x = stParDocHandle.Barcode_Sw_x;
		C_y = stParDocHandle.Barcode_Sw_y;
		C_w = stParDocHandle.Barcode_Sw_w;
		C_h = stParDocHandle.Barcode_Sw_h;

		TypeOfDecod -= DECODE_BARCODE;
	}
	else if( stParDocHandle.TypeOfDecod & DECODE_PDF417 )
	{
		CodelineType = NO_READ_CODELINE;
		TypeOfDecod -= DECODE_PDF417;
	}

	if( fStopLoopOnError )
		BuilderParam.StopLoopOnErrorInCodeline = TRUE;
	else
		BuilderParam.StopLoopOnErrorInCodeline = FALSE;

	if( (stParDocHandle.DoCheckCodeline) &&
		(stParDocHandle.Side == SIDE_NONE_IMAGE) )
		SideToFilm = SIDE_FRONT_IMAGE;
	else
		SideToFilm = stParDocHandle.Side;


	//-----------AutoDocHandle----------------------------------
	BuilderParam.Beep = TRUE;	//FALSE;

	LSBuilderSetting(hWnd, (void *)&BuilderParam);


	Reply = LSAutoDocHandle(hLS,
				hWnd,
				stParDocHandle.Stamp,
				stParDocHandle.Validate,
				CodelineType,
				stParDocHandle.ScanMode,
				AUTO_FEED,	//(short)(stParDocHandle.LinearEntry == TRUE ? SCAN_FEED :
AUTO_FEED),
				stParDocHandle.Sorter,
                                        stParDocHandle.NumDoc,
                                        stParDocHandle.ClearBlack,
                                        SideToFilm,
                                        stParDocHandle.ReadMode,
                                        stParDocHandle.SaveImage,
                                        dirBase,
                                        Filename,
                                        (float)stParDocHandle.Codeline_Sw_x,
                                        (float)stParDocHandle.Codeline_Sw_y,
                                        (float)stParDocHandle.Codeline_Sw_w,
                                        0,
                                        0,
                                        OCR_FRONT_IMAGE,
                                        stParDocHandle.SaveFormat,
                                        stParDocHandle.Qual,
                                        SAVE_OVERWRITE,     // 0,
                                        1,          // 0,
                                        stParDocHandle.WaitTimeout,
                                        stParDocHandle.BeepOnError,
                                        NULL,
                                        NULL,
                                        NULL);

if (Reply != LS_OKAY)
{
        if( (Reply != LS_FEEDER_EMPTY) && (Reply != LS_DOUBLE_LEAFING_WARNING) )
        {
                if (CheckReply(hWnd, Reply, "LSExtAutoDocHandle"))
                {
                        return 0;
                }
        }
        else
                Reply = LS_OKAY;                        }


if( stParDocHandle.SaveCodeline )
{
        strcpy(FullFName, PathAppl);
        strcat(FullFName, FILE_CODELINE);

        if( stParDocHandle.ResetFileCodeline )
                fhCodeline = fopen(FullFName, "w+t");
        else
                fhCodeline = fopen(FullFName, "a+t");
}


if( Save_BufBackNettoImage )
{
        GlobalFree( Save_BufBackNettoImage );
        Save_BufBackNettoImage = 0;
}

if( Save_BufBackImage )
{
        GlobalFree( Save_BufBackImage );
        Save_BufBackImage = 0;
}
```

```
    if( Save_BufFrontNettoImage )
    {
        GlobalFree( Save_BufFrontNettoImage );
        Save_BufFrontNettoImage = 0;
    }

    if( Save_BufFrontImage )
    {
        GlobalFree( Save_BufFrontImage );
        Save_BufFrontImage = 0;
    }

    memset(Save_BufCodelineSW,    0, sizeof(Save_BufCodelineSW));
    memset(Save_BufCodelineHW,    0, sizeof(Save_BufCodelineHW));
    memset(Save_BufBarcode,                0, sizeof(Save_BufBarcode));


    //-----------GetDocData--------------------------------------
  if( (fOptionViewImage == TRUE) && (Reply == LS_OKAY) )
    {
        while( NrCheque != NrDocToProcess )
        {

            WaitCom = WAIT_YES;

            memset(BufFrontFile,       0, sizeof(BufFrontFile));
            memset(BufFrontNettoFile, 0, sizeof(BufFrontNettoFile));
            memset(BufBackFile,                  0, sizeof(BufBackFile));
            memset(BufBackNettoFile,  0, sizeof(BufBackNettoFile));

            // free bitmaps
            if( BufBackNettoImage )
            {
                GlobalFree( BufBackNettoImage );
                BufBackNettoImage = 0;
            }
            if( BufBackImage )
            {
                GlobalFree( BufBackImage );
                BufBackImage = 0;
            }
            if( BufFrontNettoImage )
            {
                GlobalFree( BufFrontNettoImage );
                BufFrontNettoImage = 0;
            }
            if( BufFrontImage )
            {
                GlobalFree( BufFrontImage );
                BufFrontImage = 0;
            }

            memset(BufCodelineSW, 0, sizeof(BufCodelineSW));
            memset(BufCodelineHW, 0, sizeof(BufCodelineHW));
            memset(BufBarcode,                  0, sizeof(BufBarcode));

            Reply = LSGetDocData(hLS,
                                                    hWnd,
                    NULL,
                    BufFrontFile,
                    BufBackFile,
                    BufFrontNettoFile,
                    BufBackNettoFile,
                    &(LPHANDLE)BufFrontImage,
                    &(LPHANDLE)BufBackImage,
                    &(LPHANDLE)BufFrontNettoImage,
                    &(LPHANDLE)BufBackNettoImage,
                    BufCodelineSW,
                    BufCodelineHW,
                    BufBarcode,
                    NULL,
                    &NumDocRemain,
                    0,
                    NULL,
                    NULL);

if( (Reply != LS_OKAY) && (Reply != LS_DOUBLE_LEAFING_WARNING) &&
    (Reply != LS_LOOP_INTERRUPTED) &&
    (Reply != LS_SORTER1_FULL) && (Reply != LS_SORTER2_FULL) &&
    (Reply != LS_SORTERS_BOTH_FULL))
{
    if( CheckReply(hWnd, Reply, "LSGetDocData") )
    {
        break;
    }
}


NrCheque ++;

if( stParDocHandle.TypeOfDecod & DECODE_OCR )
{
    ro.PutBlanks = TRUE;
    ro.TypeRead = 'N';

    Reply = LSCodelineReadFromBitmap(hWnd,
                    (HANDLE)BufFrontImage,
                    &stParDocHandle.CodelineOptType,
                    stParDocHandle.Unit_measure,
                    stParDocHandle.Codeline_Sw_x,
                    stParDocHandle.Codeline_Sw_y,
                    stParDocHandle.Codeline_Sw_w,
                    stParDocHandle.Codeline_Sw_h,
                    &ro,
                    BufCodelineSW,
                    (UINT *)&len_codeline);

    if (Reply != LS_OKAY)
    {
        CheckReply(hWnd, Reply, "LSCodelineReadFromBitmap");
    }
}

if( stParDocHandle.TypeOfDecod & DECODE_BARCODE )
{
    Reply = LSReadBarcodeFromBitmap(hWnd,
                    BufFrontImage,
                    stParDocHandle.Barcodetype,
                    (int)stParDocHandle.Barcode_Sw_x,
                    (int)stParDocHandle.Barcode_Sw_y,
                    (int)stParDocHandle.Barcode_Sw_w,
                    (int)stParDocHandle.Barcode_Sw_h,
                    BufBarcode,
                    (UINT *)&len_barcode);
    if (Reply != LS_OKAY)
    {
        CheckReply(hWnd, Reply, "LS500_ReadBarcodeFromBitmap");
    }
}

if( stParDocHandle.TypeOfDecod & DECODE_PDF417 )
{
    Reply = LSReadPdf417FromBitmap(hWnd,
                    BufFrontImage,
                     BufBarcode,
                      (UINT *)&len_codeline,
                      &nOutGrade,
                       0, 0, 0, 0);

    if (Reply != LS_OKAY)
    {
        CheckReply(hWnd, Reply, "LS500_ReadPdf417FromBitmap");
    }
}



// Salvo la codeline letta
if( BufCodelineSW[0] )
    strcpy(CodelineRead, BufCodelineSW);
else if( BufCodelineHW[0] )
    strcpy(CodelineRead, BufCodelineHW);
else if( BufBarcode[0] )
    strcpy(CodelineRead, BufBarcode);

// Controllo se richiesta di check Codeline
if( (stParDocHandle.DoCheckCodeline ) &&
    (stParDocHandle.CodelineType != NO_READ_CODELINE) )
{
    CheckCodeline(hWnd, CodelineRead, TRUE, (char *)BufFrontImage);
}


if( !stParDocHandle.ViewOnlyLastImage )
{
    ShowCodelineAndImage(Reply,
                        NrCheque,
                        stParDocHandle.Side,
                        (unsigned char *)BufFrontImage,
                        (unsigned char *)BufBackImage,
                        (unsigned char *)BufFrontNettoImage,
                        (unsigned char *)BufBackNettoImage,
                        BufCodelineSW,
                        BufCodelineHW,
                        BufBarcode,
                        "");
```

```
} // if !stParDocHandle.ViewOnlyLastImage


// Salvo codeline su file
if( stParDocHandle.SaveCodeline )
{
        if( BufCodelineHW[0] != '\0' )
        {
                strcat(BufCodelineHW, "\n");
                fputs(BufCodelineHW, fhCodeline);
        }

        if( BufBarcode[0] != '\0' )
        {
                strcat(BufBarcode, "\n");
                fputs(BufBarcode, fhCodeline);
        }

        if( BufCodelineSW[0] != '\0' )
        {
                strcat(BufCodelineSW, "\n");
                fputs(BufCodelineSW, fhCodeline);
        }
}


// free bitmaps
if( Save_BufBackNettoImage )
        GlobalFree( Save_BufBackNettoImage );
Save_BufBackNettoImage = BufBackNettoImage;
BufBackNettoImage = 0;

if( Save_BufBackImage )
        GlobalFree( Save_BufBackImage );
Save_BufBackImage = BufBackImage;
BufBackImage = 0;

if( Save_BufFrontNettoImage )
        GlobalFree( Save_BufFrontNettoImage );
Save_BufFrontNettoImage = BufFrontNettoImage;
BufFrontNettoImage = 0;

if( Save_BufFrontImage )
        GlobalFree( Save_BufFrontImage );
Save_BufFrontImage = BufFrontImage;
BufFrontImage = 0;

memcpy(Save_BufCodelineSW, BufCodelineSW, sizeof(Save_BufCodelineSW));
memcpy(Save_BufCodelineHW, BufCodelineHW, sizeof(Save_BufCodelineHW));
memcpy(Save_BufBarcode, BufBarcode, sizeof(Save_BufBarcode));

while( PeekMessage(&msg, NULL, 0, 0, PM_REMOVE) )
{
        TranslateMessage( &msg );
        DispatchMessage( &msg );
}


if( ((Reply != LS_OKAY) && (Reply != LS_DOUBLE_LEAFING_WARNING)) &&
                    (Reply > LS_DECODE_FONT_NOT_PRESENT))
                {
                    CheckReply(hWnd, Reply, "LSGetDocData");
                    break;
                }


                if( BuilderParam.StopLoopOnErrorInCodeline )
                {
                    if( strchr(BufCodelineHW, '!') )
                        break;

                    if( strchr(BufBarcode, '!') )
                        break;

                    if( strchr(BufCodelineSW, '!') )
                        break;
                }
            } // End while(1)
    }

    if( stParDocHandle.SaveCodeline )
            fclose( fhCodeline );

    return ReplyFunz;
}  // End DoAutoDocHandle
```

14.2. Sample code in C language for the LS800 extracted from LSW8 demo code.

14.2.1. DoAutoDocHandle()

```
int DoAutoDocHandle(HWND hWnd)
{
        int         Reply, Warning;
        BOOL        fExitLoop;
        char        dirBase[_MAX_DIR];
        char        Filename[_MAX_FNAME];
        static char  BufFrontFile[_MAX_PATH];
        static char  BufBackFile[_MAX_PATH];
        static LPHANDLE  BufFrontImage;
        static LPHANDLE  BufBackImage;
        char CodelineType;
        short NumDocRemain;

        static char  BufCodelineSW[CODE_LINE_LENGTH];
        static char  BufCodelineHW[CODE_LINE_LENGTH];
        short len_codeline;
        MSG                 msg;
        long        DocToProcess;
        unsigned short      NrCheque = 1;
        char        SideToFilm;
        long        Lenghts[2];
        HANDLE hBWImage;

        READOPTIONS ro;

        FILE   *fhCodeline = NULL;


        fExitLoop = FALSE;
        CurrentBin = 0;
        if( (stParDocHandle.Sorter & DOC_SEQUENCE_SORTER) == DOC_SEQUENCE_SORTER )
                CurrentBin = 1;
        if( (stParDocHandle.Sorter & DOC_ALL_IN_BIN) == DOC_ALL_IN_BIN )
                CurrentBin = stParDocHandle.Sorter_AllDocInBin;
        fCurrentBin = TRUE;
        SorterFull = 0;


        Reply = LSDoubleLeafingSensibility(hLS, hWnd, 0, (unsigned char)stParDocHandle.DoubleLeafingLevel);
        if (Reply != LS_OKAY)
        {
                if (CheckReply(hWnd, Reply, "LSDoubleLeafingSensibility"))
                {
                   return Reply;
                }
        }

        sprintf(dirBase, "%s%s", PathAppl, SAVE_DIRECTORY_IMAGE);
        strcpy(Filename, NAME_IMAGE);

        SideToFilm = stParDocHandle.Side;
```

```
// if I want to read OCRA or OCRB I force the image to be taken
if( (stParDocHandle.TypeOfDecode & DECODE_OCR) &&
        ((stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRA) ||
         (stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRB_NUM) ||
         (stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRB_ALFANUM) ||
         (stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRB_ITALY) ||
         (stParDocHandle.CodelineOptType == READ_CODELINE_SW_E13B) ||
         (stParDocHandle.CodelineOptType == READ_CODELINE_SW_E13B_X_OCRB)) &&
        (stParDocHandle.Side == SIDE_NONE_IMAGE) )
        SideToFilm = stParDocHandle.Side = SIDE_FRONT_IMAGE;


if( stParDocHandle.TypeOfDecode & DECODE_MICR )
        CodelineType = READ_CODELINE_MICR;
else
        CodelineType = NO_READ_CODELINE;

Reply = LS800AutoDocHandle(hLS,
                        hWnd,
                        stParDocHandle.Validate,
                        CodelineType,
                        SideToFilm,
                        stParDocHandle.ScanModeFront,
                        stParDocHandle.ScanModeBack,
                        stParDocHandle.ClearBlack,
                        stParDocHandle.NumDoc,
                        stParDocHandle.SaveImage,
                        dirBase,
                        Filename,
                        UNIT_MM,
                        stParDocHandle.Codeline_Sw_x,
                        stParDocHandle.Codeline_Sw_y,
                        stParDocHandle.Codeline_Sw_w,
                        stParDocHandle.Codeline_Sw_h,
                        OCR_FRONT_IMAGE,
                        stParDocHandle.FileFormat,
                        stParDocHandle.Quality,
                        SAVE_OVERWRITE,                 // SaveMode
                        1,                                      // PageNumber
                        stParDocHandle.BeepOnError,
                        (CodelineType == READ_CODELINE_MICR ? OnCodelineRead : 0),
                        (CodelineType == NO_READ_CODELINE ? OnImageFrontReady : 0),
                        NULL,//OnImageBackReady,
                        0,                              // Reserved1
                        0,                              // Reserved2
                        NULL);          // Reserved3
if (Reply != LS_OKAY)
{
        if( Reply != LS_FEEDER_EMPTY )
        {
                if (CheckReply(hWnd, Reply, "LS800AutoDocHandle"))
                {
                        Reply = LSDisconnect(hLS, hWnd);
                        return 0;
                }
        }
}

if( stParDocHandle.SaveCodeline )
```

```
{
        strcpy(FullFName, PathAppl);
        strcat(FullFName, FILE_CODELINE);

        if( stParDocHandle.ResetFileCodeline )
                fhCodeline = fopen(FullFName, "w+t");
        else
                fhCodeline = fopen(FullFName, "a+t");
}

if( stParDocHandle.NumDoc )
        DocToProcess = stParDocHandle.NumDoc;
else
        DocToProcess = 1000000L;

//-----------GetDocData--------------------------------------
        while( Reply == LS_OKAY )
        {
                memset((char *)BufFrontFile,      0, sizeof(BufFrontFile));
                memset((char *)BufBackFile,       0, sizeof(BufBackFile));
                BufFrontImage = 0;
                BufBackImage = 0;

                memset((char *)BufCodelineSW,   0, sizeof(BufCodelineSW));
                memset((char *)BufCodelineHW,   0, sizeof(BufCodelineHW));

                Reply = LSGetDocData(hLS,
                                                        hWnd,
                                                        NULL,
                                                        BufFrontFile,
                                                        BufBackFile,
                                                        NULL,
                                                        NULL,
                                                        &BufFrontImage,
                                                        &BufBackImage,
                                                        NULL,
                                                        NULL,
                                                        BufCodelineSW,
                                                        BufCodelineHW,
                                                        NULL,
                                                        NULL,
                                                        &NumDocRemain,
                                                        Lenghts,
                                                        NULL,
                                                        NULL);

                if( Reply != LS_OKAY )
                {
                        switch( Reply )
                        {
                        case LS_SORTER_1_POCKET_1_FULL:
                                SorterFull |= MASK_POCKET_1_FULL;
                                break;

                        case LS_SORTER_1_POCKET_2_FULL:
                                SorterFull |= MASK_POCKET_2_FULL;
                                break;

                        case LS_SORTER_1_POCKET_3_FULL:
```

```
            SorterFull |= MASK_POCKET_3_FULL;
            break;

    case LS_SORTER_2_POCKET_1_FULL:
            SorterFull |= MASK_POCKET_4_FULL;
            break;

    case LS_SORTER_2_POCKET_2_FULL:
            SorterFull |= MASK_POCKET_5_FULL;
            break;

    case LS_SORTER_2_POCKET_3_FULL:
            SorterFull |= MASK_POCKET_6_FULL;
            break;

    case LS_SORTER_3_POCKET_1_FULL:
            SorterFull |= MASK_POCKET_7_FULL;
            break;

    case LS_SORTER_3_POCKET_2_FULL:
            SorterFull |= MASK_POCKET_8_FULL;
            break;

    case LS_SORTER_3_POCKET_3_FULL:
            SorterFull |= MASK_POCKET_9_FULL;
            break;

    case LS_SORTER_4_POCKET_1_FULL:
            SorterFull |= MASK_POCKET_10_FULL;
            break;

    case LS_SORTER_4_POCKET_2_FULL:
            SorterFull |= MASK_POCKET_11_FULL;
            break;

    case LS_SORTER_4_POCKET_3_FULL:
            SorterFull |= MASK_POCKET_12_FULL;
            break;

    case LS_SORTER_5_POCKET_1_FULL:
            SorterFull |= MASK_POCKET_13_FULL;
            break;

    case LS_SORTER_5_POCKET_2_FULL:
            SorterFull |= MASK_POCKET_14_FULL;
            break;

    case LS_SORTER_5_POCKET_3_FULL:
            SorterFull |= MASK_POCKET_15_FULL;
            break;

    default:
            CheckReply(hWnd, Reply, "LSGetDocData");

            if( Reply == LS_PAPER_JAM )
            {
                    Reply = LSReset(hLS, hWnd, RESET_FREE_PATH);
                    CheckReply(hWnd, Reply, "LSReset");
            }
```

```
                    fExitLoop = TRUE;
                    break;
                }

                if( fExitLoop )
                    break;
            }

            if( (SideToFilm == SIDE_ALL_IMAGE) ||
                (SideToFilm == SIDE_FRONT_IMAGE) )
            {
                //-----------ReadCodeline software ------------------------
                if( (stParDocHandle.TypeOfDecode & DECODE_OCR) &&
                    (stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRA ||
                     stParDocHandle.CodelineOptType ==
READ_CODELINE_SW_OCRB_NUM ||
                     stParDocHandle.CodelineOptType ==
READ_CODELINE_SW_OCRB_ALFANUM ||
                     stParDocHandle.CodelineOptType ==
READ_CODELINE_SW_OCRB_ITALY ||
                     stParDocHandle.CodelineOptType == READ_CODELINE_SW_E13B) )
                {
                    ro.PutBlanks = TRUE;
                    ro.TypeRead = 'N';

                    Warning = LSCodelineReadFromBitmap(hWnd,
                                                    BufFrontImage,
                                                    &stParDocHandle.CodelineOptType,
                                                    stParDocHandle.Unit_measure,
                                                    stParDocHandle.Codeline_Sw_x,
                                                    stParDocHandle.Codeline_Sw_y,
                                                    stParDocHandle.Codeline_Sw_w,
                                                    stParDocHandle.Codeline_Sw_h,
                                                    &ro,
                                                    BufCodelineSW,
                                                    (UINT *)&len_codeline);
                }
            }

            if( BufCodelineSW[0] )
                strcpy(CodelineRead, BufCodelineSW);
            else if( BufCodelineHW[0] )
                strcpy(CodelineRead, BufCodelineHW);

            if( (stParDocHandle.DoCheckCodeline ) && (stParDocHandle.TypeOfDecode) )
            {
                CheckCodeline(hWnd, CodelineRead, TRUE);
            }

            if (fOptionViewImage == TRUE)
            {
                ShowCodelineAndImage(Reply,
                                                     NrCheque,
                                                     (unsigned char *)BufFrontImage,
                                                     (unsigned char *)BufBackImage,
                                                     BufCodelineSW,
                                                     BufCodelineHW,
                                                     NULL,
                                                     NULL);
            } // end if fOptionViewImage


            if(stParDocHandle.SaveImage == IMAGE_SAVE_BOTH)
            {
                //Save the image in JPEG
                if( stParDocHandle.FileFormat == SAVE_JPEG )
                {
                    if( ((stParDocHandle.Side == SIDE_FRONT_IMAGE) ||
(stParDocHandle.Side == SIDE_ALL_IMAGE)) &&
                        (BufFrontImage) )
                    {
                        // build filename
                        sprintf(SaveFile, "%s%s\\f%s%03d.jpg", PathAppl,
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileJPG);
                        LSSaveJPEG(hWnd, BufFrontImage, stParDocHandle.Quality,
SaveFile);
                    }

                    if( ((stParDocHandle.Side == SIDE_BACK_IMAGE) ||
(stParDocHandle.Side == SIDE_ALL_IMAGE)) &&
                        (BufBackImage) )
                    {
                        // build filename
                        sprintf(SaveFile, "%s%s\\b%s%03d.jpg", PathAppl,
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileJPG);
                        LSSaveJPEG(hWnd, BufBackImage, stParDocHandle.Quality,
SaveFile);
                    }
                    NrFileJPG ++;
                }

                if( stParDocHandle.FileFormat == SAVE_BMP )
                {
                    if( ((stParDocHandle.Side == SIDE_BACK_IMAGE) ||
(stParDocHandle.Side == SIDE_ALL_IMAGE)) &&
                        (BufBackImage) )
                    {
                        // build filename
                        sprintf(SaveFile, "%s%s\\b%s%03d.bmp", PathAppl,
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileBMP);

                        LSSaveDIB(hWnd, (BITMAPINFOHEADER *)BufBackImage,
SaveFile);
                    }

                    if( ((stParDocHandle.Side == SIDE_FRONT_IMAGE) ||
(stParDocHandle.Side == SIDE_ALL_IMAGE)) &&
                        (BufFrontImage) )
                    {
                        // build filename
                        sprintf(SaveFile, "%s%s\\f%s%03d.bmp", PathAppl,
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileBMP);

                        LSSaveDIB(hWnd, (BITMAPINFOHEADER *)BufFrontImage,
SaveFile);
                    }
                    NrFileBMP ++;
                }
                //Save the image in TIFF
                if( stParDocHandle.FileFormat == FILE_TIF ||
                    stParDocHandle.FileFormat == FILE_CCITT ||
                    stParDocHandle.FileFormat == FILE_CCITT_GROUP3_1DIM ||
                    stParDocHandle.FileFormat == FILE_CCITT_GROUP3_2DIM ||
                    stParDocHandle.FileFormat == FILE_CCITT_GROUP4 )
                {
                    if( ((stParDocHandle.Side == SIDE_BACK_IMAGE) ||
(stParDocHandle.Side == SIDE_ALL_IMAGE)) &&
                        (BufBackImage) )
                    {
                        Reply = LSConvertImageToBW(hWnd, ALGORITHM_CTS ,
BufBackImage, &hBWImage, DEFAULT_POLO_FILTER, 0);

                        if( Reply == LS_OKAY )
                        {
                            // build filename
                            sprintf(SaveFile, "%s%s\\f%s%03d.tif", PathAppl,
SAVE_DIRECTORY_IMAGE, NAME_IMAGE, NrFileTIFF);
                            LSSaveTIFF(hWnd, hBWImage, SaveFile,
stParDocHandle.FileFormat, SAVE_OVERWRITE, 1);

                            GlobalFree( hBWImage );
                        }
                    }

                    if( ((stParDocHandle.Side == SIDE_FRONT_IMAGE) ||
(stParDocHandle.Side == SIDE_ALL_IMAGE)) &&
                        (BufFrontImage) )
                    {
                        Reply = LSConvertImageToBW(hWnd, ALGORITHM_CTS ,
BufFrontImage, &hBWImage, DEFAULT_POLO_FILTER, 0);

                        if( Reply == LS_OKAY )
                        {
                            // build filename
        sprintf(SaveFile, "%s%s\\f%s%03d.tif", PathAppl, SAVE_DIRECTORY_IMAGE, NAME_IMAGE,
NrFileTIFF);
                            LSSaveTIFF(hWnd, hBWImage, SaveFile,
stParDocHandle.FileFormat, SAVE_OVERWRITE, 1);

                            GlobalFree( hBWImage );
                        }
                    }
                    NrFileTIFF ++;
                }
            }

            if( stParDocHandle.SaveCodeline )
            {
                if( BufCodelineHW[0] != '\0' )
                {
                    strcat(BufCodelineHW, "\n");
                    fputs(BufCodelineHW, fhCodeline);
                }

                if( BufCodelineSW[0] != '\0' )
                {
```

```
                                        strcat(BufCodelineSW, "\n");
                                        fputs(BufCodelineSW, fhCodeline);
                                }
                        }

                        if( BufBackImage )
                        {
                                GlobalFree( BufBackImage );
                                BufBackImage = 0;
                        }
                        if( BufFrontImage )
                        {
                                GlobalFree( BufFrontImage );
                                BufFrontImage = 0;
                        }

                        while( PeekMessage(&msg, NULL, 0, 0, PM_REMOVE) )
                        {
                                TranslateMessage( &msg );
                                DispatchMessage( &msg );
                        }


                        if( (-- DocToProcess) == 0 )
                                break;
                        NrCheque ++;

                        if( (stParDocHandle.Sorter & DOC_SEQUENCE_SORTER) ==
DOC_SEQUENCE_SORTER )
                        {
                                switch( CurrentBin )
                                {
                                case 1:
                                        if( Reply == LS_SORTER_1_POCKET_1_FULL )
                                        {
                                                if( (CurrentBin + 1) >= NrPocketPresent )
                                                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                                                else
                                                        CurrentBin ++;
                                        }
                                        break;
                                case 2:
                                        if( Reply == LS_SORTER_1_POCKET_2_FULL )
                                        {
                                                if( (CurrentBin + 1) >= NrPocketPresent )
                                                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                                                else
                                                        CurrentBin ++;
                                        }
                                        break;
                                case 3:
                                        if( Reply == LS_SORTER_1_POCKET_3_FULL )
                                        {
                                                if( (CurrentBin + 1) >= NrPocketPresent )
                                                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                                                else
                                                        CurrentBin ++;
                                        }
                                        break;
```

```
case 4:
        if( Reply == LS_SORTER_2_POCKET_1_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                else
                        CurrentBin ++;
        }
        break;
case 5:
        if( Reply == LS_SORTER_2_POCKET_2_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                else
                        CurrentBin ++;
        }
        break;
case 6:
        if( Reply == LS_SORTER_2_POCKET_3_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                else
                        CurrentBin ++;
        }
        break;
case 7:
        if( Reply == LS_SORTER_3_POCKET_1_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                else
                        CurrentBin ++;
        }
        break;
case 8:
        if( Reply == LS_SORTER_3_POCKET_2_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                else
                        CurrentBin ++;
        }
        break;
case 9:
        if( Reply == LS_SORTER_3_POCKET_3_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
                else
                        CurrentBin ++;
        }
        break;
case 10:
        if( Reply == LS_SORTER_4_POCKET_1_FULL )
        {
                if( (CurrentBin + 1) >= NrPocketPresent )
                        Reply = LSStopAutoDocHandle(hLS, hWnd);
```

```
            else
                CurrentBin ++;
        }
        break;
    case 11:
        if( Reply == LS_SORTER_4_POCKET_2_FULL )
        {
            if( (CurrentBin + 1) >= NrPocketPresent )
                Reply = LSStopAutoDocHandle(hLS, hWnd);
            else
                CurrentBin ++;
        }
        break;
    case 12:
        if( Reply == LS_SORTER_4_POCKET_3_FULL )
        {
            if( (CurrentBin + 1) >= NrPocketPresent )
                Reply = LSStopAutoDocHandle(hLS, hWnd);
            else
                CurrentBin ++;
        }
        break;
    case 13:
        if( Reply == LS_SORTER_5_POCKET_1_FULL )
        {
            if( (CurrentBin + 1) >= NrPocketPresent )
                Reply = LSStopAutoDocHandle(hLS, hWnd);
            else
                CurrentBin ++;
        }
        break;
    case 14:
        if( Reply == LS_SORTER_5_POCKET_2_FULL )
        {
            if( (CurrentBin + 1) >= NrPocketPresent )
                Reply = LSStopAutoDocHandle(hLS, hWnd);
            else
                CurrentBin ++;
        }
        break;
    case 15:
        if( Reply == LS_SORTER_5_POCKET_3_FULL )
        {
            if( (CurrentBin + 1) >= NrPocketPresent )
                Reply = LSStopAutoDocHandle(hLS, hWnd);
            else
                CurrentBin ++;
        }
        break;
    }
}
if( (stParDocHandle.Sorter & DOC_ALL_IN_BIN) == DOC_ALL_IN_BIN )
{
    switch( CurrentBin )
    {
    case 1:
        if( Reply == LS_SORTER_1_POCKET_1_FULL )
            Reply = LSStopAutoDocHandle(hLS, hWnd);
        break;
```

```
		case 2:
			if( Reply == LS_SORTER_1_POCKET_2_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 3:
			if( Reply == LS_SORTER_1_POCKET_3_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 4:
			if( Reply == LS_SORTER_2_POCKET_1_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 5:
			if( Reply == LS_SORTER_2_POCKET_2_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 6:
			if( Reply == LS_SORTER_2_POCKET_3_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 7:
			if( Reply == LS_SORTER_3_POCKET_1_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 8:
			if( Reply == LS_SORTER_3_POCKET_2_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 9:
			if( Reply == LS_SORTER_3_POCKET_3_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 10:
			if( Reply == LS_SORTER_4_POCKET_1_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 11:
			if( Reply == LS_SORTER_4_POCKET_2_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 12:
			if( Reply == LS_SORTER_4_POCKET_3_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 13:
			if( Reply == LS_SORTER_5_POCKET_1_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 14:
			if( Reply == LS_SORTER_5_POCKET_2_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		case 15:
			if( Reply == LS_SORTER_5_POCKET_3_FULL )
				Reply = LSStopAutoDocHandle(hLS, hWnd);
			break;
		}
	}
```

```
            if( (Reply >= LS_SORTER_1_POCKET_1_FULL) &&
                (Reply <= LS_SORTER_5_POCKET_3_FULL) )
                Reply = LS_OKAY;

        }   // Fine while( TRUE )
    if( stParDocHandle.SaveCodeline )
        fclose( fhCodeline );


    return Reply;
}  // End DoAutoDocHandle
```

Callback functions:

```
int OnCodelineRead(S_CODELINE_INFO_LS800 *CodelineInfo)
{
    short ii;


    if( (stParDocHandle.Sorter & DOC_CIRCULAR_SORTER) == DOC_CIRCULAR_SORTER )
    {
        // Set next sorter ...
        if( fCurrentBin )
        {
            CodelineInfo->Sorter = CurrentBin ++;
            if( CurrentBin > NrPocketPresent )
            {
                if( NrPocketPresent > 1 )
                {
                    fCurrentBin = !fCurrentBin;
                    CurrentBin -= 2;
                    CodelineInfo->Sorter = 0;
                }
                else
                    CurrentBin = 0;             // Only pocket zero
            }
        }
        else
        {
            CodelineInfo->Sorter = CurrentBin --;
            if( CurrentBin <= 0 )
                fCurrentBin = !fCurrentBin;
        }

        // ... and the string
        if( stParDocHandle.Sorter_Validate[DATI_CIRCULAR] )
        {
            CodelineInfo->FormatString1 = FORMAT_NORMAL;
            if( stParDocHandle.Sorter_PrintBold[DATI_CIRCULAR] )
                CodelineInfo->FormatString1 = FORMAT_BOLD;
            strcpy(CodelineInfo->StringToPrint1,
               stParDocHandle.Sorter_szValidateText[DATI_CIRCULAR]);
        }
    }
    else if( (stParDocHandle.Sorter & DOC_SEQUENCE_SORTER) == DOC_SEQUENCE_SORTER )
    {
        // Set the sorter ...
        CodelineInfo->Sorter = CurrentBin;
        // ... and the string
        if( stParDocHandle.Sorter_Validate[DATI_SEQUENCE] )
        {
            CodelineInfo->FormatString1 = FORMAT_NORMAL;
            if( stParDocHandle.Sorter_PrintBold[DATI_SEQUENCE] )
                CodelineInfo->FormatString1 = FORMAT_BOLD;
            strcpy(CodelineInfo->StringToPrint1,
               stParDocHandle.Sorter_szValidateText[DATI_SEQUENCE]);
        }
    }
    else if( (stParDocHandle.Sorter & DOC_ALL_IN_BIN) == DOC_ALL_IN_BIN )
    {
        // Set the sorter ...
        CodelineInfo->Sorter = CurrentBin;

        // ... and the string
        if( stParDocHandle.Sorter_Validate[DATI_ALL_DOC] )
        {
            CodelineInfo->FormatString1 = FORMAT_NORMAL;
            if( stParDocHandle.Sorter_PrintBold[DATI_ALL_DOC] )
                CodelineInfo->FormatString1 = FORMAT_BOLD;
            strcpy(CodelineInfo->StringToPrint1,
               stParDocHandle.Sorter_szValidateText[DATI_ALL_DOC]);
        }
    }
    else
    {
        // Search the criteria associated at the codeline read
        for( ii = 0; ii < NrPocketPresent; ii ++)
        {
            switch( stParDocHandle.DocOption[ii].TypeChoice )
            {
            case 0:
                // Do nothing
                break;

            case 1:
                // Check for error in codeline
                if( strchr(CodelineInfo->CodelineRead, '!') )
                {
                    // Set of the sorter ...
                    CodelineInfo->Sorter = ii;

                    // ... and the string to print if present
                    if( stParDocHandle.DocOption[ii].fDoPrint )
                    {
                        if( stParDocHandle.DocOption[ii].fPrintBold )
                            CodelineInfo->FormatString1 = FORMAT_BOLD;
                        else
                            CodelineInfo->FormatString1 = FORMAT_NORMAL;

                        strcpy(CodelineInfo->StringToPrint1,
                           stParDocHandle.DocOption[ii].StringToPrint);
                    }

                    // Set ii = NrPocketPresent for terminate the for
                    ii = NrPocketPresent + 2;
                }
```

```
                break;

            case 2:
                // Check if the right codeline
                if( ! strncmp(&CodelineInfo-
>CodelineRead[stParDocHandle.DocOption[ii].StartChar - 1],
                            stParDocHandle.DocOption[ii].StringCheck,
                            strlen(stParDocHandle.DocOption[ii].StringCheck)) )
                {
                    // Set of the sorter ...
                    CodelineInfo->Sorter = ii;

                    // ... and the string to print if present
                    if( stParDocHandle.DocOption[ii].fDoPrint )
                    {
                        if( stParDocHandle.DocOption[ii].fPrintBold )
                            CodelineInfo->FormatString1 = FORMAT_BOLD;
                        else
                            CodelineInfo->FormatString1 = FORMAT_NORMAL;

                        strcpy(CodelineInfo->StringToPrint1,
                           stParDocHandle.DocOption[ii].StringToPrint);
                    }

                    // Set ii = NrPocketPresent for terminate the for
                    ii = NrPocketPresent + 2;
                }
            }
        }
    }

    // test if a sorter is associated was found
    if( ii == NrPocketPresent )
        // Set a sorter of default
        CodelineInfo->Sorter = SORTER_0_SELECTED;

    return 0;
} // OnCodelineRead


int OnImageFrontReady(S_IMAGE_INFO_LS800 *ImgInfo)
{
    int Warning;
    short len_codeline;
    short ii;
    char CodelineRead[CODE_LINE_LENGTH];
    READOPTIONS ro;


    if( (stParDocHandle.Sorter & DOC_CIRCULAR_SORTER) == DOC_CIRCULAR_SORTER )
    {
        // Set next sorter ...
        if( fCurrentBin )
        {
            ImgInfo->Sorter = CurrentBin ++;
            if( CurrentBin > NrPocketPresent )
            {
                if( NrPocketPresent > 1 )
                {
```

```
                                fCurrentBin = !fCurrentBin;
                                CurrentBin -= 2;
                                ImgInfo->Sorter = 0;
                        }
                        else
                                CurrentBin = 0;         // Only pocket zero
                }
        }
        else
        {
                ImgInfo->Sorter = CurrentBin --;
                if( CurrentBin <= 0 )
                        fCurrentBin = !fCurrentBin;
        }

        // ... and the string
        if( stParDocHandle.Sorter_Validate[DATI_CIRCULAR] )
        {
                ImgInfo->FormatString1 = FORMAT_NORMAL;
                if( stParDocHandle.Sorter_PrintBold[DATI_CIRCULAR] )
                        ImgInfo->FormatString1 = FORMAT_BOLD;
                strcpy(ImgInfo->StringToPrint1,
                      stParDocHandle.Sorter_szValidateText[DATI_CIRCULAR]);
        }
}
else if( (stParDocHandle.Sorter & DOC_SEQUENCE_SORTER) == DOC_SEQUENCE_SORTER )
{
        // Set the sorter ...
        ImgInfo->Sorter = CurrentBin;

        // ... and the string
        if( stParDocHandle.Sorter_Validate[DATI_SEQUENCE] )
        {
                ImgInfo->FormatString1 = FORMAT_NORMAL;
                if( stParDocHandle.Sorter_PrintBold[DATI_SEQUENCE] )
                        ImgInfo->FormatString1 = FORMAT_BOLD;
                strcpy(ImgInfo->StringToPrint1,
                      stParDocHandle.Sorter_szValidateText[DATI_SEQUENCE]);
        }
}
else if( (stParDocHandle.Sorter & DOC_ALL_IN_BIN) == DOC_ALL_IN_BIN )
{
        // Set the sorter ...
        ImgInfo->Sorter = CurrentBin;

        // ... and the string
        if( stParDocHandle.Sorter_Validate[DATI_ALL_DOC] )
        {
                ImgInfo->FormatString1 = FORMAT_NORMAL;
                if( stParDocHandle.Sorter_PrintBold[DATI_ALL_DOC] )
                        ImgInfo->FormatString1 = FORMAT_BOLD;
                strcpy(ImgInfo->StringToPrint1,
                       stParDocHandle.Sorter_szValidateText[DATI_ALL_DOC]);
        }
}
else
{
        // Undecode the codeline OCR
        if( (stParDocHandle.TypeOfDecode & DECODE_OCR) &&
    (stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRA ||
     stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRB_NUM ||
     stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRB_ALFANUM ||
     stParDocHandle.CodelineOptType == READ_CODELINE_SW_OCRB_ITALY ||
     stParDocHandle.CodelineOptType == READ_CODELINE_SW_E13B) )
{
    ro.PutBlanks = TRUE;
    ro.TypeRead = 'N';

    Warning = LSCodelineReadFromBitmap(0,
                    ImgInfo->hImage,
                    &stParDocHandle.CodelineOptType,
                    stParDocHandle.Unit_measure,
                    stParDocHandle.Codeline_Sw_x,
                    stParDocHandle.Codeline_Sw_y,
                    stParDocHandle.Codeline_Sw_w,
                    stParDocHandle.Codeline_Sw_h,
                    &ro,
                    CodelineRead,
                    (UINT *)&len_codeline);
    if( Warning != LS_OKAY )
    {
        // On error set sorter ... zero
        ImgInfo->Sorter = 0;
        return 0;
    }
}

// Search the criteria associated at the codeline read
for( ii = 0; ii < NrPocketPresent; ii ++)
{
    switch( stParDocHandle.DocOption[ii].TypeChoice )
    {
    case 0:
        // Do nothing
        break;

    case 1:
        // Check for error in codeline
        if( strchr(CodelineRead, '!') )
        {
            // Set of the sorter ...
            ImgInfo->Sorter = ii;

            // ... and the string to print if present
            if( stParDocHandle.DocOption[ii].fDoPrint )
            {
                if( stParDocHandle.DocOption[ii].fPrintBold )
                    ImgInfo->FormatString1 = FORMAT_BOLD;
                else
                    ImgInfo->FormatString1 = FORMAT_NORMAL;

                strcpy(ImgInfo->StringToPrint1,
                      stParDocHandle.DocOption[ii].StringToPrint);
            }

            // Set ii = NrPocketPresent for terminate the for
            ii = NrPocketPresent + 2;
        }
```

```
                break;

            case 2:
                // Check if the right codeline
                if( ! strncmp(&CodelineRead[stParDocHandle.DocOption[ii].StartChar - 1],
                                            stParDocHandle.DocOption[ii].StringCheck,
                                            strlen(stParDocHandle.DocOption[ii].StringCheck)) )
                {
                    // Set of the sorter ...
                    ImgInfo->Sorter = ii;

                    // ... and the string to print if present
                    if( stParDocHandle.DocOption[ii].fDoPrint )
                    {
                        if( stParDocHandle.DocOption[ii].fPrintBold )
                            ImgInfo->FormatString1 = FORMAT_BOLD;
                        else
                            ImgInfo->FormatString1 = FORMAT_NORMAL;

                        strcpy(ImgInfo->StringToPrint1,
                            stParDocHandle.DocOption[ii].StringToPrint);
                    }

                    // Set ii = NrPocketPresent for terminate the for
                    ii = NrPocketPresent + 2;
                }
            }
        }
    }

    // test if a sorter is associated was found
    if( ii == NrPocketPresent )
        // Set a sorter of default
        ImgInfo->Sorter = SORTER_0_SELECTED;

    return 0;
} // OnImageFrontReady
```

14.3. Define and Declare for Visual Basic

Add a module in your VB program with the following definitions:

```
'// Parameter Type peripheral
Public Const LS_100_USB = 100
Public Const LS_100_RS232 = 101 '(not yet supported)
Public Const LS_215_USB = 201
Public Const LS_5xx_SCSI = 500
Public Const LS_510_TCPIP = 501 '(not yet supported)
Public Const LS_515_USB = 502
Public Const LS_800_USB = 801

'// Parameter Type_com
Public Const SUSPENSIVE_MODE = &H53                    'S'
Public Const NOT_SUSPENSIVE_MODE = &H54                      'T'

'// Parameter FrontStamp
Public Const NO_STAMP = 0
Public Const FRONT_STAMP = 1
Public Const BACK_STAMP = 2
Public Const FRONT_AND_BACK_STAMP = 3

'// Parameter Validate
Public Const NO_PRINT_VALIDATE = 0
Public Const PRINT_VALIDATE = 1

'// Parameter Feed
Public Const AUTO_FEED = 0
Public Const PATH_FEED = 1

'// Parameter Sorter
Public Const HOLD_DOCUMENT = 0
Public Const SORTER_BAY1 = 1
Public Const SORTER_BAY2 = 2
Public Const SORTER_AUTOMATIC = 3
Public Const SORTER_SWITCH_1_TO_2 = 4
Public Const EJECT_DOCUMENT = 5
Public Const SORTER_ON_CODELINE_CALLBACK = 6

'// Parameter Codeline
Public Const NO_READ_CODELINE = 0
Public Const READ_CODELINE_MICR = 1
Public Const READ_BARCODE_PDF417 = 2
Public Const READ_BARCODE_HW = 6
Public Const READ_MICR_AND_BARCODE_HW = 7
Public Const READ_CODELINE_OPTIC = 21
Public Const READ_MICR_AND_OPTIC = 22
Public Const READ_OPTIC_AND_PDF417 = 23

Public Const READ_BARCODE_2_OF_5 = 50
Public Const READ_BARCODE_CODE39 = 51
Public Const READ_BARCODE_CODE128 = 52
Public Const READ_BARCODE_EAN13 = 53

Public Const READ_CODELINE_HW_OCRA = &H41           '//'A'
Public Const READ_CODELINE_HW_OCRB_NUM = &H42        '//'B'
Public Const READ_CODELINE_HW_OCRB_ALFANUM = &H43    '//'C'
Public Const READ_CODELINE_HW_E13B = &H45           '//'E'
Public Const READ_CODELINE_HW_E13B_X_OCRB = &H58    '//'X'
Public Const READ_CODELINE_HW_MULTI_READ = &H4D     '//'M'

Public Const READ_CODELINE_SW_OCRA = &H41           '//'A'
Public Const READ_CODELINE_SW_OCRB_NUM = &H42       '//'B'
Public Const READ_CODELINE_SW_OCRB_ALFANUM = &H43    '//'C'
Public Const READ_CODELINE_SW_OCRB_ITALY = &H46      '//'F'
Public Const READ_CODELINE_SW_E13B = &H45           '//'E'
Public Const READ_CODELINE_SW_E13B_X_OCRB = &H58    '//'X'
Public Const READ_CODELINE_SW_MULTI_READ = &H4D     '//'M'

Public Const READ_ONE_CODELINE_TYPE = &H4E          '//'N'

'// Parameter OriginMeasureDoc
Public Const BOTTOM_LEFT_PIXEL = 0
Public Const BOTTOM_RIGHT_MM = 10
Public Const BOTTOM_RIGHT_INCH = 20


'// Parameter Unit
Public Const UNIT_MM = 0
Public Const UNIT_INCH = 1

'// Value of height to decode a software Codeline
Public Const MAX_PIXEL_HEIGHT = 42
Public Const OCR_VALUE_IN_MM = 10.5
Public Const OCR_VALUE_IN_INCH = 0.41

'// Parameter OCR_Image_Side
Public Const OCR_FRONT_IMAGE = 0
Public Const OCR_BACK_IMAGE = 1

'// Parameter ScanMode
Public Const SCAN_MODE_BW = 1
Public Const SCAN_MODE_16GR100 = 2
Public Const SCAN_MODE_16GR200 = 3
Public Const SCAN_MODE_256GR100 = 4
Public Const SCAN_MODE_256GR200 = 5
Public Const SCAN_MODE_COLOR_100 = 10
Public Const SCAN_MODE_COLOR_200 = 11

'// Parameter ReadMode
Public Const READMODE_BRUTTO = 0
Public Const READMODE_NETTO = 1
Public Const READMODE_ALL = 2

'// Parameter ClearBlack
Public Const NO_CLEAR_BLACK = 0
Public Const CLEAR_ALL_BLACK = 1

'// Parameter Side
Public Const SIDE_NONE_IMAGE = &H4E         'N'
Public Const SIDE_FRONT_IMAGE = &H46        'F'
Public Const SIDE_BACK_IMAGE = &H42         'B'
Public Const SIDE_ALL_IMAGE = &H58          'X'
Public Const SIDE_FRONT_BLUE_IMAGE = &H47   'G'
Public Const SIDE_BACK_BLUE_IMAGE = &H43    'C'
Public Const SIDE_ALL_BLUE_IMAGE = &H59     'Y'
Public Const SIDE_FRONT_GREEN_IMAGE = &H48  'H'
Public Const SIDE_BACK_GREEN_IMAGE = &H44   'D'
Public Const SIDE_ALL_GREEN_IMAGE = &H57    'W'
Public Const SIDE_FRONT_RED_IMAGE = &H49    'I'
Public Const SIDE_BACK_RED_IMAGE = &H45     'E'
Public Const SIDE_ALL_RED_IMAGE = &H5A      'Z'

'// Parameter Image Coordinate
Public Const IMAGE_MAX_WIDTH = 1720
Public Const IMAGE_MAX_HEIGHT = 848

'// Parameter Method
Public Const ALGORITHM_CTS = 4
Public Const ALGORITHM_NODITHERING = &H10
Public Const ALGORITHM_FLOYDSTEINDITHERING = &H11
Public Const ALGORITHM_STUCKIDITHERING = &H12
Public Const ALGORITHM_BURKESDITHERING = &H13
Public Const ALGORITHM_SIERRADITHERING = &H14
Public Const ALGORITHM_STEVENSONARCEDITHERING = &H15
Public Const ALGORITHM_JARVISDITHERING = &H16

Public Const DEFAULT_POLO_FILTER = 450

'// Parameter Format
Public Const FORMAT_NORMAL = &H4E           'N'
Public Const FORMAT_BOLD = &H42             'B'
Public Const FORMAT_NORMAL_15 = &H41            'A'
Public Const PRINT_UP_FORMAT_NORMAL = &H6E      'n'
Public Const PRINT_UP_FORMAT_BOLD = &H62        'b'
Public Const  PRINT_UP_ NORMAL_15_CHAR = &H61       'a'

Public Const FORMAT_IATA = &H20             '// Badge Track 1
Public Const FORMAT_ABA = &H40              '// Badge Track 2
Public Const FORMAT_MINTS = &H80            '// Badge Track 3
Public Const FORMAT_IATA_ABA = &H60         '// Badge Track 1 and 2
Public Const FORMAT_ABA_MINTS = &HC0        '// Badge Track 2 and 3

'// Parameter Wait_com
Public Const WAIT_NO = &H47                 'G'
Public Const WAIT_YES = &H57                'W'

'//Parameter Beep
Public Const NO_BEEP = 0
Public Const YES_BEEP = 1

'//Parameter SaveOnFile
Public Const IMAGE_SAVE_ON_FILE = 4
Public Const IMAGE_SAVE_HANDLE = 5
Public Const IMAGE_SAVE_BOTH = 6
Public Const IMAGE_SAVE_NONE = 7

'//Parameter FileFormat
Public Const SAVE_JPEG = 10
Public Const SAVE_BMP = 11

'// Parameter: Tiff type
Public Const FILE_TIF = 3                       '// Tagged Image File Format
Public Const FILE_CCITT = 25                    '// TIFF  CCITT
Public Const FILE_CCITT_GROUP3_1DIM = 27        '// CCITT Group3 one dimension
Public Const FILE_CCITT_GROUP3_2DIM = 28        '// CCITT Group3 two dimensions
Public Const FILE_CCITT_GROUP4 = 29                '// CCITT Group4 two dimensions

'// Parameter: uSaveMulti
Public Const SAVE_OVERWRITE = 0
Public Const SAVE_APPEND = 1
Public Const SAVE_REPLACE = 2
Public Const SAVE_INSERT = 3

'// Scanner choise
Public Const SCANNER_BACK = 0
Public Const SCANNER_FRONT = 1

'// Printer choise
Public Const PRINTER_FRONT = 0
Public Const PRINTER_BACK = 1

Public Const CODE_LINE_LENGTH = 256                '// Max length of returned codeline

Public Const MAX_OPTICAL_WINDOWS = 5               '// Nr. window * 5 bytes per window
Public Const MAX_CRITERIA = 5                  '// Nr. max of selection criteria
Public Const MAX_CHAR_CHECK = 10                   '// Nr. max of check char

'// Parameter Sorter Criteria only for LS515
Public Const CRITERIA_NO = &H0
Public Const CRITERIA_ERROR_IN_CODELINE = &H1
Public Const CRITERIA_CODELINE_EQUAL_STR1 = &H2
Public Const CRITERIA_CODELINE_DIFF_STR1 = &H3
Public Const CRITERIA_CODELINE_GREAT_STR1 = &H4
Public Const CRITERIA_CODELINE_MIN_STR1 = &H5
Public Const CRITERIA_CODELINE_INTO_STR1_STR2 = &H6
Public Const CRITERIA_CODELINE_OUT_STR1_STR2 = &H7
Public Const CRITERIA_CODELINE_EQUAL_STR1_OR_STR2 = &H8
Public Const CRITERIA_CODELINE_DIFF_STR1_AND_STR2 = &H9

'// Parameter Double Leafing only for LS515
Public Const DOUBLE_LEAFING_WARNING = 0
Public Const DOUBLE_LEAFING_ERROR = 1

Public Const DOUBLE_LEAFING_STORAGE = 10
Public Const DOUBLE_LEAFING_VOLATILE = 11

Public Const DOUBLE_LEAFING_LEVEL1 = &H1
Public Const DOUBLE_LEAFING_LEVEL2 = &H2
Public Const DOUBLE_LEAFING_LEVEL3 = &H3
Public Const DOUBLE_LEAFING_DEFAULT = &H4
Public Const DOUBLE_LEAFING_LEVEL4 = &H5
Public Const DOUBLE_LEAFING_LEVEL5 = &H6
Public Const DOUBLE_LEAFING_DISABLE = &H7

'// Parameter History
Public Const CMD_READ_HISTORY = 1
Public Const CMD_ERASE_HISTORY = 2


Public Const PERIPHERAL_LS100_SIZE_MEMORY = 36864 '36 * 1024 '//Total memory of the peripheral
Public Const PERIPHERAL_LS5xx_SIZE_MEMORY = 65536 '64 * 1024 '//Total memory of the peripheral


'// String for identify the periferal connected
Public Const MODEL_LS100_1 = "LS100USB"
Public Const MODEL_LS100_2 = "LS100RS_"
Public Const MODEL_LS100_3 = "LS100/3_"
Public Const MODEL_LS100_4 = "LS100/4_"
Public Const MODEL_LS100_5 = "LS100/5_"

Public Const MODEL_LS200_USB = "LS200USB"
Public Const MODEL_LS200_2 = "C.T.S.  LS200/2"
Public Const MODEL_LS800_1 = "LS8/1"
Public Const MODEL_LS800_2 = "LS8/2"

Public Const MODEL_LS500 = "C.T.S.  LS500"
Public Const MODEL_LS505 = "C.T.S.  LS505"
Public Const MODEL_LS510S = "C.T.S.  LS510S"
Public Const MODEL_LS510D = "C.T.S.  LS510D"
Public Const MODEL_LS515S = "C.T.S.  LS515S"
Public Const MODEL_LS515D = "C.T.S.  LS515D"
Public Const MODEL_LS5151 = "C.T.S.  LS515/1"
Public Const MODEL_LS5152 = "C.T.S.  LS515/2"
Public Const MODEL_LS5153 = "C.T.S.  LS515/3"
Public Const MODEL_LS5155 = "C.T.S.  LS515/5"
Public Const MODEL_LS5156 = "C.T.S.  LS515/6"



'// -----------------------------------------------------------------------
'//                        REPLY-CODE
'// -----------------------------------------------------------------------

Public Const LS_OKAY = 0

'// -----------------------------------------------------------------------
'//                 ERRORS
'// -----------------------------------------------------------------------
Public Const LS_SYSTEM_ERROR = -1
Public Const LS_USB_ERROR = -2
Public Const LS_PERIPHERAL_NOT_FOUND = -3
Public Const LS_HARDWARE_ERROR = -4
Public Const LS_PERIPHERAL_OFF_ON = -5
Public Const LS_RESERVED_ERROR = -6
Public Const LS_PAPER_JAM = -7
Public Const LS_TARGET_BUSY = -8
Public Const LS_INVALID_COMMAND = -9
Public Const LS_DATA_LOST = -10
Public Const LS_COMMAND_IN_EXECUTION_YET = -11
Public Const LS_JPEG_ERROR = -12
Public Const LS_COMMAND_SEQUENCE_ERROR = -13
Public Const LS_NOT_USED = -14
Public Const LS_IMAGE_OVERWRITE = -15
Public Const LS_INVALID_HANDLE = -16
Public Const LS_NO_LIBRARY_LOAD = -17
Public Const LS_BMP_ERROR = -18
Public Const LS_TIFF_ERROR = -19
Public Const LS_IMAGE_NO_MORE_AVAILABLE = -20
Public Const LS_IMAGE_NO_FILMED = -21
Public Const LS_IMAGE_NOT_PRESENT = -22
Public Const LS_FUNCTION_NOT_AVAILABLE = -23
Public Const LS_DOCUMENT_NOT_SUPPORTED = -24
Public Const LS_BARCODE_ERROR = -25
Public Const LS_INVALID_LIBRARY = -26
Public Const LS_INVALID_IMAGE = -27
Public Const LS_INVALID_IMAGE_FORMAT = -28
Public Const LS_INVALID_BARCODE_TYPE = -29
Public Const LS_OPEN_NOT_DONE = -30
Public Const LS_INVALID_TYPE_COMMAND = -31
Public Const LS_INVALID_CLEARBLACK = -32
Public Const LS_INVALID_SIDE = -33
Public Const LS_MISSING_IMAGE = -34
Public Const LS_INVALID_TYPE = -35
Public Const LS_INVALID_SAVEMODE = -36
Public Const LS_INVALID_PAGE_NUMBER = -37
Public Const LS_INVALID_NRIMAGE = -38
Public Const LS_INVALID_STAMP = -39
Public Const LS_INVALID_WAITTIMEOUT = -40
Public Const LS_INVALID_VALIDATE = -41
Public Const LS_INVALID_CODELINE_TYPE = -42
Public Const LS_MISSING_NRIMAGE = -43
Public Const LS_INVALID_SCANMODE = -44
Public Const LS_INVALID_BEEP = -45
Public Const LS_INVALID_FEEDER = -46
Public Const LS_INVALID_SORTER = -47
Public Const LS_INVALID_BADGE_TRACK = -48
Public Const LS_MISSING_FILENAME = -49
Public Const LS_INVALID_QUALITY = -50
Public Const LS_INVALID_FILEFORMAT = -51
Public Const LS_INVALID_COORDINATE = -52
Public Const LS_MISSING_HANDLE_VARIABLE = -53
Public Const LS_INVALID_POLO_FILTER = -54
Public Const LS_INVALID_ORIGIN_MEASURES = -55
Public Const LS_INVALID_SIZEH_VALUE = -56
Public Const LS_INVALID_FORMAT = -57
Public Const LS_STRINGS_TOO_LONGS = -58
Public Const LS_READ_IMAGE_FAILED = -59
Public Const LS_INVALID_CMD_HISTORY = -60
Public Const LS_MISSING_BUFFER_HISTORY = -61
Public Const LS_INVALID_ANSWER = -62
Public Const LS_OPEN_FILE_ERROR_OR_NOT_FOUND = -63
Public Const LS_READ_TIMEOUT_EXPIRED = -64
Public Const LS_INVALID_METHOD = -65
Public Const LS_CALIBRATION_FAILED = -66
Public Const LS_INVALID_SAVEIMAGE = -67
Public Const LS_UNIT = -68
Public Const LS_INVALID_NRWINDOWS = -71
Public Const LS_INVALID_VALUE = -72
Public Const LS_ILLEGAL_REQUEST = -73
Public Const LS_INVALID_NR_CRITERIA = -74
Public Const LS_MISSING_CRITERIA_STRUCTURE = -75
Public Const LS_INVALID_MOVEMENT = -76
Public Const LS_INVALID_DEGREE = -77
Public Const LS_R0TATE_ERROR = -78

Public Const LS_SCAN_NETTO_IMAGE_NOT_SUPPORTED = -521
Public Const LS_256_GRAY_NOT_SUPPORTED = -522
Public Const LS_INVALID_PATH = -523
Public Const LS_MISSING_CALLBACK_FUNCTION = -526
Public Const LS_INVALID_OCR_IMAGE_SIDE = -558
Public Const LS_PERIPHERAL_NOT_ANSWER = -599

Public Const LS_INVALID_CONNECTION_HANDLE = -1000
Public Const LS_INVALID_CONNECT_PERIPHERAL = -1001
Public Const LS_PERIPHERAL_NOT_YET_INTEGRATE = -1002
Public Const LS_UNKNOW_PERIPHERAL_REPLY = -1003

Public Const LS_DECODE_FONT_NOT_PRESENT = -1101
Public Const LS_DECODE_INVALID_COORDINATE = -1102
Public Const LS_DECODE_INVALID_OPTION = -1103
Public Const LS_DECODE_INVALID_CODELINE_TYPE = -1104
Public Const LS_DECODE_SYSTEM_ERROR = -1105
Public Const LS_DECODE_DATA_TRUNC = -1106
Public Const LS_DECODE_INVALID_BITMAP = -1107
Public Const LS_DECODE_ILLEGAL_USE = -1108

Public Const LS_BARCODE_GENERIC_ERROR = -1201
Public Const LS_BARCODE_NOT_DECODABLE = -1202
Public Const LS_BARCODE_OPENFILE_ERROR = -1203
Public Const LS_BARCODE_READBMP_ERROR = -1204
Public Const LS_BARCODE_MEMORY_ERROR = -1205
Public Const LS_BARCODE_START_NOTFOUND = -1206
Public Const LS_BARCODE_STOP_NOTFOUND = -1207

Public Const LS_PDF_NOT_DECODABLE = -1301
Public Const LS_PDF_READBMP_ERROR = -1302
Public Const LS_PDF_BITMAP_FORMAT_ERROR = -1303
Public Const LS_PDF_MEMORY_ERROR = -1304
Public Const LS_PDF_START_NOTFOUND = -1305
Public Const LS_PDF_STOP_NOTFOUND = -1306
Public Const LS_PDF_LEFTIND_ERROR = -1307
Public Const LS_PDF_RIGHTIND_ERROR = -1308
Public Const LS_PDF_OPENFILE_ERROR = -1309


'// -----------------------------------------------------------------------
'//                   WARNINGS
'// -----------------------------------------------------------------------
Public Const LS_FEEDER_EMPTY = 1
Public Const LS_DATA_TRUNCATED = 2
Public Const LS_DOC_PRESENT = 3
Public Const LS_BADGE_TIMEOUT = 4
Public Const LS_ALREADY_OPEN = 5
Public Const LS_PERIF_BUSY = 6
Public Const LS_DOUBLE_LEAFING_WARNING = 7
Public Const LS_COMMAND_NOT_ENDED = 8
Public Const LS_RETRY = 9
Public Const LS_NO_OTHER_DOCUMENT = 10
Public Const LS_QUEUE_FULL = 11
Public Const LS_NO_SENSE = 12
Public Const LS_TRY_TO_RESET = 14
Public Const LS_STRING_TRUNCATED = 15
Public Const LS_COMMAND_NOT_SUPPORTED = 19
Public Const LS_DOUBLE_LEAFING_ERROR = 38
Public Const LS_KEEP_DOC_ON_CODELINE_ERROR = 39
Public Const LS_LOOP_INTERRUPTED = 40
Public Const LS_SORTER1_FULL = 35
Public Const LS_SORTER2_FULL = 36
Public Const LS_SORTERS_BOTH_FULL = 37
Public Const LS_SORTER_1_POCKET_1_FULL = 51
Public Const LS_SORTER_1_POCKET_2_FULL = 52
Public Const LS_SORTER_1_POCKET_3_FULL = 53
Public Const LS_SORTER_2_POCKET_1_FULL = 54
Public Const LS_SORTER_2_POCKET_2_FULL = 55
Public Const LS_SORTER_2_POCKET_3_FULL = 56
Public Const LS_SORTER_3_POCKET_1_FULL = 57
Public Const LS_SORTER_3_POCKET_2_FULL = 58
Public Const LS_SORTER_3_POCKET_3_FULL = 59
Public Const LS_SORTER_4_POCKET_1_FULL = 60
Public Const LS_SORTER_4_POCKET_2_FULL = 61
Public Const LS_SORTER_4_POCKET_3_FULL = 62
Public Const LS_SORTER_5_POCKET_1_FULL = 63
Public Const LS_SORTER_5_POCKET_2_FULL = 64
Public Const LS_SORTER_5_POCKET_3_FULL = 65




'// -----------------------------------------------------------------------
'//                  DEFINES STRUTTURES
'// -----------------------------------------------------------------------

Public Type TReadOption
    PutBlanks As Long
    TypeRead As Byte
End Type 'READOPTIONS, *LPREADOPTIONS;

Public Type TDATAOPTICALWINDOW
    TypeRead As Byte            '// Type of read choise
    Reserved As Byte            '// Reserved
    XRightBottom As Integer     '// X1 coordinates
    YRightBottom As Integer     '// Y1 coordinates
    Size As Integer             '// size
    Height As Integer           '// height
End Type 'DATAOPTICALWINDOW, *PDATAOPTICALWINDOW;

Public Type TDATASORTERSELECT
    TypeCriteria As Byte                    '// Type of criteria choise
    CharToStart As Byte                     '// Char to start check in the codeline
    NrCharCheck As Byte                     '// Lenght string 1 and/or string 2
    String1(1 To MAX_CHAR_CHECK) As Byte    '// String 1
    String2(1 To MAX_CHAR_CHECK) As Byte    '// String 2
    bin As Byte                             '// Bin where put the document
End Type 'DATASORTERSELECT, *PDATASORTERSELECT;

Public Type LS_DATASORTERSELECT

    ' struct defined in this way to make it contiguous

    TypeCriteria1 As Byte
    CharToStart1 As Byte
    NrCharCheck1 As Byte
    FristString1 As String * 10
    SecondString1 As String * 10
    Bin1 As Byte
    TypeCriteria2 As Byte
    CharToStart2 As Byte
    NrCharCheck2 As Byte
    FristString2 As String * 10
    SecondString2 As String * 10
    Bin2 As Byte
    TypeCriteria3 As Byte
    CharToStart3 As Byte
    NrCharCheck3 As Byte
    FristString3 As String * 10
    SecondString3 As String * 10
    Bin3 As Byte
    TypeCriteria4 As Byte
    CharToStart4 As Byte
    NrCharCheck4 As Byte
    FristString4 As String * 10
    SecondString4 As String * 10
    Bin4 As Byte
    TypeCriteria5 As Byte
    CharToStart5 As Byte
    NrCharCheck5 As Byte
    FristString5 As String * 10
    SecondString5 As String * 10
    Bin5 As Byte
End Type

'// structure for read usefull information about the just stored image
Public Type THistory
    Size As Integer                '// Size of the struct
    doc_sorted As Long             '// Document sortered
    bourrage_feeder As Long        '// Jam in the feeder
    bourrage_micr As Long          '// Jam during the MICR reading
    doc_retain As Long             '// Nr. of document retained
    bourrage_exit As Long          '// Jam after the film
    doc_cmc7_err As Long           '// Nr. of document CMC7, read with error
    doc_e13b_err As Long           '// Nr. of document E13B, read with error
    time_peripheral_on As Long     '// Peripheral time life
    num_turn_on As Long            '// Nr. of power on
    doc_ink_jet As Long            '// Nr. of document printed
    doc_stamp As Long              '// Nr. of document stamped

End Type 'S_HISTORY, *LPS_HISTORY;

'// structures for LS800 call backs
Public Type TCODELINE_INFO
    '// Parameter filled by LsApi
    Size As Integer                          '// Size of the struct
    NrDoc As Long                            '// Progessive document number
    CodelineRead As String * CODE_LINE_LENGTH '// Codeline returned
    NrBytes As Integer                       '// Length of the codeline
    Reserved As Long                         '// Reserved for future use

    '// Parameter filled by Application
    Sorter As Integer                        '// Sorter where put the document
    FormatString1 As Byte                    '// Set from application NORMAL or BOLD
    StringToPrint1 As String * 80            '// String line 1 to print rear of the document
    FormatString2 As Byte                    '// Set from application NORMAL or BOLD
    StringToPrint2 As String * 80            '// String line 2 to print rear of the document
    FormatString3 As Byte                    '// Set from application NORMAL or BOLD
    StringToPrint3 As String * 80            '// String line 3 to print rear of the document
    FormatString4 As Byte                    '// Set from application NORMAL or BOLD
    StringToPrint4 As String * 80            '// String line 4 to print rear of the document
End Type


Public Type TIMAGE_INFO
    Size As Integer                          '// Size of the struct
    NrDoc As Long                            '// Progessive document number
    hImage As Long                           '// Image handle
    ImageSize As Long                        '// Image size bytes
```

```
   Width  As Long                    '// Image width
   Height As Long                    '// Image height
   Resolution As Long                  '// Image resolution
   BitCount As Long                   '// Image bit count (level of grey)
   CodelineRead As String * CODE_LINE_LENGTH '// Codeline returned
   NrBytes As Integer                 '// Length of the codeline
   Reserved As Long                    '// Reserved for future use

   '// Parameter filled by  Application
   Sorter As Integer                '// Sorter where put the document
   FormatString1 As Integer             '// Set from application NORMAL or BOLD
   StringToPrint1 As String * 80      '// String line 1 to print rear of the document
   FormatString2 As Integer             '// Set from application NORMAL or BOLD
   StringToPrint2 As String * 80      '// String line 2 to print rear of the document
   FormatString3 As Integer             '// Set from application NORMAL or BOLD
   StringToPrint3 As String * 80      '// String line 3 to print rear of the document
   FormatString4 As Integer             '// Set from application NORMAL or BOLD
   StringToPrint4 As String * 80      '// String line 4 to print rear of the document
End Type

Public Type TLS500CODELINE_INFO
   '// Parameter compiled by LsApi.dll
   Size As Integer                  '// Size of the struct
   NrDoc As Long                      '// Progessive document number
   CodelineRead As String * CODE_LINE_LENGTH '// Codeline returned
   NrBytes As Integer                 '// Length of the codeline
   Reserved As Long                    '// Reserved for future use

   '// Parameter compiled by Application
   Sorter As Integer                     '// Sorter where put the document
   FormatString As Integer               '// Set from application NORMAL or BOLD
   StringToPrint As String * 80          '// String line 1 to print rear of the document

End Type

'// ----------------------------------------------------------------------
'//                EXPORT FUNCTIONS
'// ----------------------------------------------------------------------
Public Declare Function LSConnect Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal hInst As Long, _
   ByVal iPeripheral As Integer, _
   ByRef lphConnect As Integer) As Long

Public Declare Function LSDisconnect Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long) As Long

Public Declare Function LSIdentify Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByRef bIdent As Byte, _
   ByVal sVendorModel As String, _
   ByVal sProductVersion As String, _
   ByVal sDecoderExpVersion As String, _
   ByVal sInkJetVersion As String, _
   ByVal reserved1 As String, _
   ByVal reserved2 As String) As Long

Public Declare Function LSDocHandle Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal iStamp As Integer, _
   ByVal iValidate As Integer, _
   ByVal iCodeline As Integer, _
   ByVal bSide As Byte, _
   ByVal iScanMode As Integer, _
   ByVal iFeeder As Integer, _
   ByVal iSorter As Integer, _
   ByVal iWaitTimeout As Integer, _
   ByVal iBeep As Integer, _
   ByRef lpNrDoc As Long, _
   ByVal iReserved1 As Integer, _
   ByVal lReserved2 As Long) As Long

Public Declare Function LSSetSorterCriteria Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByRef pCriteria As LS_DATASORTERSELECT, _
   ByVal NrCriteria As Integer) As Long

Public Declare Function LSReadImage Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal iClearBlack As Integer, _
   ByVal bSide As Byte, _
   ByVal iReadMode As Integer, _
   ByVal lNrDoc As Long, _
   ByRef lphFrontImage As Long, _
   ByRef lphBackImage As Long, _
   ByRef lphReserved1 As Long, _
   ByRef lphReserved2 As Long) As Long

Public Declare Function LSSaveDIB Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal hImage As Long, _
   ByVal sFilename As String) As Long

Public Declare Function LSSetOpticalWindows Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByRef pDimWindows As TDATAOPTICALWINDOW, _
   ByVal NrWindows As Integer) As Long

Public Declare Function LSReadCodeline Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal sCodeline As String, _
   ByRef lpiLength_Codeline As Integer, _
   ByVal sBarcode As String, _
   ByRef lpiLength_Barcode As Integer, _
   ByVal sOptic As String, _
   ByRef lpiLength_Optic As Integer) As Long

Public Declare Function LSCodelineReadFromBitmap Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal hImage As Long, _
   ByRef lpbCodelineType As Byte, _
   ByVal iUnitMeasure As Integer, _
   ByVal x As Single, _
   ByVal y As Single, _
   ByVal sizeW As Single, _
   ByVal sizeH As Single, _
   ByRef lpOption As TReadOption, _
   ByVal sCodeline As String, _
   ByRef lpLength As Long) As Long

Public Declare Function LSReadPdf417FromBitmap Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal hImage As Long, _
   ByVal sCodeline As String, _
   ByRef lpLength As Long, _
   ByRef lpbErrorRate As Byte, _
   ByVal lReserved1 As Long, _
   ByVal lReserved2 As Long, _
   ByVal lReserved3 As Long, _
   ByVal lReserved4 As Long) As Long

Public Declare Function LSReadBarcodeFromBitmap Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal hImage As Long, _
   ByVal bTypeBarcode As Byte, _
   ByVal lPos_x As Long, _
   ByVal lPos_y As Long, _
   ByVal lSizeW As Long, _
   ByVal lSizeH As Long, _
   ByVal sCodeline As String, _
   ByRef lpLength As Long) As Long

Public Declare Function LSConvertImageToBW Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal iMethod As Integer, _
   ByVal hGrayImage As Long, _
   ByRef lphBWImage As Long, _
   ByVal iPoloFilter As Integer, _
   ByVal sngReserved As Single) As Long

Public Declare Function LSFreeImage Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByRef lphImage As Long) As Long

Public Declare Function LSLoadString Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal bFormat As Byte, _
   ByVal iLength As Integer, _
   ByVal sString As String) As Long

Public Declare Function LSReset Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal bReserved As Byte) As Long

Public Declare Function LSAutoDocHandle Lib "LsApi.dll" (ByVal hConnect As Integer, ByVal HWND As
Long, _
   ByVal iStamp As Integer, _
   ByVal iValidate As Integer, _
   ByVal iCodeline As Integer, _
   ByVal iScanMode As Integer, _
   ByVal iFeeder As Integer, _
   ByVal iSorter As Integer, _
   ByVal iNumDocument As Integer, _
   ByVal iClearBlack As Integer, _
   ByVal bSide As Byte, _
   ByVal iReadMode As Integer, _
   ByVal iSaveImage As Integer, _
   ByVal sDirectoryFile As String, _
   ByVal sBaseFilename As String, _
   ByVal pos_x As Single, ByVal pos_y As Single, _
   ByVal sizeW As Single, ByVal sizeH As Single, _
   ByVal iOriginMeasureDoc As Integer, _
   ByVal iOcrImageSide As Integer, _
   ByVal iFileFormat As Integer, _
   ByVal lQuality As Long, _
   ByVal lSaveMode As Long, _
   ByVal lPageNumber As Long, _
   ByVal iWaitTimeout As Integer, _
   ByVal iBeep As Integer, _
   ByVal lpfunc1 As Long, ByVal lReserved2 As Long, _
   ByVal lReserved3 As Long) As Long

Public Declare Function LSAutoDocHandleVB Lib "LsApi.dll" (ByVal hConnect As Integer,
   ByVal HWND As Long, _
   ByVal iStamp As Integer, _
   ByVal iValidate As Integer, _
   ByVal iCodeline As Integer, _
   ByVal iScanMode As Integer, _
   ByVal iFeeder As Integer, _
   ByVal iSorter As Integer, _
   ByVal iNumDocument As Integer, _
   ByVal iClearBlack As Integer, _
   ByVal bSide As Byte, _
   ByVal iReadMode As Integer, _
   ByVal iSaveImage As Integer, _
   ByVal sDirectoryFile As String, _
   ByVal sBaseFilename As String, _
   ByVal pos_x As Single, ByVal pos_y As Single,  _
   ByVal sizeW As Single, ByVal sizeH As Single, _
   ByVal iOriginMeasureDoc As Integer, _
   ByVal iOcrImageSide As Integer, _
   ByVal iFileFormat As Integer, _
   ByVal lQuality As Long, _
   ByVal lSaveMode As Long, _
   ByVal lPageNumber As Long, _
   ByVal iWaitTimeout As Integer, _
   ByVal iBeep As Integer, _
   ByVal lpfunc1 As Long, ByVal lReserved2 As Long, _
   ByVal lReserved3 As Long) As Long

Public Declare Function LS800AutoDocHandleVB Lib "LsApi.dll" (ByVal hConnect As Integer, _
   ByVal HWND As Long, ByVal iValidate As Byte, _
   ByVal iCodeline As Integer, ByVal bSide As Byte, _
   ByVal iScanModeFront As Integer, ByVal iScanModeBack As Integer, _
   ByVal iClearBlack As Integer, _
   ByVal iNumDocument As Integer, _
   ByVal iSaveImage As Integer, _
   ByVal sDirectoryFile As String, _
   ByVal sBaseFilename As String, _
   ByVal unit As Integer, _
   ByVal pos_x As Single, ByVal pos_y As Single,  _
   ByVal sizeW As Single, ByVal sizeH As Single, _
   ByVal iOcrImageSide As Integer, _
   ByVal iFileFormat As Integer, _
   ByVal lQuality As Long, _
   ByVal lSaveMode As Long, _
   ByVal lPageNumber As Long, _
   ByVal iBeep As Integer, _
   ByVal lpfunc1 As Long, ByVal lpfunc2 As Long, ByVal lpfunc3 As Long, _
   ByVal lReserved1 As Integer, ByVal lReserved2 As Long, _
   ByVal lReserved3 As Long) As Long

Public Declare Function LSGetDocData Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByRef lpNrDoc As Long, _
   ByVal sFilenameFront As String, _
   ByVal sFilenameBack As String, _
   ByVal lReserved1 As Long, _
   ByVal lReserved2 As Long, _
   ByRef lpFrontImage As Long, _
   ByRef lpBackImage As Long, _
   ByVal lReserved3 As Long, _
   ByVal lReserved4 As Long, _
   ByVal sCodelineSW As String, _
   ByVal sCodelineHw As String, _
   ByVal sBarcode As String, _
   ByVal sCodelineOptical As String, _
   ByRef lpiDocToRead As Integer, _
   ByRef lpErrorType As Long, _
   ByVal lReserved5 As Long, _
   ByVal lReserved6 As Long) As Long

Public Declare Function LSPeripheralStatus Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByRef lpbSenseKey As Byte, _
   ByRef lpbStatusByte As Byte) As Long

Public Declare Function LSDoubleLeafingSensibility Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal myType As Integer, _
   ByVal Value As Byte) As Long

Public Declare Function LSViewOCRRectangle Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal fView As Long) As Long

Public Declare Function LSRS232Configuration Lib "LsApi.dll" ( _
   ByVal hConnect As Integer, _
   ByVal HWND As Long, _
   ByVal Port As String, _
   ByVal BaudeRate As Long, _
   ByVal Parity As Byte, _
   ByVal ByteSize As Byte, _
   ByVal BitsStop As Byte) As Long

Public Declare Function LSConvertImage200To100dpi Lib "LsApi.dll" ( _
   ByVal HWND As Long, _
   ByVal h As Long, _
```

```
ByRef h1 As Long) As Long
```

14.4. Sample code in C# (for .NET)

The source is available on demand :

CtsLsClass.cs

```
using System;
using System.Runtime.InteropServices;

namespace Ls_Test
{
    class CtsLs
    {
        [DllImport("LsApi.dll")]
        public static extern int LSConnect(int hWnd, int hInst, short
Peripheral, ref short hConnect);
        [DllImport("LsApi.dll")]
        public static extern int LSDisconnect(short hConnect, int hWnd);
        [DllImport("LsApi.dll")]
        public static extern int LSUnitIdentify(short hConnect, int hWnd, byte[]
pLsCfg, IntPtr LsModel, IntPtr FwVersion, IntPtr FwDate, IntPtr PeripheralID,
IntPtr BoardVersion, IntPtr DecoderExpVersion, IntPtr InkJetVersion, IntPtr
FeederVersion, IntPtr SorterVersion, IntPtr MotorVersion, IntPtr Reserved1,
IntPtr Reserved2);
        [DllImport("LsApi.dll")]
        public static extern int LSUnitStatus(short hConnect, int hWnd, ref
UNITSTATUS lpStatus);
        [DllImport("LsApi.dll")]
        public static extern int LSReset(short hConnect, int hWnd, short
ResetType);
        [DllImport("LsApi.dll")]
        public static extern int LSLoadStringWithCounterEx(short hConnect, int
hWnd, short PrintType, IntPtr strEndorse, short LenEndorse, UInt32 StartNumber,
short Step);
        [DllImport("LsApi.dll")]
        public static extern int LSLoadString(short hConnect, int hWnd, short
PrintType, short LenEndorse, IntPtr strEndorse);
        [DllImport("LsApi.dll")]
        public static extern int LSConfigDoubleLeafingAndDocLength(short
hConnect, int hWnd, Int32 Type, short Value, Int32 DocMin, Int32 DocMax);
        [DllImport("LsApi.dll")]
        public static extern int LSChangeStampPosition(short hConnect, int hWnd,
short Step, byte Reserved);
        [DllImport("LsApi.dll")]
        public static extern int LSDisableWaitDocument(short hConnect, int hWnd,
bool fWait);
        [DllImport("LsApi.dll")]
        public static extern int LSSetUnitSpeed(short hConnect, int hWnd, short
UnitSpeed);
        [DllImport("LsApi.dll")]
        public static extern int LSSetLightIntensity(short hConnect, int hWnd,
short UnitSpeed);
        [DllImport("LsApi.dll")]
        public static extern int LSModifyPWMUltraViolet(short hConnect, int
hWnd, short UnitSpeed, bool HighContrast, short Reserved);
        [DllImport("LsApi.dll")]
        public static extern int LSAutoDocHandle(short hConnect, int hWnd, short
Stamp, short Validate, short CodeLine, short ScanMode, short Feeder, short
Sorter, short NumDocument, short ClearBlack, char Side, short ReadMode, short
SaveImage, IntPtr DirectoryFile, IntPtr BaseFilename, Single pos_x, Single
pos_y, Single sizeW, Single sizeH, short OriginMeasureDoc, short OcrImageSide,
short FileFormat, int Quality, int SaveMode, int PageNumber, short WaitTimeout,
short Beep, int Reserved1, IntPtr Reserved2, IntPtr Reserved3);
        [DllImport("LsApi.dll")]
        public static extern int LSGetDocData(short hConnect, int hWnd, ref
UInt32 NrDoc, IntPtr FilenameFront, IntPtr FilenameBack, IntPtr Reserved1,
IntPtr Reserved2, ref IntPtr FrontImage, ref IntPtr Backmage, ref IntPtr
Reserved3, ref IntPtr Reserved4, IntPtr CodelineSW, IntPtr CodelineHW, IntPtr
Barcode, IntPtr CodelinesOptical, ref short DocToRead, ref Int32 NrPrinted,
IntPtr Reserved5, IntPtr Reserved6);
        [DllImport("LsApi.dll")]
        public static extern int LSDocHandle(short hConnect, int hWnd, short
Stamp, short Validate, short CodeLine, char Side, short ScanMode, short Feeder,
short Sorter, short WaitTimeout, short Beep, ref UInt32 NrDoc, Int16
ScanDocType, Int32 Reserved);
        [DllImport("LsApi.dll")]
        public static extern int LSReadCodeline(short hConnect, int hWnd, IntPtr
CodelineHW, ref short LenCodelineHW, IntPtr Barcode, ref short LenBarcode,
IntPtr CodelinesOptical, ref short LenOptic);
        [DllImport("LsApi.dll")]
        public static extern int LSReadImage(short hConnect, int hWnd, short
ClearBlack, char Side, short ReadMode, UInt32 NrDoc, ref IntPtr FrontImage, ref
IntPtr RearImage, ref IntPtr Reserved1, IntPtr Reserved2);
        [DllImport("LsApi.dll")]
        public static extern int LSCodelineReadFromBitmap(int hWnd, IntPtr
hImage, byte[] CodelineType, short UintMeasure, float Pos_x, float Pos_y, float
Width, float Height, ref READOPTIONS ro, IntPtr Codeline, ref int
Length_Codeline);
        [DllImport("LsApi.dll")]
        public static extern int LSReadBarcodeFromBitmap(int hWnd, IntPtr
hImage, byte BarcodeType, float Pos_x, float Pos_y, float Width, float Height,
IntPtr Codeline, ref int Length_Codeline);
        [DllImport("LsApi.dll")]
        public static extern int LSReadPdf417FromBitmap(int hWnd, IntPtr hImage,
IntPtr Codeline, ref int Length_Codeline, byte Reserved, float Pos_x, float
Pos_y, float Width, float Height);
        [DllImport("LsApi.dll")]
        public static extern int LSMergeImageGrayAndUV(int hWnd, IntPtr
hFrontGrayImage, IntPtr hFrontUVImage, float Reserved, float Reserved2, ref
IntPtr hGrayUVImage);
        [DllImport("LsApi.dll")]
        public static extern int LSFreeImage(int hWnd, ref IntPtr hImage);

        public struct BITMAPINFOHEADER
        {
            public UInt32 biSize;
            public Int32 biWidth;
            public Int32 biHeight;
            public Int16 biPlanes;
            public Int16 biBitCount;
            public UInt32 biCompression;
            public UInt32 biSizeImage;
            public Int32 biXPelsPerMeter;
            public Int32 biYPelsPerMeter;
            public UInt32 biClrUsed;
            public UInt32 biClrImportant;
        };
```

```
public struct UNITSTATUS
{
    public int Size;                             // Size of the structure

    public int UnitStatus;             // Ls40 Ls100 Ls150 Ls5xx Ls800
    public bool Photo_Feeder;          // Ls40 Ls100 Ls150 Ls5xx Ls800
    public bool Photo_Sorter;          //      Ls100
    public bool Photo_MICR;            //      Ls100 Ls150 Ls5xx
    public bool Photo_Path_Ls100;      //      Ls100
    public bool Photo_Scanners;        //      Ls100
    public bool Unit_Just_ON;          // Ls40 Ls100 Ls150
    public bool Photo_Double_Leafing_Down;//   Ls100 Ls150
    public bool Photo_Double_Leafing_Middle;//       Ls150
    public bool Photo_Double_Leafing_Up;//      Ls100 Ls150
    public bool Photo_Card;            //            Ls150
    public bool Pockets_All_Full;      //            Ls150 Ls5xx
    public bool Photo_Stamp;           //                  Ls5xx
    public bool Photo_Exit;            //                  Ls5xx
    public bool Pocket_1_Full;         //                  Ls5xx
    public bool Pocket_2_Full;         //                  Ls5xx

    public bool Photo_Path_Feeder;     //                        Ls800
    public bool Photo_Path_Module_Begin;//                        Ls800
    public bool Photo_Path_Binary_Rigth;//                        Ls800
    public bool Photo_Path_Binary_Left;//                        Ls800
    public bool Photo_Path_Module_End; //                        Ls800
    public bool Sorter_1_input_pocket_1;//                        Ls800
    public bool Sorter_1_pocket_1_full;//                        Ls800
    public bool Sorter_1_input_pocket_2;//                        Ls800
    public bool Sorter_1_pocket_2_full;//                        Ls800
    public bool Sorter_1_input_pocket_3;//                        Ls800
    public bool Sorter_1_pocket_3_full;//                        Ls800
    public bool Sorter_2_input_pocket_1;//                        Ls800
    public bool Sorter_2_pocket_1_full;//                        Ls800
    public bool Sorter_2_input_pocket_2;//                        Ls800
    public bool Sorter_2_pocket_2_full;//                        Ls800
    public bool Sorter_2_input_pocket_3;//                        Ls800
    public bool Sorter_2_pocket_3_full;//                        Ls800
    public bool Sorter_3_input_pocket_1;//                        Ls800
    public bool Sorter_3_pocket_1_full;//                        Ls800
    public bool Sorter_3_input_pocket_2;//                        Ls800
    public bool Sorter_3_pocket_2_full;//                        Ls800
    public bool Sorter_3_input_pocket_3;//                        Ls800
    public bool Sorter_3_pocket_3_full;//                        Ls800
    public bool Sorter_4_input_pocket_1;//                        Ls800
    public bool Sorter_4_pocket_1_full;//                        Ls800
    public bool Sorter_4_input_pocket_2;//                        Ls800
    public bool Sorter_4_pocket_2_full;//                        Ls800
    public bool Sorter_4_input_pocket_3;//                        Ls800
    public bool Sorter_4_pocket_3_full;//                        Ls800
    public bool Sorter_5_input_pocket_1;//                        Ls800
    public bool Sorter_5_pocket_1_full;//                        Ls800
    public bool Sorter_5_input_pocket_2;//                        Ls800
    public bool Sorter_5_pocket_2_full;//                        Ls800
    public bool Sorter_5_input_pocket_3;//                        Ls800
    public bool Sorter_5_pocket_3_full;//                        Ls800
    public bool Sorter_6_input_pocket_1;//                        Ls800
    public bool Sorter_6_pocket_1_full;//                        Ls800
    public bool Sorter_6_input_pocket_2;//                        Ls800
```

```
            public bool Sorter_6_pocket_2_full;//                        Ls800
            public bool Sorter_6_input_pocket_3;//                        Ls800
            public bool Sorter_6_pocket_3_full;//                        Ls800
            public bool Sorter_7_input_pocket_1;//                        Ls800
            public bool Sorter_7_pocket_1_full;//                        Ls800
            public bool Sorter_7_input_pocket_2;//                        Ls800
            public bool Sorter_7_pocket_2_full;//                        Ls800
            public bool Sorter_7_input_pocket_3;//                        Ls800
            public bool Sorter_7_pocket_3_full;//                        Ls800

            public bool Photo_Trigger;         // Ls40
            public bool Document_Retained;     // Ls40
        };

        public struct READOPTIONS
        {
            public int PutBlanks;     // 0 = CodeLIne whitout blans, 1 =
CodeLine with 1 blanks
            public char TypeRead;     // 'N' for 1 type of CodeLine, 'X' for
CodeLine E13B switch OCRB
        };


        // Parameter Peripheral Type
        public enum LsUnitType : short
            {
            LS_40_LSCONNECT = 39,
            LS_40_USB = 40,
            LS_100_LSCONNECT = 109,
            LS_100_USB = 100,
            LS_100_RS232 = 101,
            LS_100_ETH = 110,
            LS_150_LSCONNECT = 149,
            LS_150_USB = 150,
            LS_200_USB = 201,
            LS_5xx_SCSI = 500,
            LS_515_LSCONNECT = 501,
            LS_515_USB = 502,
            LS_520_USB = 520,
            LS_800_USB = 801,
        };

        public enum Stamp : short
            {
                  STAMP_NO = 0,                      // No stamp is done
                  STAMP_FRONT = 1,             // Stamp on front document
                  STAMP_BACK = 2,                    // Stamp on rear document
                  STAMP_FRONT_AND_BACK = 3,    // Stamp front and rear document
            };

        public enum PrintValidate : short
            {
                  NO_PRINT_VALIDATE = 0,             // No print is done
                  PRINT_VALIDATE = 1,                    // Print done
            PRINT_LOGO = 4,                  // Print a logo only
                  PRINT_VALIDATE_WITH_LOGO = 5, // Print logo and lines
            };

        public enum Feeder : short
            {
                  FEED_AUTO = 0,                     // Start Document from
Feeder
                  FEED_FROM_PATH = 1,                // Start Document from Unit
Path
            };

        public enum Sorter : short
            {
                  SORTER_DOC_HOLDED = 0,
                  SORTER_POCKET_1 = 1,
                  SORTER_POCKET_2 = 2,
                  SORTER_AUTOMATIC = 3,
                  SORTER_SWICTH_1_TO_2 = 4,
                  SORTER_DOC_EJECTED = 5,
                  SORTER_ON_CODELINE_CALLBACK = 6,

                  // For Ls800 unit
                  SORTER_CIRCULAR = 48,
                  SORTER_SEQUENTIAL = 49,
                  SORTER_POCKET_0_SELECTED = 50,
                  SORTER_POCKET_1_SELECTED = 51,
                  SORTER_POCKET_2_SELECTED = 52,
                  SORTER_POCKET_3_SELECTED = 53,
                  SORTER_POCKET_4_SELECTED = 54,
                  SORTER_POCKET_5_SELECTED = 55,
                  SORTER_POCKET_6_SELECTED = 56,
                  SORTER_POCKET_7_SELECTED = 57,
                  SORTER_POCKET_8_SELECTED = 58,
                  SORTER_POCKET_9_SELECTED = 59,
                  SORTER_POCKET_10_SELECTED = 60,
                  SORTER_POCKET_11_SELECTED = 61,
                  SORTER_POCKET_12_SELECTED = 62,
                  SORTER_POCKET_13_SELECTED = 63,
                  SORTER_POCKET_14_SELECTED = 64,
                  SORTER_POCKET_15_SELECTED = 65,
                  SORTER_POCKET_16_SELECTED = 66,
                  SORTER_POCKET_17_SELECTED = 67,
                  SORTER_POCKET_18_SELECTED = 68,
                  SORTER_POCKET_19_SELECTED = 69,
                  SORTER_POCKET_20_SELECTED = 70,
                  SORTER_POCKET_21_SELECTED = 71,
            };

        public enum CodeLineType : byte
            {
                  NO_READ_CODELINE = 0,
                  READ_CODELINE_HW_MICR = 1,
                  READ_CODELINE_E13B_MICR_WITH_OCR = 15,

                  READ_CODELINE_SW_OCRA = 65, //'A',
                  READ_CODELINE_SW_OCRB_NUM = 66, //'B',
                  READ_CODELINE_SW_OCRB_ALFANUM = 67, //'C',
                  READ_CODELINE_SW_OCRB_ITALY = 70, //'F',
                  READ_CODELINE_SW_E13B = 69, //'E',
                  READ_CODELINE_SW_E13B_X_OCRB = 88, //'X',

                  READ_BARCODE_2_OF_5 = 50,
                  READ_BARCODE_CODE39 = 51,
                  READ_BARCODE_CODE128 = 52,
                  READ_BARCODE_EAN13 = 53,
            MAX_CODE_LINE_LENGTH = 254,
            };

        public enum Unit : short
            {
                  UNIT_MM = 0,
                  UNIT_INCH = 1,
            };

            public class OcrHeight
            {
                public const double OCR_MAX_HEIGHT_IN_MM = 10.5;
            public const double OCR_MAX_HEIGHT_IN_INCH = 0.41;
            };

            public enum BlankInCodeline : short
            {
                  BLANK_IN_CODELINE_NO = 0,
                  BLANK_IN_CODELINE_YES = 1,
            };

            public enum OriginOCR : short
            {
                  ORIGIN_BOTTOM_RIGHT_MM = 10,
                  ORIGIN_BOTTOM_RIGHT_INCH = 20,
            };

            public enum ScanMode : short
            {
                  SCAN_MODE_BW = 1,
                  SCAN_MODE_16_GRAY_100 = 2,
                  SCAN_MODE_16_GRAY_200 = 3,
                  SCAN_MODE_256_GRAY_100 = 4,
                  SCAN_MODE_256_GRAY_200 = 5,
                  SCAN_MODE_COLOR_100 = 10,
                  SCAN_MODE_COLOR_200 = 11,
                  SCAN_MODE_16_GRAY_300 = 20,
                  SCAN_MODE_256_GRAY_300 = 21,
                  SCAN_MODE_COLOR_300     = 22,
                  SCAN_MODE_256GR100_AND_UV = 40,
                  SCAN_MODE_256GR200_AND_UV = 41,
                  SCAN_MODE_256GR300_AND_UV = 42,
            };

            public enum ScanDocType : short
            {
                  SCAN_PAPER_DOCUMENT = 0 ,
                  SCAN_CARD = 1 ,
            };

            public enum Side : short
            {
                  SIDE_NONE_IMAGE = 78, //'N',
                  SIDE_FRONT_IMAGE = 70, //'F',
                  SIDE_BACK_IMAGE = 66, //'B',
                  SIDE_ALL_IMAGE = 88, //'X',
                  SIDE_FRONT_UV = 85, //'U',
                  SIDE_FRONT_MERGED = 77, //'M',
            };
```

```
public enum Wait : short
{
        WAIT_NO = 71, //'G',
        WAIT_YES = 87, //'W',
};

public enum Beep : short
{
        BEEP_NO = 0,
        BEEP_YES = 1,
};

public enum ClearBlack : short
{
        CLEAR_BLACK_NO = 0,
        CLEAR_BLACK_YES = 1,
        CLEAR_AND_ALIGN_IMAGE = 2 ,
};

public enum PrintFont : byte
{
        PRINT_FORMAT_NORMAL = 78, //'N',
        PRINT_FORMAT_BOLD = 66, //'B',
        PRINT_FORMAT_NORMAL_15 = 65, //'A',
        PRINT_FORMAT_DOUBLE_HIGH = 68, //'D',

        PRINT_UP_FORMAT_NORMAL = 110, //'n',
        PRINT_UP_FORMAT_BOLD = 98, //'b',
        PRINT_UP_FORMAT_NORMAL_15_CHAR = 97, //'a',
};

public enum DoubleLeafing : short
{

        DOUBLE_LEAFING_WARNING = 0 ,
        DOUBLE_LEAFING_ERROR = 1 ,
        //DOUBLE_LEAFING_LEVEL1 = 1,non lo uso piu'
        DOUBLE_LEAFING_LEVEL2 = 2,
        DOUBLE_LEAFING_LEVEL3 = 3,
        DOUBLE_LEAFING_DEFAULT = 4,
        DOUBLE_LEAFING_LEVEL4 = 5,
        DOUBLE_LEAFING_LEVEL5 = 6,
        DOUBLE_LEAFING_DISABLE = 7,
};

public enum Reset : short
{
        RESET_ERROR = 48, //'0',
        RESET_PATH = 49, //'1',
        RESET_BELT_CLEANING     = 50, //'2',
};

public enum ImageSave : short
{
        IMAGE_SAVE_ON_FILE = 4,
        IMAGE_SAVE_HANDLE = 5,
        IMAGE_SAVE_BOTH = 6,
        IMAGE_SAVE_NONE = 7,
};
```

```
        public enum FileType : short
        {
              FILE_JPEG = 10,
              FILE_BMP = 11,
              FILE_TIF = 3,
              FILE_CCITT = 25,
              FILE_CCITT_GROUP3_1DIM = 27,
              FILE_CCITT_GROUP3_2DIM = 28,
              FILE_CCITT_GROUP4 = 29,
        };

        public enum  FileAttribute : short
        {
              SAVE_OVERWRITE = 0,
              SAVE_APPEND = 1,
              SAVE_REPLACE = 2,
              SAVE_INSERT = 3,
        };

        public enum Badge : short
        {
              BADGE_READ_TRACK_1 = 0x20,
              BADGE_READ_TRACK_2 = 0x40,
              BADGE_READ_TRACK_3 = 0x80,
              BADGE_READ_TRACKS_1_2 = 0x60,
              BADGE_READ_TRACKS_2_3 = 0xc0,
              BADGE_READ_TRACKS_1_2_3 = 0xe0,
        };

    public class LsReply
    {
        // --------------------------------------------------------------
        //                          REPLY-CODE
        // --------------------------------------------------------------
        public const int LS_OKAY = 0;


        // --------------------------------------------------------------
        //                  ERRORS
        // --------------------------------------------------------------
        public const int LS_SYSTEM_ERROR = -1;
        public const int LS_USB_ERROR = -2;
        public const int LS_PERIPHERAL_NOT_FOUND = -3;
        public const int LS_HARDWARE_ERROR = -4;
        public const int LS_PERIPHERAL_OFF_ON = -5;
        public const int LS_RESERVED_ERROR = -6;
        public const int LS_PAPER_JAM = -7;
        public const int LS_TARGET_BUSY = -8;
        public const int LS_INVALID_COMMAND = -9;
        public const int LS_DATA_LOST = -10;
        public const int LS_COMMAND_IN_EXECUTION_YET = -11;
        public const int LS_JPEG_ERROR = -12;
        public const int LS_COMMAND_SEQUENCE_ERROR = -13;
        public const int LS_PC_HW_ERROR = -14;
        public const int LS_IMAGE_OVERWRITE = -15;
        public const int LS_INVALID_HANDLE = -16;
        public const int LS_NO_LIBRARY_LOAD = -17;
        public const int LS_BMP_ERROR = -18;
        public const int LS_TIFF_ERROR = -19;
        public const int LS_IMAGE_NO_MORE_AVAILABLE = -20;
        public const int LS_IMAGE_NO_FILMED = -21;
        public const int LS_IMAGE_NOT_PRESENT = -22;
        public const int LS_FUNCTION_NOT_AVAILABLE = -23;
        public const int LS_DOCUMENT_NOT_SUPPORTED = -24;
        public const int LS_BARCODE_ERROR = -25;
        public const int LS_INVALID_LIBRARY = -26;
        public const int LS_INVALID_IMAGE = -27;
        public const int LS_INVALID_IMAGE_FORMAT = -28;
        public const int LS_INVALID_BARCODE_TYPE = -29;
        public const int LS_OPEN_NOT_DONE = -30;
        public const int LS_INVALID_TYPE_COMMAND = -31;
        public const int LS_INVALID_CLEARBLACK = -32;
        public const int LS_INVALID_SIDE = -33;
        public const int LS_MISSING_IMAGE = -34;
        public const int LS_INVALID_TYPE = -35;
        public const int LS_INVALID_SAVEMODE = -36;
        public const int LS_INVALID_PAGE_NUMBER = -37;
        public const int LS_INVALID_NRIMAGE = -38;
        public const int LS_INVALID_STAMP = -39;
        public const int LS_INVALID_WAITTIMEOUT = -40;
        public const int LS_INVALID_VALIDATE = -41;
        public const int LS_INVALID_CODELINE_TYPE = -42;
        public const int LS_MISSING_NRIMAGE = -43;
        public const int LS_INVALID_SCANMODE = -44;
        public const int LS_INVALID_BEEP = -45;
        public const int LS_INVALID_FEEDER = -46;
        public const int LS_INVALID_SORTER = -47;
        public const int LS_INVALID_BADGE_TRACK = -48;
        public const int LS_MISSING_FILENAME = -49;
        public const int LS_INVALID_QUALITY = -50;
        public const int LS_INVALID_FILEFORMAT = -51;
        public const int LS_INVALID_COORDINATE = -52;
        public const int LS_MISSING_HANDLE_VARIABLE = -53;
        public const int LS_INVALID_POLO_FILTER = -54;
        public const int LS_INVALID_ORIGIN_MEASURES = -55;
        public const int LS_INVALID_SIZEH_VALUE = -56;
        public const int LS_INVALID_FORMAT = -57;
        public const int LS_STRINGS_TOO_LONGS = -58;
        public const int LS_READ_IMAGE_FAILED = -59;
        public const int LS_INVALID_CMD_HISTORY = -60;
        public const int LS_MISSING_BUFFER_HISTORY = -61;
        public const int LS_INVALID_ANSWER = -62;
        public const int LS_OPEN_FILE_ERROR_OR_NOT_FOUND = -63;
        public const int LS_READ_TIMEOUT_EXPIRED = -64;
        public const int LS_INVALID_METHOD = -65;
        public const int LS_CALIBRATION_FAILED = -66;
        public const int LS_INVALID_SAVEIMAGE = -67;
        public const int LS_INVALID_UNIT = -68;
        public const int LS_INVALID_NRWINDOWS = -71;
        public const int LS_INVALID_VALUE = -72;
        public const int LS_ILLEGAL_REQUEST = -73;
        public const int LS_INVALID_NR_CRITERIA = -74;
        public const int LS_MISSING_CRITERIA_STRUCTURE = -75;
        public const int LS_INVALID_MOVEMENT = -76;
        public const int LS_INVALID_DEGREE = -77;
        public const int LS_R0TATE_ERROR = -78;
        public const int LS_MICR_VALUE_OUT_OF_RANGE = -79;
        public const int LS_PERIPHERAL_RESERVED = -80;
        public const int LS_INVALID_NCHANGE = -81;
```

```
public const int LS_BRIGHTNESS_ERROR = -82;
public const int LS_CONTRAST_ERROR = -83;
public const int LS_INVALID_SIDETOPRINT = -84;
public const int LS_DOUBLE_LEAFING_ERROR = -85;
public const int LS_INVALID_BADGE_TIMEOUT = -86;
public const int LS_INVALID_RESET_TYPE = -87;
public const int LS_MISSING_SET_CALLBACK = -88;
public const int LS_IMAGE_NOT_200_DPI = -89;
public const int LS_DOWNLOAD_ERROR = -90;
public const int LS_INVALID_SORT_ON_CHOICE = -91;
public const int LS_INVALID_FONT = -92;
public const int LS_INVALID_UNIT_SPEED = -93;
public const int LS_INVALID_LENGTH = -94;
public const int LS_SHORT_PAPER = -95;
public const int LS_INVALID_DOC_LENGTH = -96;
public const int LS_INVALID_DOCSLONG = -97;
public const int LS_IMAGE_NOT_256_COLOR = -98;
public const int LS_BATTERY_NOT_CHARGED = -99;
public const int LS_INVALID_SCAN_DOC_TYPE = -100;
public const int LS_ILLEGAL_SCAN_CARD_SPEED = -101;
public const int LS_INVALID_PWM_VALUE = -102;
public const int LS_INVALID_KEY_LENGTH = -103;
public const int LS_INVALID_PASSWORD = -104;
public const int LS_UNIT_LOCKED = -105;
public const int LS_INVALID_IMAGEFORMAT = -106;
public const int LS_INVALID_THRESHOLD = -107;
public const int LS_NO_START_FOR_SORTER_FULL = -108;
public const int LS_IPBOX_ADDRESS_NOT_FOUNDED = -109;
public const int LS_INVALID_LED_COMMAND = -110;
public const int LS_INVALID_COLOR_PARAMETER = -111;

public const int LS_JAM_AT_MICR_PHOTO = -201;
public const int LS_JAM_DOC_TOO_LONG = -202;
public const int LS_JAM_AT_SCANNER_PHOTO = -203;

public const int LS_SCAN_NETTO_IMAGE_NOT_SUPPORTED = -521;
public const int LS_256_GRAY_NOT_SUPPORTED = -522;
public const int LS_INVALID_PATH = -523;
public const int LS_MISSING_CALLBACK_FUNCTION = -526;
public const int LS_INVALID_OCR_IMAGE_SIDE = -558;
public const int LS_PERIPHERAL_NOT_ANSWER = -599;

public const int LS_INVALID_CONNECTION_HANDLE = -1000;
public const int LS_INVALID_CONNECT_PERIPHERAL = -1001;
public const int LS_PERIPHERAL_NOT_YET_INTEGRATE = -1002;
public const int LS_UNKNOW_PERIPHERAL_REPLY = -1003;
public const int LS_CODELINE_ALREADY_DEFINED = -1004;
public const int LS_INVALID_NUMBER_OF_DOC = -1005;

public const int LS_DECODE_FONT_NOT_PRESENT = -1101;
public const int LS_DECODE_INVALID_COORDINATE = -1102;
public const int LS_DECODE_INVALID_OPTION = -1103;
public const int LS_DECODE_INVALID_CODELINE_TYPE = -1104;
public const int LS_DECODE_SYSTEM_ERROR = -1105;
public const int LS_DECODE_DATA_TRUNC = -1106;
public const int LS_DECODE_INVALID_BITMAP = -1107;
public const int LS_DECODE_ILLEGAL_USE = -1108;

public const int LS_BARCODE_GENERIC_ERROR = -1201;
public const int LS_BARCODE_NOT_DECODABLE = -1202;
public const int LS_BARCODE_OPENFILE_ERROR = -1203;
public const int LS_BARCODE_READBMP_ERROR = -1204;
public const int LS_BARCODE_MEMORY_ERROR = -1205;
public const int LS_BARCODE_START_NOTFOUND = -1206;
public const int LS_BARCODE_STOP_NOTFOUND = -1207;

public const int LS_PDF_NOT_DECODABLE = -1301;
public const int LS_PDF_READBMP_ERROR = -1302;
public const int LS_PDF_BITMAP_FORMAT_ERROR = -1303;
public const int LS_PDF_MEMORY_ERROR = -1304;
public const int LS_PDF_START_NOTFOUND = -1305;
public const int LS_PDF_STOP_NOTFOUND = -1306;
public const int LS_PDF_LEFTIND_ERROR = -1307;
public const int LS_PDF_RIGHTIND_ERROR = -1308;
public const int LS_PDF_OPENFILE_ERROR = -1309;


// --------------------------------------------------------------
//                  WARNINGS
// --------------------------------------------------------------
public const int LS_FEEDER_EMPTY = 1;
public const int LS_DATA_TRUNCATED = 2;
public const int LS_DOC_PRESENT = 3;
public const int LS_BADGE_TIMEOUT = 4;
public const int LS_ALREADY_OPEN = 5;
public const int LS_PERIPHERAL_BUSY = 6;
public const int LS_DOUBLE_LEAFING_WARNING = 7;
public const int LS_COMMAND_NOT_ENDED = 8;
public const int LS_RETRY = 9;
public const int LS_NO_OTHER_DOCUMENT = 10;
public const int LS_QUEUE_FULL = 11;
public const int LS_NO_SENSE = 12;
public const int LS_TRY_TO_RESET = 14;
public const int LS_STRING_TRUNCATED = 15;
public const int LS_COMMAND_NOT_SUPPORTED = 19;
public const int LS_SORTER1_FULL = 35;
public const int LS_SORTER2_FULL = 36;
public const int LS_SORTERS_BOTH_FULL = 37;
public const int LS_KEEP_DOC_ON_CODELINE_ERROR = 39;
public const int LS_LOOP_INTERRUPTED = 40;

public const int LS_SORTER_1_POCKET_1_FULL = 51;
public const int LS_SORTER_1_POCKET_2_FULL = 52;
public const int LS_SORTER_1_POCKET_3_FULL = 53;
public const int LS_SORTER_2_POCKET_1_FULL = 54;
public const int LS_SORTER_2_POCKET_2_FULL = 55;
public const int LS_SORTER_2_POCKET_3_FULL = 56;
public const int LS_SORTER_3_POCKET_1_FULL = 57;
public const int LS_SORTER_3_POCKET_2_FULL = 58;
public const int LS_SORTER_3_POCKET_3_FULL = 59;
public const int LS_SORTER_4_POCKET_1_FULL = 60;
public const int LS_SORTER_4_POCKET_2_FULL = 61;
public const int LS_SORTER_4_POCKET_3_FULL = 62;
public const int LS_SORTER_5_POCKET_1_FULL = 63;
public const int LS_SORTER_5_POCKET_2_FULL = 64;
public const int LS_SORTER_5_POCKET_3_FULL = 65;
public const int LS_SORTER_6_POCKET_1_FULL = 66;
public const int LS_SORTER_6_POCKET_2_FULL = 67;
public const int LS_SORTER_6_POCKET_3_FULL = 68;
public const int LS_SORTER_7_POCKET_1_FULL = 69;
```

```
            public const int LS_SORTER_7_POCKET_2_FULL = 70;
            public const int LS_SORTER_7_POCKET_3_FULL = 71;
        };
    }
}




private void identify(short model)
{
      byte[] UnitCfg = new byte[4];
      IntPtr strLsModel = Marshal.AllocHGlobal(20);
      IntPtr strFwVersion = Marshal.AllocHGlobal(20);
      IntPtr Date_Fw = Marshal.AllocHGlobal(20);
      IntPtr strUnitID = Marshal.AllocHGlobal(20);
      IntPtr strInkJetVersion = Marshal.AllocHGlobal(20);
      IntPtr DecoderExpVersion = Marshal.AllocHGlobal(20);
      string i, i1;
      byte a, b;

      ident.Text = "";
      Reply = LSConnect(0, 0, model, ref hConnect);
      if (Reply == CtsLs.LsReply.LS_OKAY)
      {
           String strIdentify;
           string blankOpt = "      ";

            ReplyCode.Text = "OK"; ReplyCode.Visible = true;

            Reply = CtsLs.LSUnitIdentify(hConnect, 0,
                                         UnitCfg,
                                         strLsModel,
                                         strFwVersion,
                                         Date_Fw,
                                         strUnitID,
                                         IntPtr.Zero, // BoardVersion
                                         DecoderExpVersion,
                                         strInkJetVersion,
                                         IntPtr.Zero, //FeederVersion,
                                         IntPtr.Zero, //SorterVersion,
                                         IntPtr.Zero, //Motorversion,
                                         IntPtr.Zero, //Reserved1,
                                         IntPtr.Zero);//Reserved2
            if (Reply == LS_OKAY)
            {
                switch (model)
                {
                case LS_100_USB:
   strIdentify = "FW version: " + Marshal.PtrToStringAnsi(strLsModel) + "\n"
               + "FW Date: " + Marshal.PtrToStringAnsi(strFwVersion) + "\n"
               + "Serial #: " + Marshal.PtrToStringAnsi(strSerialNumber) + "\n";

                     i1 = "OPTIONS:\n";
                     b = 0x1;
                     a = (byte) (LpldPnt[1] & b);
                     if (a == 0x1)
                         i1 = i1 + "Micr ";
                     b = 0x8;
                     a = (byte) (LpldPnt[1] & b);
                     if (a == 0x8)
                     {
                          i1 = i1 + "Ink-jet ";
                          inkJetPresent = true;
                     }
                     b = 0x10;
                     a = (byte)(LpldPnt[1] & b);
                     if (a == 0x10)
                          i1 = i1 + "Feeder ";
                     b = 0x20;
                     a = (byte)(LpldPnt[1] & b);
                     if (a == 0x20)
                          i1 = i1 + "Stamp ";

                     a = (byte)(LpldPnt[2] & 0x04);
                     if (a != 0)               // Badge present
                     {
                         a = (byte)(LpldPnt[2] & 0x10);
                         if (a != 0)
                              i1 = i1 + "Badge123";
                         a = (byte)(LpldPnt[2] & 0x08);
                         if (a != 0)
                              i1 = i1 + "Badge12";
                          else
                              i1 = i1 + "Badge23";
                     }
                     strLsModel += i1 ;
                     break;

                case LS_150_USB:
        strIdentify = "Model : " + Marshal.PtrToStringAnsi(strLsModel) + "\n" +
                "FW version : " + Marshal.PtrToStringAnsi(strFwVersion) + "\n" +
                      "FW date : " + Marshal.PtrToStringAnsi(Date_Fw) + "\n\n" +
                    "Serial #: " + Marshal.PtrToStringAnsi(strUnitID) + "\n\n" +
                     "Options :\n";

                    if ((UnitCfg[0] & 0x01) == 0x01)
                        strIdentify += blankOpt + "MICR reader\n";
                    if ((UnitCfg[0] & 0x02) == 0x02)
                        strIdentify += blankOpt + "Unit set in Normal Speed\n";
                    else
                        strIdentify += blankOpt + "Unit set in High Speed\n";
                    if ((UnitCfg[0] & 0x04) == 0x04)
                        strIdentify += blankOpt + "Feeder Motorized\n";
                    if ((UnitCfg[0] & 0x08) == 0x08)
                    {
                        if ((UnitCfg[2] & 0x08) == 0x08)
                            strIdentify += blankOpt + "High Definition Ink-jet
printer\n";
                        else
                            strIdentify += blankOpt + "Endorsement Ink-jet
printer\n";
                    }
                    if ((UnitCfg[0] & 0x10) == 0x10)
                        strIdentify += blankOpt + "Feeder with Electromagnet 50
Doc.\n";
                    if ((UnitCfg[0] & 0x20) == 0x20)
                        strIdentify += blankOpt + "Voiding front stamp\n";
                    if ((UnitCfg[1] & 0x04) == 0x04)
```

```
                        strIdentify += blankOpt + "Scanner FRONT with Ultra
Violet\n";
                    else if ((UnitCfg[1] & 0x01) == 0x01)
                        strIdentify += blankOpt + "Scanner FRONT\n";
                    if ((UnitCfg[1] & 0x02) == 0x02)
                        strIdentify += blankOpt + "Scanner REAR\n";
                    if ((UnitCfg[1] & 0x20) == 0x20)
                        strIdentify += blankOpt + "COLOR version\n";
                    if ((UnitCfg[1] & 0x08) == 0x08)
                    {
                        if ((UnitCfg[1] & 0x10) == 0x10)
                            strIdentify += blankOpt + "Badge reader with tracks
1/2/3\n";
                        else
                            strIdentify += blankOpt + "Badge reader with tracks
2/3\n";
                    }
                    else if ((UnitCfg[1] & 0x10) == 0x10)
                        strIdentify += blankOpt + "Badge reader with tracks
1/2\n";
                            break;

                case LS_515_USB:
      strIdentify = "FW version: " + Marshal.PtrToStringAnsi(strLsModel) + "\n"
                   + "FW Date: " + Marshal.PtrToStringAnsi(strFwVersion) + "\n"
               + "Serial #: " + Marshal.PtrToStringAnsi(strSerialNumber) + "\n";

                       i1 = "OPTIONS:\n";
                       b = 0x1;
                       a = (byte)(LpldPnt[1] & b);
                       if (a == 0x1)
                            i1 = i1 + "CMC7 ";
                       b = 0x2;
                       a = (byte)(LpldPnt[1] & b);
                       if (a == 0x2)
                           i1 = i1 + "E13B ";
                       b = 0x8;
                       a = (byte)(LpldPnt[1] & b);
                       if (a == 0x8)
                       {
                           i1 = i1 + "Ink-jet ";
                           inkJetPresent = true;
                       }
                       b = 0x10;
                       a = (byte)(LpldPnt[1] & b);
                       if (a == 0x10)
                           i1 = i1 + "BackStamp ";
                       b = 0x20;
                       a = (byte)(LpldPnt[1] & b);
                       if (a == 0x20)
                           i1 = i1 + "FrontStamp ";
                       a = (byte)(LpldPnt[2] & 0x10);
                       if (a == 0x10)                // Badge present
                           i1 = i1 + "Badge";

                       strIdentify += i1;
                       break;
                }
            }
            else
            {
                strIdentify = "Connect not possible: error " +
                                   ErrorCode.ToString());
            }

            MessageBox.Show(strIdentify, TITLE_POPUP);

            LSDisconnect(hConnect, 0);
        } // ok the Connect
        else
            MessageBox.Show("Connect not possible: error", TITLE_POPUP);
    }
}


public int DocumentHandle(short operation)
{
      int          Reply;
      //    string      FileOut;

      IntPtr  BufFrontFile = Marshal.AllocHGlobal(1024);
      IntPtr  BufRearFile = Marshal.AllocHGlobal(1024);
      IntPtr BufFrontImage;
      IntPtr BufRearImage;
      IntPtr BufFrontUVImage;
      IntPtr BufFrontGrayUVImage;
      IntPtr NoImage;
      IntPtr  BufCodelineSW =
Marshal.AllocHGlobal((int)CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH);
      IntPtr  BufCodelineHW =
Marshal.AllocHGlobal((int)CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH);
      IntPtr  BufBarcode =
Marshal.AllocHGlobal((int)CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH);
      CtsLs.CodeLineType CodelineType;
      float   C_x;
      float   C_y;
      float   C_w;
      float   C_h;

      Reply = LSConnect(0, 0, CURRENT_MODEL, ref hConnect);

      if (Reply != LS_OKAY)
      {
           // Error Handling
           MessageBox.Show("Open not done: error " + Reply.ToString());
           return Reply;
      }

      PrintValidate = CtsLs.PrintValidate.PRINT_VALIDATE;
      //-----------LoadString---------------------------------------
      if (stParAppl.PrintValidate == CtsLs.PrintValidate.PRINT_VALIDATE ||
          stParAppl.PrintValidate ==
(CtsLs.PrintValidate)CtsLs.PrintFont.PRINT_FORMAT_NORMAL_15)
      {
            IntPtr strEndorse = Marshal.AllocHGlobal(160); ;

            if (stParAppl.PrintValidate ==
(CtsLs.PrintValidate)CtsLs.PrintFont.PRINT_FORMAT_NORMAL_15)
               PrintType = CtsLs.PrintFont.PRINT_FORMAT_NORMAL_15;
            else if( stParAppl.PrintBold )
               PrintType = CtsLs.PrintFont.PRINT_FORMAT_BOLD;
            else
               PrintType = CtsLs.PrintFont.PRINT_FORMAT_NORMAL;

            if( stParAppl.PrintHigh == 0 )
                PrintType += 0x20;

            // Copy the Secure string to unmanaged memory (and decrypt it).
            strEndorse = Marshal.StringToHGlobalAnsi(stParAppl.Endorse_str);

            if( stParAppl.Endorse_str.Contains("%d") )
                 Reply = CtsLs.LSLoadStringWithCounterEx(hConnect, 0,
                                       (short)PrintType,
                                        strEndorse,
                                        (short)stParAppl.Endorse_str.Length,
                                        8, 3);
            else
                  Reply = CtsLs.LSLoadString(hConnect, 0,
                                           (short)PrintType,
                                           (short)stParAppl.Endorse_str.Length,
                                           strEndorse);
            if (Reply != CtsLs.LsReply.LS_OKAY)
            {
                 if (CheckReply(Reply, "LSLoadString"))
                 {
                      return Reply;
                 }
            }

            // Set variable for print
            PrintValidate = CtsLs.PrintValidate.PRINT_VALIDATE;
      }
      //-----------Print Logo-------------------------------------
      else if( stParAppl.PrintLogo )
      {
            PrintValidate = CtsLs.PrintValidate.PRINT_LOGO;
      }


      //----------- Set Sensibilità foto doppia sfogliatura -----------
      Reply = CtsLs.LSConfigDoubleLeafingAndDocLength(hConnect, 0,
                                                      stParAppl.DL_Type,
                                                      stParAppl.DL_Value,
                                                      stParAppl.DL_MinDoc,
                                                      stParAppl.DL_MaxDoc);
      if (Reply != CtsLs.LsReply.LS_OKAY)
      {
           if( CheckReply(Reply, "LSConfigDoubleLeafingAndDocLength"))
           {
              return Reply;
           }
      }


      //----------- Change stamp position -----------
      //Reply = CtsLs.LSChangeStampPosition(hConnect, 0,
                                    stParAppl.StampPosition, 0);
      //if (Reply != CtsLs.LsReply.LS_OKAY)
      //{
```

```
//    if (CheckReply(Reply, "LSChangeStampPosition"))
//    {
//        return Reply;
//    }
//}


//----------- Set attesa introduzione documento -----------
Reply = CtsLs.LSDisableWaitDocument(hConnect, 0, stParAppl.WaitTimeout);
if (Reply != CtsLs.LsReply.LS_OKAY)
{
     if (CheckReply(Reply, "LSDisableWaitDocument"))
     {
        return Reply;
     }
}


//----------- Set speed document -----------
Reply = CtsLs.LSSetUnitSpeed(hConnect, 0, stParAppl.LowSpeed);
if (Reply != CtsLs.LsReply.LS_OKAY)
{
     if (CheckReply(Reply, "LSSetSpeedUnit"))
     {
        return Reply;
     }
}


//----------- Set Light Intensity --------------------------
Reply = CtsLs.LSSetLightIntensity(hConnect, 0, stParAppl.LightIntensity);
if (Reply != CtsLs.LsReply.LS_OKAY)
{
     if (CheckReply(Reply, "LSSetLightIntensity"))
     {
             return Reply;
      }
}


//----------- Only for Ultra Violet type -----------
if( (UnitCfg[1] & MASK_SCANNER_UV) == MASK_SCANNER_UV )
{
    Reply = CtsLs.LSModifyPWMUltraViolet(hConnect, 0,
                        stParAppl.PercentPWM_UV,
                        stParAppl.Contrast_UV,
                        stParAppl.Threshold_UV);
    if (Reply != CtsLs.LsReply.LS_OKAY)
    {
        if (CheckReply(Reply, "LSModifyPWMUltraViolet"))
        {
              return Reply;
        }
    }
}


// Settaggio dei parametri fissi per LS_AutoDocHandle
dirBase = Marshal.StringToHGlobalAnsi(PathAppl + SAVE_DIRECTORY_IMAGE);
Filename = Marshal.StringToHGlobalAnsi(NAME_IMAGE);
```

```
      CodelineType = CtsLs.CodeLineType.NO_READ_CODELINE;
      TypeOfDecod = stParAppl.TypeOfDecod;
      C_x = C_y = C_w = C_h = 0;
      if( stParAppl.CodelineMICR != CtsLs.CodeLineType.NO_READ_CODELINE )
      {
           CodelineType = stParAppl.CodelineMICR;
      }
      else if( stParAppl.CodelineOCR != CtsLs.CodeLineType.NO_READ_CODELINE )
      {
           CodelineType = stParAppl.CodelineOCR;
           C_x = stParAppl.Codeline_Sw_x;
           C_y = stParAppl.Codeline_Sw_y;
           C_w = stParAppl.Codeline_Sw_w;
           C_h = stParAppl.Codeline_Sw_h;
           TypeOfDecod -= DECODE_OCR;
      }
      else if( stParAppl.BarcodeType != CtsLs.CodeLineType.NO_READ_CODELINE )
      {
           CodelineType = stParAppl.BarcodeType;
           C_x = stParAppl.Barcode_Sw_x;
           C_y = stParAppl.Barcode_Sw_y;
           C_w = stParAppl.Barcode_Sw_w;
           C_h = stParAppl.Barcode_Sw_h;
           TypeOfDecod -= DECODE_OCR;
      }

      Reply = CtsLs.LSAutoDocHandle(hConnect, 0,
                                     stParAppl.FrontStamp,
                                     (short)PrintValidate,
                                      (short)CodelineType,
                                      stParAppl.ScanMode,
                                      (short)CtsLs.Feeder.FEED_AUTO,
                                      (short)CtsLs.Sorter.SORTER_POCKET_1,
                                      stParAppl.NumDoc,
                                      stParAppl.ClearAlignImage,
                                      stParAppl.Side,
                                       0,
                                      stParAppl.SaveImage,
                                      dirBase,
                                      Filename,
                                      C_x, C_y, C_w, C_h,
                                      (short)CtsLs.Unit.UNIT_MM,
                                       0,
                                       stParAppl.FileFormat,
                                       stParAppl.Qual,
                                      (short)CtsLs.FileAttribute.SAVE_OVERWRITE,
                                       1,
                                        (short)(stParAppl.WaitTimeout ?
CtsLs.Wait.WAIT_YES : CtsLs.Wait.WAIT_NO),
                                        (short)stParAppl.BeepOnError,
                                       0,
                                       IntPtr.Zero,
                                       IntPtr.Zero);
      if (Reply != CtsLs.LsReply.LS_OKAY)
      {
           if (CheckReply(Reply, "LSAutoDocHandle"))
           {
              return Reply;
           }
```

```
}


// Salvo il nr. di documenti da processare
if( stParAppl.NumDoc != 0 )
     DocToProcess = stParAppl.NumDoc;
else
     DocToProcess = 1000000;


if (Save_FrontImage != IntPtr.Zero)
{
     CtsLs.LSFreeImage(0, ref Save_FrontImage);
     Save_FrontImage = IntPtr.Zero;
}
if (Save_RearImage != IntPtr.Zero)
{
     CtsLs.LSFreeImage(0, ref Save_RearImage);
     Save_RearImage = IntPtr.Zero;
}

//------------GetDocData-------------------------------------
Bitmap pBitmap = null;

//    if( Reply == LS_OKAY )
//    {
//          while( TRUE )
while (Reply == CtsLs.LsReply.LS_OKAY)
{
     Marshal.WriteByte(BufFrontFile, 0);
     Marshal.WriteByte(BufRearFile, 0);
     BufFrontImage = IntPtr.Zero;
     BufRearImage = IntPtr.Zero;
     BufFrontUVImage = BufFrontGrayUVImage = IntPtr.Zero;
     NoImage = IntPtr.Zero;
     NrDoc = 0;
     DocToRead = 0;
     NrPrinted = 0;

     Marshal.WriteByte(BufCodelineSW, 0);
     Marshal.WriteByte(BufCodelineHW, 0);
     Marshal.WriteByte(BufBarcode, 0);

     Reply = CtsLs.LSGetDocData(hConnect, 0,
                                   ref NrDoc,
                                   BufFrontFile,
                                   BufRearFile,
                                   IntPtr.Zero, //BufFrontNettoFile,
                                  IntPtr.Zero, //BufBackNettoFile,
                                   ref BufFrontImage,
                                   ref BufRearImage,
                                   ref BufFrontUVImage,
                                   ref NoImage, //BufbackNettoImage,
                                   BufCodelineSW,
                                   BufCodelineHW,
                                  BufBarcode,
                                   IntPtr.Zero,
                                   ref DocToRead,
                                   ref NrPrinted,
                                   IntPtr.Zero,
                                         IntPtr.Zero);

            // Se e` doppia sfogliatura non termino
            if (((Reply != CtsLs.LsReply.LS_OKAY) &&
                  (Reply != CtsLs.LsReply.LS_SORTER1_FULL) &&
                  (Reply != CtsLs.LsReply.LS_DOUBLE_LEAFING_WARNING)))
            {
                if( Reply != CtsLs.LsReply.LS_FEEDER_EMPTY )
                    CheckReply(Reply, "LS_GetDocData");
                break;
            }


            //metto il test per le altre codeline
            if ((TypeOfDecod & DECODE_OCR) == DECODE_OCR )
            {
                 CtsLs.READOPTIONS ro;
                 byte[] CodelineOpt  = new byte[4];
                 int len_codeline;

                 ro.PutBlanks = 1;
                 ro.TypeRead = 'N';
  if( stParAppl.CodelineOCR == CtsLs.CodeLineType.READ_CODELINE_SW_E13B_X_OCRB )
                 {
                       ro.TypeRead = 'X';
                CodelineOpt[0] = (byte)CtsLs.CodeLineType.READ_CODELINE_SW_E13B;
          CodelineOpt[1] = (byte)CtsLs.CodeLineType.READ_CODELINE_SW_OCRB_ITALY;
                      CodelineOpt[2] = (byte)'\0';
                 }
                 else
                  {
                        CodelineOpt[0] = (byte)stParAppl.CodelineOCR;
                        CodelineOpt[2] = (byte)'\0';
                  }

                  len_codeline = (int)CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH;

                  Reply = CtsLs.LSCodelineReadFromBitmap(0,
                                                        BufFrontImage,
                                                        CodelineOpt,
                                                        stParAppl.Unit_measure,
                                                        stParAppl.Codeline_Sw_x,
                                                        stParAppl.Codeline_Sw_y,
                                                        stParAppl.Codeline_Sw_w,
                                                        stParAppl.Codeline_Sw_h,
                                                        ref ro,
                                                        BufCodelineSW,
                                                        ref len_codeline);
                  if( Reply != CtsLs.LsReply. LS_OKAY )
                  {
                        CheckReply(Reply, "LSCodelineReadFromBitmap");
                        Reply = CtsLs.LsReply.LS_OKAY;     // Set Ok for not
exit from the loop
                  }
             }
             if ((TypeOfDecod & DECODE_BARCODE) == DECODE_BARCODE )
             {
                 int len_barcode = (int)CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH;

                 Reply = CtsLs.LSReadBarcodeFromBitmap(0,
                                                    BufFrontImage,
                                                    (byte)stParAppl.BarcodeType,
                                                    (int)stParAppl.Barcode_Sw_x,
                                                    (int)stParAppl.Barcode_Sw_y,
                                                    (int)stParAppl.Barcode_Sw_w,
                                                    (int)stParAppl.Barcode_Sw_h,
                                                    BufBarcode,
                                                    ref len_barcode);
                 if (Reply != CtsLs.LsReply. LS_OKAY)
                 {
                    CheckReply(Reply, "LSReadBarcodeFromBitmap");
                    Reply = CtsLs.LsReply.LS_OKAY;         // Set Ok for not
exit from the loop
                 }
             }
             if ((TypeOfDecod & DECODE_PDF417) == DECODE_PDF417 )
             {
                 int len_barcode = (int)CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH;

                 Reply = CtsLs.LSReadPdf417FromBitmap(0,
                                                      BufFrontImage,
                                                       BufBarcode,
                                                       ref len_barcode,
                                                       0, 0, 0, 0, 0);
                 if (Reply != CtsLs.LsReply. LS_OKAY)
                  {
                       CheckReply(Reply, "LSReadPdf417FromBitmap");
                      Reply = CtsLs.LsReply.LS_OKAY;       // Set Ok for not
exit from the loop
                  }
             }


    if (stParAppl.ScanMode == (short)CtsLs.ScanMode.SCAN_MODE_256GR100_AND_UV ||
        stParAppl.ScanMode == (short)CtsLs.ScanMode.SCAN_MODE_256GR200_AND_UV ||
        stParAppl.ScanMode == (short)CtsLs.ScanMode.SCAN_MODE_256GR300_AND_UV)
             {
                  // Build the mergered gray UV image
                  if (BufFrontUVImage != IntPtr.Zero)
                  {
                         CtsLs.LSMergeImageGrayAndUV(0,
                                                    BufFrontImage,
                                                    BufFrontUVImage,
                                                    0, //stParAppl.Threshold_UV,
                                                    0,
                                                    ref BufFrontGrayUVImage);
            //if( stParAppl.SaveImage == IMAGE_SAVE_ON_FILE ||
                  stParAppl.SaveImage == IMAGE_SAVE_BOTH )
            //{
            //    // Check if I must save the image
            //    if( stParAppl.FileFormat == (short)CtsLs.FileType.FILE_JPEG )
            //    {
            //        FullFName, "%s\\%s%04dGUV.jpg", dirBase, Filename, (NrDoc
- 1));
            //        Reply = LSSaveJPEG(hWnd, BufFrontGrayUVImage,
stParAppl.Qual, FullFName);
            //    }
            //    else if( stParAppl.FileFormat ==
(short)CtsLs.FileType.FILE_BMP )
            //    {
```

```
            //        sprintf(FullFName, "%s\\%s%04dGUV.bmp", dirBase, Filename,
(NrDoc - 1));
            //        Reply = LSSaveDIB(hWnd, BufFrontGrayUVImage, FullFName);
            //    }
            //    else if( stParAppl.FileFormat ==
(short)CtsLs.FileType.FILE_CCITT_GROUP4 )
            //    {
            //        sprintf(FullFName, "%s\\%s%04dGUV.tif", dirBase, Filename,
(NrDoc - 1));
            //        Reply = LSSaveTIFF(hWnd, BufFrontGrayUVImage, FullFName,
FILE_TIF, SAVE_OVERWRITE, 1);
            //    }
            //}
                  }
            }


            // Show Codeline
           if (Marshal.ReadByte(BufCodelineHW) != 0)
               tbCodeline.Text = Marshal.PtrToStringAnsi(BufCodelineHW);
           else
               tbCodeline.Text = "";

          //// Show the immage
          CtsLs.BITMAPINFOHEADER pBmp =
                 (CtsLs.BITMAPINFOHEADER)Marshal.PtrToStructure(BufFrontImage,
                 typeof(CtsLs.BITMAPINFOHEADER));

          pBitmap = new Bitmap(pBmp.biWidth, pBmp.biHeight,
          PixelFormat.Format24bppRgb);

//                BitmapData bmpData = pBitmap.LockBits(new Rectangle(0, 0,
pBitmap.Width, pBitmap.Height), ImageLockMode.WriteOnly, pBitmap.PixelFormat);
          int xx, yy;
          Int32 diff, WidthBytes = pBmp.biWidth;
          if( (diff = WidthBytes % 4) != 0 )
               WidthBytes += (4 - diff);
          Int32 row, col;
          row = pBmp.biHeight - 1;
          col = pBmp.biWidth - 1;
          for (yy = 0; yy < pBmp.biHeight; yy++)
          {
              for (xx = 0; xx < pBmp.biWidth; xx++)
              {
                  byte Pixel = Marshal.ReadByte((IntPtr)((int)BufFrontImage +
1064 + ((yy * WidthBytes) + xx)));
                  Color Pixel24;
                  Pixel24 = Color.FromArgb((Pixel * 256 * 256) + (Pixel * 256) +
Pixel);
                  pBitmap.SetPixel(xx, (row - yy), Pixel24);
                  //Marshal.WriteByte((IntPtr)((int)bmpData.Scan0 + (xx * yy)),
Pixel);
                  //Marshal.WriteByte((IntPtr)((int)bmpData.Scan0 + (xx * yy) +
1), Pixel);
                  //Marshal.WriteByte((IntPtr)((int)bmpData.Scan0 + (xx * yy) +
2), Pixel);
              }
          }
 //      CopyMemory(bmpData.Scan0, (IntPtr)((int)BufFrontImage + 1064),
pBmp.biSizeImage);
```

```
//      pBitmap.UnlockBits(bmpData);

          if (pbImage.Image != null)
          {
               pbImage.Image.Dispose();
               //    //pbImage.Image = null;
          }
          pbImage.Image = pBitmap;

          // Refresh the form
          Application.DoEvents();

          // Abilito il bottone visualizzazione retro
          btRear.Enabled = true;

          // Free the previous image memory and save the current
          if (Save_FrontImage != IntPtr.Zero)
          {
               CtsLs.LSFreeImage(0, ref Save_FrontImage);
          }
          if (BufFrontGrayUVImage != IntPtr.Zero)
          {
               Save_FrontImage = BufFrontGrayUVImage;
               CtsLs.LSFreeImage(0, ref BufFrontImage);
               CtsLs.LSFreeImage(0, ref BufFrontUVImage);
          }
          else if (BufFrontImage != IntPtr.Zero)
          {
               Save_FrontImage = BufFrontImage;
          }

          if (Save_RearImage != IntPtr.Zero)
          {
               CtsLs.LSFreeImage(0, ref Save_RearImage);
          }
          Save_RearImage = BufRearImage;

          NrCheque ++;
      }     // Fine while( TRUE )

     // Free of local variable
     Marshal.FreeHGlobal(Filename);
     Marshal.FreeHGlobal(dirBase);
     Marshal.FreeHGlobal(BufBarcode);
     Marshal.FreeHGlobal(BufCodelineHW);
     Marshal.FreeHGlobal(BufCodelineSW);
     Marshal.FreeHGlobal(BufRearFile);
     Marshal.FreeHGlobal(BufFrontFile);
}
```

14.5. Sample code in VB (for .NET)

The source is available on demand :

CtsLsClass.vb

```
Imports System.Runtime.InteropServices

Public Class CtsLs
    <DllImport("LsApi.dll")> _
    Public Shared Function LSConnect(ByVal hWnd As Integer, ByVal hInst As
Integer, ByVal Peripheral As Short, ByRef hConnect As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSDisconnect(ByVal hConnect As Short, ByVal hWnd As
Integer) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSUnitIdentify(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal pLsCfg As Byte(), ByVal LsModel As IntPtr, ByVal FwVersion As
IntPtr, ByVal FwDate As IntPtr, _
     ByVal PeripheralID As IntPtr, ByVal BoardVersion As IntPtr, ByVal
DecoderExpVersion As IntPtr, ByVal InkJetVersion As IntPtr, ByVal FeederVersion
As IntPtr, ByVal SorterVersion As IntPtr, _
     ByVal MotorVersion As IntPtr, ByVal Reserved1 As IntPtr, ByVal Reserved2 As
IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSUnitStatus(ByVal hConnect As Short, ByVal hWnd As
Integer, ByRef lpStatus As UNITSTATUS) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSReset(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal ResetType As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSLoadStringWithCounterEx(ByVal hConnect As Short,
ByVal hWnd As Integer, ByVal PrintType As Short, ByVal strEndorse As IntPtr,
ByVal LenEndorse As Short, ByVal StartNumber As UInt32, _
     ByVal [Step] As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSLoadString(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal PrintType As Short, ByVal LenEndorse As Short, ByVal strEndorse
As IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSConfigDoubleLeafingAndDocLength(ByVal hConnect As
Short, ByVal hWnd As Integer, ByVal Type As Int32, ByVal Value As Short, ByVal
DocMin As Int32, ByVal DocMax As Int32) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSChangeStampPosition(ByVal hConnect As Short, ByVal
hWnd As Integer, ByVal [Step] As Short, ByVal Reserved As Byte) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSDisableWaitDocument(ByVal hConnect As Short, ByVal
hWnd As Integer, ByVal fWait As Boolean) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSSetUnitSpeed(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal UnitSpeed As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSSetLightIntensity(ByVal hConnect As Short, ByVal
hWnd As Integer, ByVal UnitSpeed As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSModifyPWMUltraViolet(ByVal hConnect As Short, ByVal
hWnd As Integer, ByVal UnitSpeed As Short, ByVal HighContrast As Boolean, ByVal
Reserved As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSAutoDocHandle(ByVal hConnect As Short, ByVal hWnd
As Integer, ByVal Stamp As Short, ByVal Validate As Short, ByVal CodeLine As
Short, ByVal ScanMode As Short, _
     ByVal Feeder As Short, ByVal Sorter As Short, ByVal NumDocument As Short,
ByVal ClearBlack As Short, ByVal Side As Byte, ByVal ReadMode As Short, _
     ByVal SaveImage As Short, ByVal DirectoryFile As IntPtr, ByVal BaseFilename
As IntPtr, ByVal pos_x As [Single], ByVal pos_y As [Single], ByVal sizeW As
[Single], _
     ByVal sizeH As [Single], ByVal OriginMeasureDoc As Short, ByVal
OcrImageSide As Short, ByVal FileFormat As Short, ByVal Quality As Integer,
ByVal SaveMode As Integer, _
     ByVal PageNumber As Integer, ByVal WaitTimeout As Short, ByVal Beep As
Short, ByVal Reserved1 As Integer, ByVal Reserved2 As IntPtr, ByVal Reserved3 As
IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSGetDocData(ByVal hConnect As Short, ByVal hWnd As
Integer, ByRef NrDoc As UInt32, ByVal FilenameFront As IntPtr, ByVal
FilenameRear As IntPtr, ByVal Reserved1 As IntPtr, _
     ByVal Reserved2 As IntPtr, ByRef FrontImage As IntPtr, ByRef RearImage As
IntPtr, ByRef Reserved3 As IntPtr, ByRef Reserved4 As IntPtr, ByVal CodelineSW
As IntPtr, _
     ByVal CodelineHW As IntPtr, ByVal Barcode As IntPtr, ByVal CodelinesOptical
As IntPtr, ByRef DocToRead As Short, ByRef NrPrinted As Int32, ByVal Reserved5
As IntPtr, _
     ByVal Reserved6 As IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSDocHandle(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal Stamp As Short, ByVal Validate As Short, ByVal CodeLine As Short,
ByVal Side As Byte, _
     ByVal ScanMode As Short, ByVal Feeder As Short, ByVal Sorter As Short,
ByVal WaitTimeout As Short, ByVal Beep As Short, ByRef NrDoc As UInt32, _
     ByVal ScanDocType As Int16, ByVal Reserved As Int32) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSReadCodeline(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal CodelineHW As IntPtr, ByRef LenCodelineHW As Short, ByVal Barcode
As IntPtr, ByRef LenBarcode As Short, _
     ByVal CodelinesOptical As IntPtr, ByRef LenOptic As Short) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSReadImage(ByVal hConnect As Short, ByVal hWnd As
Integer, ByVal ClearBlack As Short, ByVal Side As Byte, ByVal ReadMode As Short,
ByVal NrDoc As UInt32, _
     ByRef FrontImage As IntPtr, ByRef RearImage As IntPtr, ByRef Reserved1 As
IntPtr, ByVal Reserved2 As IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSCodelineReadFromBitmap(ByVal hWnd As Integer, ByVal
hImage As IntPtr, ByVal CodelineType As Byte(), ByVal UintMeasure As Short,
ByVal Pos_x As Single, ByVal Pos_y As Single, _
     ByVal Width As Single, ByVal Height As Single, ByRef ro As READOPTIONS,
ByVal Codeline As IntPtr, ByRef Length_Codeline As Integer) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSReadBarcodeFromBitmap(ByVal hWnd As Integer, ByVal
hImage As IntPtr, ByVal BarcodeType As Byte, ByVal Pos_x As Single, ByVal Pos_y
As Single, ByVal Width As Single, _
     ByVal Height As Single, ByVal Codeline As IntPtr, ByRef Length_Codeline As
Integer) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSReadPdf417FromBitmap(ByVal hWnd As Integer, ByVal
hImage As IntPtr, ByVal Codeline As IntPtr, ByRef Length_Codeline As Integer,
ByVal Reserved As Byte, ByVal Pos_x As Single, _
     ByVal Pos_y As Single, ByVal Width As Single, ByVal Height As Single) As
Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSMergeImageGrayAndUV(ByVal hWnd As Integer, ByVal
hFrontGrayImage As IntPtr, ByVal hFrontUVImage As IntPtr, ByVal Reserved As
Single, ByVal Reserved2 As Single, ByRef hGrayUVImage As IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSFreeImage(ByVal hWnd As Integer, ByRef hImage As
IntPtr) As Integer
    End Function
    <DllImport("LsApi.dll")> _
    Public Shared Function LSUnitHistory(ByVal hConnect As Short, ByVal hWnd As
Integer, ByRef UnitHistory As UNITHISTORY) As Integer
    End Function

    Public Structure BITMAPINFOHEADER
        Public biSize As UInt32
        Public biWidth As Int32
        Public biHeight As Int32
        Public biPlanes As Int16
        Public biBitCount As Int16
        Public biCompression As UInt32
        Public biSizeImage As UInt32
        Public biXPelsPerMeter As Int32
        Public biYPelsPerMeter As Int32
        Public biClrUsed As UInt32
        Public biClrImportant As UInt32
    End Structure


    Public Structure UNITSTATUS
        Public Size As Integer                  ' Size of the structure
        Public UnitStatus As Integer                  ' Ls40 Ls100 Ls150 Ls5xx Ls800
        Public Photo_Feeder As Boolean                ' Ls40 Ls100 Ls150 Ls5xx Ls800
        Public Photo_Sorter As Boolean            '      Ls100
        Public Photo_MICR As Boolean              '      Ls100 Ls150 Ls5xx
        Public Photo_Path_Ls100 As Boolean        '      Ls100
        Public Photo_Scanners As Boolean          '      Ls100
        Public Unit_Just_ON As Boolean            ' Ls40 Ls100 Ls150
        Public Photo_Double_Leafing_Down As Boolean   '      Ls100 Ls150
        Public Photo_Double_Leafing_Middle As Boolean '            Ls150
        Public Photo_Double_Leafing_Up As Boolean '      Ls100 Ls150
        Public Photo_Card As Boolean              '            Ls150
        Public Pockets_All_Full As Boolean        '            Ls150 Ls5xx
        Public Photo_Stamp As Boolean             '                  Ls5xx
        Public Photo_Exit As Boolean              '                  Ls5xx
        Public Pocket_1_Full As Boolean           '                  Ls5xx
        Public Pocket_2_Full As Boolean           '                  Ls5xx
        Public Photo_Path_Feeder As Boolean       '                        Ls800
        Public Photo_Path_Module_Begin As Boolean '                        Ls800
        Public Photo_Path_Binary_Rigth As Boolean '                        Ls800
        Public Photo_Path_Binary_Left As Boolean  '                        Ls800
        Public Photo_Path_Module_End As Boolean   '                        Ls800
        Public Sorter_1_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_1_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_1_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_1_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_1_input_pocket_3 As Boolean '                        Ls800
        Public Sorter_1_pocket_3_full As Boolean  '                        Ls800
        Public Sorter_2_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_2_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_2_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_2_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_2_input_pocket_3 As Boolean '                        Ls800
        Public Sorter_2_pocket_3_full As Boolean  '                        Ls800
        Public Sorter_3_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_3_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_3_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_3_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_3_input_pocket_3 As Boolean '                        Ls800
        Public Sorter_3_pocket_3_full As Boolean  '                        Ls800
        Public Sorter_4_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_4_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_4_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_4_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_4_input_pocket_3 As Boolean '                        Ls800
        Public Sorter_4_pocket_3_full As Boolean  '                        Ls800
        Public Sorter_5_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_5_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_5_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_5_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_5_input_pocket_3 As Boolean '                        Ls800
        Public Sorter_5_pocket_3_full As Boolean  '                        Ls800
        Public Sorter_6_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_6_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_6_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_6_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_6_input_pocket_3 As Boolean '                        Ls800
        Public Sorter_6_pocket_3_full As Boolean  '                        Ls800
        Public Sorter_7_input_pocket_1 As Boolean '                        Ls800
        Public Sorter_7_pocket_1_full As Boolean  '                        Ls800
        Public Sorter_7_input_pocket_2 As Boolean '                        Ls800
        Public Sorter_7_pocket_2_full As Boolean  '                        Ls800
        Public Sorter_7_input_pocket_3 As Boolean '                        Ls800
```

```
        Public Sorter_7_pocket_3_full As Boolean  '                        Ls800
        Public Photo_Trigger As Boolean           ' Ls40
        Public Document_Retained As Boolean       ' Ls40
    End Structure

    Public Structure UNITHISTORY
        Public Size As Int32                ' Size of the structure
        Public doc_sorted As UInt32         ' Document sortered
        Public doc_retained As UInt32       ' Nr. of document retained
        Public doc_retained_micr As UInt32  ' Nr Doc. retained after MICR header
        Public doc_retained_scan As UInt32  ' Nr Doc .retained after scanning
        Public doc_ink_jet As UInt32        ' Nr. of document printed
        Public doc_stamped As UInt32        ' Nr. of document stamped
        Public tot_paper_jams As UInt32     ' Totally of Paper jam
        Public jams_in_feeder As UInt32     ' Nr. jam in the feeder
        Public jams_in_micr As UInt32       ' Nr. jam during the MICR reading
        Public jams_scanner As UInt32       ' Nr. jam between scanners
        Public jams_stamp As UInt32         ' Nr. jam at stamp document
        Public jams_on_exit As UInt32       ' Nr. jam after the film
        Public jams_card As UInt32          ' Nr. jam in the card entry
        Public nr_double_leafing As UInt32  ' Nr DoubleLeafing occurs Ls800 only
        Public tot_doc_MICR_err As UInt32   ' Nr. document MICR, read with error
        Public doc_cmc7_err As UInt32       ' Nr. document CMC7, read with error
        Public doc_e13b_err As UInt32       ' Nr. document E13B, read with error
        Public doc_hw_barcode_err As UInt32 ' Nr. of document Barcode, read from
LS with error
        Public doc_hw_optic_err As UInt32   ' Nr. of document OCR, read from LS
with error
        Public num_turn_on As UInt32        ' Nr. of power ON
        Public time_peripheral_on As UInt32 ' Minutes peripheral time life
                                            ' Section specific Ls800 unit
        Public jam_front_scanner As UInt32  ' Jam in scanner front
        Public jam_track_left As UInt32     ' Jam in the left track
        Public jam_track_right As UInt32    ' Jam in the right track
        Public jam_back_scanner As UInt32   ' Jam in scanner back
        Public jam_in_the_sorters As UInt32 ' Jam in sorters track
                                         ' Section compiled only from Ls800 unit
        Public nr_drops_printed As UInt32   ' Nr. drops printed
    End Structure

    Public Structure READOPTIONS
        Public PutBlanks As Integer  ' 0 = CodeLIne whitout blans, 1 = CodeLine
with 1 blanks
        Public TypeRead As Char      ' 'N' for 1 type of CodeLine, 'X' for
CodeLine E13B switch OCRB
    End Structure


    ' Parameter Peripheral Type
    Public Enum LsUnitType As Short
        LS_40_LSCONNECT = 39
        LS_40_USB = 40
        LS_100_LSCONNECT = 109
        LS_100_USB = 100
        LS_100_RS232 = 101
        LS_100_ETH = 110
        LS_150_LSCONNECT = 149
        LS_150_USB = 150
        LS_200_USB = 201
        LS_5xx_SCSI = 500
        LS_515_LSCONNECT = 501
        LS_515_USB = 502
        LS_520_USB = 520
        LS_800_USB = 801
    End Enum

    Public Enum Stamp As Short
        STAMP_NO = 0
        ' No stamp is done
        STAMP_FRONT = 1
        ' Stamp on front document
        STAMP_BACK = 2
        ' Stamp on rear document
        STAMP_FRONT_AND_BACK = 3
        ' Stamp front and rear document
    End Enum

    Public Enum PrintValidate As Short
        NO_PRINT_VALIDATE = 0
        ' No print is done
        PRINT_VALIDATE = 1
        ' Print done
        PRINT_LOGO = 4
        ' Print a logo only
        PRINT_VALIDATE_WITH_LOGO = 5
        ' Print logo and lines
    End Enum

    Public Enum Feeder As Short
        FEED_AUTO = 0
        ' Start Document from Feeder
        FEED_FROM_PATH = 1
        ' Start Document from Unit Path
    End Enum

    Public Enum Sorter As Short
        SORTER_DOC_HOLDED = 0
        SORTER_POCKET_1 = 1
        SORTER_POCKET_2 = 2
        SORTER_AUTOMATIC = 3
        SORTER_SWICTH_1_TO_2 = 4
        SORTER_DOC_EJECTED = 5
        SORTER_ON_CODELINE_CALLBACK = 6

        ' For Ls800 unit
        SORTER_CIRCULAR = 48
        SORTER_SEQUENTIAL = 49
        SORTER_POCKET_0_SELECTED = 50
        SORTER_POCKET_1_SELECTED = 51
        SORTER_POCKET_2_SELECTED = 52
        SORTER_POCKET_3_SELECTED = 53
        SORTER_POCKET_4_SELECTED = 54
        SORTER_POCKET_5_SELECTED = 55
        SORTER_POCKET_6_SELECTED = 56
        SORTER_POCKET_7_SELECTED = 57
        SORTER_POCKET_8_SELECTED = 58
        SORTER_POCKET_9_SELECTED = 59
        SORTER_POCKET_10_SELECTED = 60
        SORTER_POCKET_11_SELECTED = 61
        SORTER_POCKET_12_SELECTED = 62
        SORTER_POCKET_13_SELECTED = 63
        SORTER_POCKET_14_SELECTED = 64
        SORTER_POCKET_15_SELECTED = 65
        SORTER_POCKET_16_SELECTED = 66
        SORTER_POCKET_17_SELECTED = 67
        SORTER_POCKET_18_SELECTED = 68
        SORTER_POCKET_19_SELECTED = 69
        SORTER_POCKET_20_SELECTED = 70
        SORTER_POCKET_21_SELECTED = 71
    End Enum

    Public Enum CodeLineType As Byte
        NO_READ_CODELINE = 0
        READ_CODELINE_HW_MICR = 1
        READ_CODELINE_E13B_MICR_WITH_OCR = 15

        READ_CODELINE_SW_OCRA = 65             ''A',
        READ_CODELINE_SW_OCRB_NUM = 66         ''B',
        READ_CODELINE_SW_OCRB_ALFANUM = 67     ''C',
        READ_CODELINE_SW_OCRB_ITALY = 70       ''F',
        READ_CODELINE_SW_E13B = 69             ''E',
        READ_CODELINE_SW_E13B_X_OCRB = 88      ''X',
        READ_BARCODE_2_OF_5 = 50
        READ_BARCODE_CODE39 = 51
        READ_BARCODE_CODE128 = 52
        '               READ_BARCODE_EAN13 = 53,

        MAX_CODE_LINE_LENGTH = 254
    End Enum

    Public Enum Unit As Short
        UNIT_MM = 0
        UNIT_INCH = 1
    End Enum

    Public Class OcrHeight
        Public Const OCR_MAX_HEIGHT_IN_MM As Double = 10.5
        Public Const OCR_MAX_HEIGHT_IN_INCH As Double = 0.41
    End Class

    Public Enum BlankInCodeline As Short
        BLANK_IN_CODELINE_NO = 0
        BLANK_IN_CODELINE_YES = 1
    End Enum

    Public Enum OriginOCR As Short
        ORIGIN_BOTTOM_RIGHT_MM = 10
        ORIGIN_BOTTOM_RIGHT_INCH = 20
    End Enum

    Public Enum ScanMode As Short
        SCAN_MODE_BW = 1
        SCAN_MODE_16_GRAY_100 = 2
        SCAN_MODE_16_GRAY_200 = 3
        SCAN_MODE_256_GRAY_100 = 4
        SCAN_MODE_256_GRAY_200 = 5
        SCAN_MODE_COLOR_100 = 10
        SCAN_MODE_COLOR_200 = 11
        SCAN_MODE_16_GRAY_300 = 20
        SCAN_MODE_256_GRAY_300 = 21
        SCAN_MODE_COLOR_300 = 22
        SCAN_MODE_256GR100_AND_UV = 40
        SCAN_MODE_256GR200_AND_UV = 41
        SCAN_MODE_256GR300_AND_UV = 42
    End Enum

    Public Enum ScanDocType As Short
        SCAN_PAPER_DOCUMENT = 0
        SCAN_CARD = 1
    End Enum

    Public Enum Side As Short
        SIDE_NONE_IMAGE = 78         ''N',
        SIDE_FRONT_IMAGE = 70        ''F',
        SIDE_BACK_IMAGE = 66         ''B',
        SIDE_ALL_IMAGE = 88          ''X',
        SIDE_FRONT_UV = 85           ''U',
        SIDE_FRONT_MERGED = 77       ''M',
    End Enum

    Public Enum Wait As Short
        WAIT_NO = 71                 ''G',
        WAIT_YES = 87                ''W',
    End Enum

    Public Enum Beep As Short
        BEEP_NO = 0
        BEEP_YES = 1
    End Enum

    Public Enum ClearBlack As Short
        CLEAR_BLACK_NO = 0
        CLEAR_BLACK_YES = 1
        CLEAR_AND_ALIGN_IMAGE = 2
    End Enum

    Public Enum PrintFont As Byte
        PRINT_NO_STRING = 0
        PRINT_FONT_NORMAL = 78            ''N',
        PRINT_FONT_BOLD = 66              ''B',
        PRINT_FONT_NORMAL_15 = 65         ''A',
        PRINT_UP_FONT_NORMAL = 110        ''n',
        PRINT_UP_FONT_BOLD = 98           ''b',
        PRINT_UP_FONT_NORMAL_15_CHAR = 97 ''a',
    End Enum

    Public Enum DoubleLeafing As Short

        DOUBLE_LEAFING_WARNING = 0
        DOUBLE_LEAFING_ERROR = 1     'DOUBLE_LEAFING_LEVEL1 = 1,non lo uso piu'
        DOUBLE_LEAFING_LEVEL2 = 2
        DOUBLE_LEAFING_LEVEL3 = 3
        DOUBLE_LEAFING_DEFAULT = 4
        DOUBLE_LEAFING_LEVEL4 = 5
        DOUBLE_LEAFING_LEVEL5 = 6
        DOUBLE_LEAFING_DISABLE = 7
    End Enum

    Public Enum Reset As Short
        RESET_ERROR = 48           ''0',
        RESET_PATH = 49           ''1',
        RESET_BELT_CLEANING = 50  ''2',
    End Enum

    Public Enum ImageSave As Short
        IMAGE_SAVE_ON_FILE = 4
        IMAGE_SAVE_HANDLE = 5
        IMAGE_SAVE_BOTH = 6
        IMAGE_SAVE_NONE = 7
    End Enum

    Public Enum FileType As Short
        FILE_JPEG = 10
        FILE_BMP = 11
        FILE_TIF = 3
        FILE_CCITT = 25
        FILE_CCITT_GROUP3_1DIM = 27
        FILE_CCITT_GROUP3_2DIM = 28
        FILE_CCITT_GROUP4 = 29
    End Enum

    Public Enum FileAttribute As Short
        SAVE_OVERWRITE = 0
        SAVE_APPEND = 1
        SAVE_REPLACE = 2
        SAVE_INSERT = 3
    End Enum

    Public Enum LsSpeed As Short
        SPEED_DEFAULT = 0
        SPEED_STAMP = 1
    End Enum

    Public Enum Badge As Short
        BADGE_READ_TRACK_1 = &H20
        BADGE_READ_TRACK_2 = &H40
        BADGE_READ_TRACK_3 = &H80
        BADGE_READ_TRACKS_1_2 = &H60
        BADGE_READ_TRACKS_2_3 = &HC0
        BADGE_READ_TRACKS_1_2_3 = &HE0
    End Enum

    Public Class LsReply
        ' ---------------------------------------------------------------------
        '                          REPLY-CODE
        ' ---------------------------------------------------------------------
        Public Const LS_OKAY As Integer = 0


        ' ---------------------------------------------------------------------
        '                  ERRORS
        ' ---------------------------------------------------------------------
        Public Const LS_SYSTEM_ERROR As Integer = -1
        Public Const LS_USB_ERROR As Integer = -2
        Public Const LS_PERIPHERAL_NOT_FOUND As Integer = -3
        Public Const LS_HARDWARE_ERROR As Integer = -4
        Public Const LS_PERIPHERAL_OFF_ON As Integer = -5
        Public Const LS_RESERVED_ERROR As Integer = -6
        Public Const LS_PAPER_JAM As Integer = -7
        Public Const LS_TARGET_BUSY As Integer = -8
```

```
Public Const LS_INVALID_COMMAND As Integer = -9
Public Const LS_DATA_LOST As Integer = -10
Public Const LS_COMMAND_IN_EXECUTION_YET As Integer = -11
Public Const LS_JPEG_ERROR As Integer = -12
Public Const LS_COMMAND_SEQUENCE_ERROR As Integer = -13
Public Const LS_PC_HW_ERROR As Integer = -14
Public Const LS_IMAGE_OVERWRITE As Integer = -15
Public Const LS_INVALID_HANDLE As Integer = -16
Public Const LS_NO_LIBRARY_LOAD As Integer = -17
Public Const LS_BMP_ERROR As Integer = -18
Public Const LS_TIFF_ERROR As Integer = -19
Public Const LS_IMAGE_NO_MORE_AVAILABLE As Integer = -20
Public Const LS_IMAGE_NO_FILMED As Integer = -21
Public Const LS_IMAGE_NOT_PRESENT As Integer = -22
Public Const LS_FUNCTION_NOT_AVAILABLE As Integer = -23
Public Const LS_DOCUMENT_NOT_SUPPORTED As Integer = -24
Public Const LS_BARCODE_ERROR As Integer = -25
Public Const LS_INVALID_LIBRARY As Integer = -26
Public Const LS_INVALID_IMAGE As Integer = -27
Public Const LS_INVALID_IMAGE_FORMAT As Integer = -28
Public Const LS_INVALID_BARCODE_TYPE As Integer = -29
Public Const LS_OPEN_NOT_DONE As Integer = -30
Public Const LS_INVALID_TYPE_COMMAND As Integer = -31
Public Const LS_INVALID_CLEARBLACK As Integer = -32
Public Const LS_INVALID_SIDE As Integer = -33
Public Const LS_MISSING_IMAGE As Integer = -34
Public Const LS_INVALID_TYPE As Integer = -35
Public Const LS_INVALID_SAVEMODE As Integer = -36
Public Const LS_INVALID_PAGE_NUMBER As Integer = -37
Public Const LS_INVALID_NRIMAGE As Integer = -38
Public Const LS_INVALID_STAMP As Integer = -39
Public Const LS_INVALID_WAITTIMEOUT As Integer = -40
Public Const LS_INVALID_VALIDATE As Integer = -41
Public Const LS_INVALID_CODELINE_TYPE As Integer = -42
Public Const LS_MISSING_NRIMAGE As Integer = -43
Public Const LS_INVALID_SCANMODE As Integer = -44
Public Const LS_INVALID_BEEP As Integer = -45
Public Const LS_INVALID_FEEDER As Integer = -46
Public Const LS_INVALID_SORTER As Integer = -47
Public Const LS_INVALID_BADGE_TRACK As Integer = -48
Public Const LS_MISSING_FILENAME As Integer = -49
Public Const LS_INVALID_QUALITY As Integer = -50
Public Const LS_INVALID_FILEFORMAT As Integer = -51
Public Const LS_INVALID_COORDINATE As Integer = -52
Public Const LS_MISSING_HANDLE_VARIABLE As Integer = -53
Public Const LS_INVALID_POLO_FILTER As Integer = -54
Public Const LS_INVALID_ORIGIN_MEASURES As Integer = -55
Public Const LS_INVALID_SIZEH_VALUE As Integer = -56
Public Const LS_INVALID_FORMAT As Integer = -57
Public Const LS_STRINGS_TOO_LONGS As Integer = -58
Public Const LS_READ_IMAGE_FAILED As Integer = -59
Public Const LS_INVALID_CMD_HISTORY As Integer = -60
Public Const LS_MISSING_BUFFER_HISTORY As Integer = -61
Public Const LS_INVALID_ANSWER As Integer = -62
Public Const LS_OPEN_FILE_ERROR_OR_NOT_FOUND As Integer = -63
Public Const LS_READ_TIMEOUT_EXPIRED As Integer = -64
Public Const LS_INVALID_METHOD As Integer = -65
Public Const LS_CALIBRATION_FAILED As Integer = -66
Public Const LS_INVALID_SAVEIMAGE As Integer = -67
Public Const LS_INVALID_UNIT As Integer = -68
Public Const LS_INVALID_NRWINDOWS As Integer = -71
Public Const LS_INVALID_VALUE As Integer = -72
Public Const LS_ILLEGAL_REQUEST As Integer = -73
Public Const LS_INVALID_NR_CRITERIA As Integer = -74
Public Const LS_MISSING_CRITERIA_STRUCTURE As Integer = -75
Public Const LS_INVALID_MOVEMENT As Integer = -76
Public Const LS_INVALID_DEGREE As Integer = -77
Public Const LS_R0TATE_ERROR As Integer = -78
Public Const LS_MICR_VALUE_OUT_OF_RANGE As Integer = -79
Public Const LS_PERIPHERAL_RESERVED As Integer = -80
Public Const LS_INVALID_NCHANGE As Integer = -81
Public Const LS_BRIGHTNESS_ERROR As Integer = -82
Public Const LS_CONTRAST_ERROR As Integer = -83
Public Const LS_INVALID_SIDETOPRINT As Integer = -84
Public Const LS_DOUBLE_LEAFING_ERROR As Integer = -85
Public Const LS_INVALID_BADGE_TIMEOUT As Integer = -86
Public Const LS_INVALID_RESET_TYPE As Integer = -87
Public Const LS_MISSING_SET_CALLBACK As Integer = -88
Public Const LS_IMAGE_NOT_200_DPI As Integer = -89
Public Const LS_DOWNLOAD_ERROR As Integer = -90
Public Const LS_INVALID_SORT_ON_CHOICE As Integer = -91
Public Const LS_INVALID_FONT As Integer = -92
Public Const LS_INVALID_UNIT_SPEED As Integer = -93
Public Const LS_INVALID_LENGTH As Integer = -94
Public Const LS_SHORT_PAPER As Integer = -95
Public Const LS_INVALID_DOC_LENGTH As Integer = -96
Public Const LS_INVALID_DOCSLONG As Integer = -97
Public Const LS_IMAGE_NOT_256_COLOR As Integer = -98
Public Const LS_BATTERY_NOT_CHARGED As Integer = -99
Public Const LS_INVALID_SCAN_DOC_TYPE As Integer = -100
Public Const LS_ILLEGAL_SCAN_CARD_SPEED As Integer = -101
Public Const LS_INVALID_PWM_VALUE As Integer = -102
Public Const LS_INVALID_KEY_LENGTH As Integer = -103
Public Const LS_INVALID_PASSWORD As Integer = -104
Public Const LS_UNIT_LOCKED As Integer = -105
Public Const LS_INVALID_IMAGEFORMAT As Integer = -106
Public Const LS_INVALID_THRESHOLD As Integer = -107
Public Const LS_NO_START_FOR_SORTER_FULL As Integer = -108
Public Const LS_IPBOX_ADDRESS_NOT_FOUNDED As Integer = -109
Public Const LS_INVALID_LED_COMMAND As Integer = -110
Public Const LS_INVALID_COLOR_PARAMETER As Integer = -111

Public Const LS_JAM_AT_MICR_PHOTO As Integer = -201
Public Const LS_JAM_DOC_TOO_LONG As Integer = -202
Public Const LS_JAM_AT_SCANNER_PHOTO As Integer = -203

Public Const LS_SCAN_NETTO_IMAGE_NOT_SUPPORTED As Integer = -521
Public Const LS_256_GRAY_NOT_SUPPORTED As Integer = -522
Public Const LS_INVALID_PATH As Integer = -523
Public Const LS_MISSING_CALLBACK_FUNCTION As Integer = -526
Public Const LS_INVALID_OCR_IMAGE_SIDE As Integer = -558
Public Const LS_PERIPHERAL_NOT_ANSWER As Integer = -599

Public Const LS_INVALID_CONNECTION_HANDLE As Integer = -1000
Public Const LS_INVALID_CONNECT_PERIPHERAL As Integer = -1001
Public Const LS_PERIPHERAL_NOT_YET_INTEGRATE As Integer = -1002
Public Const LS_UNKNOW_PERIPHERAL_REPLY As Integer = -1003
Public Const LS_CODELINE_ALREADY_DEFINED As Integer = -1004
Public Const LS_INVALID_NUMBER_OF_DOC As Integer = -1005
Public Const LS_DECODE_FONT_NOT_PRESENT As Integer = -1101
Public Const LS_DECODE_INVALID_COORDINATE As Integer = -1102
Public Const LS_DECODE_INVALID_OPTION As Integer = -1103
Public Const LS_DECODE_INVALID_CODELINE_TYPE As Integer = -1104
Public Const LS_DECODE_SYSTEM_ERROR As Integer = -1105
Public Const LS_DECODE_DATA_TRUNC As Integer = -1106
Public Const LS_DECODE_INVALID_BITMAP As Integer = -1107
Public Const LS_DECODE_ILLEGAL_USE As Integer = -1108

Public Const LS_BARCODE_GENERIC_ERROR As Integer = -1201
Public Const LS_BARCODE_NOT_DECODABLE As Integer = -1202
Public Const LS_BARCODE_OPENFILE_ERROR As Integer = -1203
Public Const LS_BARCODE_READBMP_ERROR As Integer = -1204
Public Const LS_BARCODE_MEMORY_ERROR As Integer = -1205
Public Const LS_BARCODE_START_NOTFOUND As Integer = -1206
Public Const LS_BARCODE_STOP_NOTFOUND As Integer = -1207

Public Const LS_PDF_NOT_DECODABLE As Integer = -1301
Public Const LS_PDF_READBMP_ERROR As Integer = -1302
Public Const LS_PDF_BITMAP_FORMAT_ERROR As Integer = -1303
Public Const LS_PDF_MEMORY_ERROR As Integer = -1304
Public Const LS_PDF_START_NOTFOUND As Integer = -1305
Public Const LS_PDF_STOP_NOTFOUND As Integer = -1306
Public Const LS_PDF_LEFTIND_ERROR As Integer = -1307
Public Const LS_PDF_RIGHTIND_ERROR As Integer = -1308
Public Const LS_PDF_OPENFILE_ERROR As Integer = -1309


' -------------------------------------------------------------------
'                  WARNINGS
' -------------------------------------------------------------------
Public Const LS_FEEDER_EMPTY As Integer = 1
Public Const LS_DATA_TRUNCATED As Integer = 2
Public Const LS_DOC_PRESENT As Integer = 3
Public Const LS_BADGE_TIMEOUT As Integer = 4
Public Const LS_ALREADY_OPEN As Integer = 5
Public Const LS_PERIPHERAL_BUSY As Integer = 6
Public Const LS_DOUBLE_LEAFING_WARNING As Integer = 7
Public Const LS_COMMAND_NOT_ENDED As Integer = 8
Public Const LS_RETRY As Integer = 9
Public Const LS_NO_OTHER_DOCUMENT As Integer = 10
Public Const LS_QUEUE_FULL As Integer = 11
Public Const LS_NO_SENSE As Integer = 12
Public Const LS_TRY_TO_RESET As Integer = 14
Public Const LS_STRING_TRUNCATED As Integer = 15
Public Const LS_COMMAND_NOT_SUPPORTED As Integer = 19
Public Const LS_SORTER1_FULL As Integer = 35
Public Const LS_SORTER2_FULL As Integer = 36
Public Const LS_SORTERS_BOTH_FULL As Integer = 37
Public Const LS_KEEP_DOC_ON_CODELINE_ERROR As Integer = 39
Public Const LS_LOOP_INTERRUPTED As Integer = 40

Public Const LS_SORTER_1_POCKET_1_FULL As Integer = 51
Public Const LS_SORTER_1_POCKET_2_FULL As Integer = 52
Public Const LS_SORTER_1_POCKET_3_FULL As Integer = 53
Public Const LS_SORTER_2_POCKET_1_FULL As Integer = 54
Public Const LS_SORTER_2_POCKET_2_FULL As Integer = 55
Public Const LS_SORTER_2_POCKET_3_FULL As Integer = 56
Public Const LS_SORTER_3_POCKET_1_FULL As Integer = 57
Public Const LS_SORTER_3_POCKET_2_FULL As Integer = 58
```

```
        Public Const LS_SORTER_3_POCKET_3_FULL As Integer = 59
        Public Const LS_SORTER_4_POCKET_1_FULL As Integer = 60
        Public Const LS_SORTER_4_POCKET_2_FULL As Integer = 61
        Public Const LS_SORTER_4_POCKET_3_FULL As Integer = 62
        Public Const LS_SORTER_5_POCKET_1_FULL As Integer = 63
        Public Const LS_SORTER_5_POCKET_2_FULL As Integer = 64
        Public Const LS_SORTER_5_POCKET_3_FULL As Integer = 65
        Public Const LS_SORTER_6_POCKET_1_FULL As Integer = 66
        Public Const LS_SORTER_6_POCKET_2_FULL As Integer = 67
        Public Const LS_SORTER_6_POCKET_3_FULL As Integer = 68
        Public Const LS_SORTER_7_POCKET_1_FULL As Integer = 69
        Public Const LS_SORTER_7_POCKET_2_FULL As Integer = 70
        Public Const LS_SORTER_7_POCKET_3_FULL As Integer = 71
    End Class
End Class



Private Sub Identify(ByVal model As int)
        Dim Reply As Integer
        Dim hConnect As Short

        Dim UnitCfg As Byte() = New Byte(3) {}
        Dim strLsModel As IntPtr = Marshal.AllocHGlobal(20)
        Dim strFwVersion As IntPtr = Marshal.AllocHGlobal(20)
        Dim Date_Fw As IntPtr = Marshal.AllocHGlobal(20)
        Dim strUnitID As IntPtr = Marshal.AllocHGlobal(20)
        Dim strInkJetVersion As IntPtr = Marshal.AllocHGlobal(20)
        Dim DecoderExpVersion As IntPtr = Marshal.AllocHGlobal(20)


        hConnect = 0

        Reply = CtsLs.LSConnect(0, 0, model, hConnect)
        If Reply = CtsLs.LsReply.LS_OKAY Then
            Dim strIdentify As [String]
            Dim blankOpt As String = "      "

            ' BoardVersion
            'FeederVersion,
            'SorterVersion,
            'Motorversion,
            'Reserved1,
            Reply = CtsLs.LSUnitIdentify(hConnect, 0, UnitCfg, strLsModel,
strFwVersion, Date_Fw, _
             strUnitID, IntPtr.Zero, DecoderExpVersion, strInkJetVersion,
IntPtr.Zero, IntPtr.Zero, _
             IntPtr.Zero, IntPtr.Zero, IntPtr.Zero)
            'Reserved2
            If Reply = CtsLs.LsReply.LS_OKAY Then
                strIdentify = "Model : " & Marshal.PtrToStringAnsi(strLsModel) &
vbLf & "FW version : " & Marshal.PtrToStringAnsi(strFwVersion) & vbLf & "FW date
: " & Marshal.PtrToStringAnsi(Date_Fw) & vbLf & vbLf & "Serial #: " &
Marshal.PtrToStringAnsi(strUnitID) & vbLf & vbLf & "Options :" & vbLf

                If Marshal.PtrToStringAnsi(strLsModel).Contains("LS40") = True
Then
                    If (UnitCfg(0) And &H1) = &H1 Then
                        strIdentify += blankOpt & "MICR reader" & vbLf
                    End If
                    If (UnitCfg(0) And &H2) = &H2 Then
                        strIdentify += blankOpt & "Card Processing" & vbLf
                    End If
                    If (UnitCfg(0) And &H8) = &H8 Then
                        strIdentify += blankOpt & "Endorsement Ink-jet printer"
& vbLf
                    End If
                    If (UnitCfg(0) And &H10) = &H10 Then
                        strIdentify += blankOpt & "USB Powered" & vbLf
                    End If
                    If (UnitCfg(0) And &H20) = &H20 Then
                        strIdentify += blankOpt & "Voiding front stamp" & vbLf
                    End If

                    If (UnitCfg(1) And &H1) = &H1 Then
                        strIdentify += blankOpt & "Scanner FRONT" & vbLf
                    End If
                    If (UnitCfg(1) And &H2) = &H2 Then
                        strIdentify += blankOpt & "Scanner REAR" & vbLf
                    End If
                    If (UnitCfg(1) And &H4) = &H4 Then
                        strIdentify += blankOpt & "Badge reader with tracks 1/2"
& vbLf
                    End If
                    If (UnitCfg(1) And &H8) = &H8 Then
                        strIdentify += blankOpt & "Badge reader with tracks 2/3"
& vbLf
                    End If
                    If (UnitCfg(1) And &H10) = &H10 Then
                        strIdentify += blankOpt & "Badge reader with tracks
1/2/3" & vbLf
                    End If
                ElseIf Marshal.PtrToStringAnsi(strLsModel).Contains("LS100") =
True Then
                    If (UnitCfg(1) And &H1) = &H1 Then
                        strIdentify += blankOpt & "MICR reader" & vbLf
                    End If
                    If (UnitCfg(1) And &H2) = &H2 Then
                        strIdentify += blankOpt & "OCR reader" & vbLf
                    End If
                    If (UnitCfg(1) And &H8) = &H8 Then
                        strIdentify += blankOpt & "Endorsement Ink-jet printer"
& vbLf
                    End If
                    If (UnitCfg(1) And &H10) = &H10 Then
                        strIdentify += blankOpt & "Feeder " & vbLf
                    End If
                    If (UnitCfg(1) And &H20) = &H20 Then
                        strIdentify += blankOpt & "Voiding front stamp" & vbLf
                    End If

                    If (UnitCfg(2) And &H1) = &H1 Then
                        strIdentify += blankOpt & "Scanner FRONT" & vbLf
                    End If
                    If (UnitCfg(2) And &H2) = &H2 Then
                        strIdentify += blankOpt & "Scanner REAR" & vbLf
                    End If
                    If (UnitCfg(2) And &H4) = &H4 Then
                        If (UnitCfg(2) And &H10) = &H10 Then
```

```
                            strIdentify += blankOpt & "Badge reader with tracks
1/2/3" & vbLf
                        Else
                            If (UnitCfg(2) And &H8) = &H8 Then
                                strIdentify += blankOpt & "Badge reader with
tracks 1/2" & vbLf
                            Else
                                strIdentify += blankOpt & "Badge reader with
tracks 2/3" & vbLf
                            End If
                        End If
                    End If
                ElseIf Marshal.PtrToStringAnsi(strLsModel).Contains("LS150") =
True Then
                    If (UnitCfg(0) And &H1) = &H1 Then
                        strIdentify += blankOpt & "MICR reader" & vbLf
                    End If
                    If (UnitCfg(0) And &H2) = &H2 Then
                        strIdentify += blankOpt & "Unit set in Normal Speed" &
vbLf
                    Else
                        strIdentify += blankOpt & "Unit set in High Speed" &
vbLf
                    End If
                    If (UnitCfg(0) And &H4) = &H4 Then
                        strIdentify += blankOpt & "Feeder Motorized" & vbLf
                    End If
                    If (UnitCfg(0) And &H8) = &H8 Then
                        If (UnitCfg(2) And &H8) = &H8 Then
                            strIdentify += blankOpt & "High Definition Ink-jet
printer" & vbLf
                        Else
                            strIdentify += blankOpt & "Endorsement Ink-jet
printer" & vbLf
                        End If
                    End If
                    If (UnitCfg(0) And &H10) = &H10 Then
                        strIdentify += blankOpt & "Feeder with Electromagnet 50
Doc." & vbLf
                    End If
                    If (UnitCfg(0) And &H20) = &H20 Then
                        strIdentify += blankOpt & "Voiding front stamp" & vbLf
                    End If

                    If (UnitCfg(1) And &H4) = &H4 Then
                        strIdentify += blankOpt & "Scanner FRONT with Ultra
Violet" & vbLf
                    ElseIf (UnitCfg(1) And &H1) = &H1 Then
                        strIdentify += blankOpt & "Scanner FRONT" & vbLf
                    End If
                    If (UnitCfg(1) And &H2) = &H2 Then
                        strIdentify += blankOpt & "Scanner REAR" & vbLf
                    End If
                    If (UnitCfg(1) And &H20) = &H20 Then
                        strIdentify += blankOpt & "COLOR version" & vbLf
                    End If
                    If (UnitCfg(1) And &H8) = &H8 Then
                        If (UnitCfg(1) And &H10) = &H10 Then
                            strIdentify += blankOpt & "Badge reader with tracks
1/2/3" & vbLf
```

```
                        Else
                            strIdentify += blankOpt & "Badge reader with tracks
2/3" & vbLf
                        End If
                    ElseIf (UnitCfg(1) And &H10) = &H10 Then
                        strIdentify += blankOpt & "Badge reader with tracks 1/2"
& vbLf
                    End If
                ElseIf Marshal.PtrToStringAnsi(strLsModel).Contains("LS515") =
True Then
                    If (UnitCfg(1) And &H1) = &H1 Then
                        strIdentify += blankOpt & "CMC7 reader" & vbLf
                    End If
                    If (UnitCfg(1) And &H1) = &H2 Then
                        strIdentify += blankOpt & "E13B reader" & vbLf
                    End If
                    If (UnitCfg(1) And &H8) = &H8 Then
                        strIdentify += blankOpt & "Endorsement Ink-jet printer"
& vbLf
                    End If
                    If (UnitCfg(1) And &H20) = &H20 Then
                        strIdentify += blankOpt & "Voiding front stamp" & vbLf
                    End If

                    If (UnitCfg(2) And &H1) = &H1 Then
                        If (UnitCfg(2) And &HC) = &H4 Then
                            strIdentify += blankOpt & "Scanner FRONT with Ultra
Violet" & vbLf
                        Else
                            strIdentify += blankOpt & "Scanner FRONT" & vbLf
                        End If
                    End If
                    If (UnitCfg(2) And &H2) = &H2 Then
                        strIdentify += blankOpt & "Scanner REAR" & vbLf
                    End If
                    If (UnitCfg(1) And &H10) = &H10 Then
                        strIdentify += blankOpt & "Badge reader" & vbLf
                    End If
                    If (UnitCfg(1) And &H20) = &H20 Then
                        strIdentify += blankOpt & "Double Leafing sensor" & vbLf
                    End If
                End If

                MessageBox.Show(strIdentify, TITLE_POPUP)
            End If

            Reply = CtsLs.LSDisconnect(hConnect, 0)
        Else
            CheckReply(Reply, "LSConnect")
        End If

    End Sub



Private Function DoAutoDocHandle(ByVal hConnect As Short, ByVal UnitCfg As
Byte()) As Integer
        Dim Reply As Integer
        '   string      FileOut;
        Dim BufFrontFile As IntPtr = Marshal.AllocHGlobal(1024)
        Dim BufRearFile As IntPtr = Marshal.AllocHGlobal(1024)
        Dim BufFrontImage As IntPtr
        Dim BufRearImage As IntPtr
        Dim BufFrontUVImage As IntPtr
        Dim BufFrontGrayUVImage As IntPtr
        Dim NoImage As IntPtr
        Dim BufCodelineSW As IntPtr =
Marshal.AllocHGlobal(CInt(CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH))
        Dim BufCodelineHW As IntPtr =
Marshal.AllocHGlobal(CInt(CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH))
        Dim BufBarcode As IntPtr =
Marshal.AllocHGlobal(CInt(CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH))
        Dim CodelineType As CtsLs.CodeLineType
        Dim C_x As Single
        Dim C_y As Single
        Dim C_w As Single
        Dim C_h As Single

        '           int     ImageBW;
        Dim DocToRead As Short
        Dim NrPrinted As Int32

        Dim DocToProcess As Int32
        Dim NrDoc As UInt32
        Dim NrCheque As Short = 1
        Dim PrintValidate As CtsLs.PrintValidate
        Dim dirBase As IntPtr
        Dim Filename As IntPtr

        'FILE  *fhCodeline = NULL;

        ' COORDINATE PER BARCODE O PER OCR
        Dim TypeOfDecod As Byte



        Reply = CtsLs.LSConnect(0, 0, model, hConnect)
        If Reply <> CtsLs.LsReply.LS_OKAY Then
           // Error Handling
           MessageBox.Show("Open not done: error " + Reply.ToString());
           return Reply;
        End If

        PrintValidate = CtsLs.PrintValidate.NO_PRINT_VALIDATE
        '-----------LoadString--------------------------------------
        If stParAppl.PrintValidate <> CtsLs.PrintFont.PRINT_NO_STRING Then
            Dim strEndorse As IntPtr = Marshal.AllocHGlobal(160)



            ' Copy the Secure string to unmanaged memory (and decrypt it).
            strEndorse = Marshal.StringToHGlobalAnsi(stParAppl.Endorse_str)

            If stParAppl.Endorse_str.Contains("%d") Then
                Reply = CtsLs.LSLoadStringWithCounterEx(hConnect, 0,
CShort(stParAppl.PrintValidate), strEndorse,
CShort(stParAppl.Endorse_str.Length), 8, _
                 3)
            Else
                Reply = CtsLs.LSLoadString(hConnect, 0,
CShort(stParAppl.PrintValidate), CShort(stParAppl.Endorse_str.Length),
strEndorse)
            End If
            If Reply <> CtsLs.LsReply.LS_OKAY Then
                If CheckReply(Reply, "LSLoadString") Then
                    Return Reply
                End If
            End If

            ' Set variable for print
            PrintValidate = CtsLs.PrintValidate.PRINT_VALIDATE
            'PrintValidate = CtsLs.PrintValidate.PRINT_VALIDATE;
            'Reply = CtsLs.LSLoadString(hConnect, 0, PRINT_FORMAT_HEAD_TEST, 1,
" ");

            'if (Reply != CtsLs.LsReply.LS_OKAY)
            '{
            '    if (CheckReply(Reply, "LSLoadString"))
            '    {
            '        return Reply;
            '    }
            '}
        End If



        '----------- Set Sensibilità foto doppia sfogliatura -----------
        Reply = CtsLs.LSConfigDoubleLeafingAndDocLength(hConnect, 0,
stParAppl.DL_Type, stParAppl.DL_Value, stParAppl.DL_MinDoc, stParAppl.DL_MaxDoc)
        If Reply <> CtsLs.LsReply.LS_OKAY Then
            If CheckReply(Reply, "LSConfigDoubleLeafingAndDocLength") Then
                Return Reply
            End If
        End If


        '----------- Change stamp position -----------
        'Reply = CtsLs.LSChangeStampPosition(hConnect, 0,
stParAppl.StampPosition, 0);
        'if (Reply != CtsLs.LsReply.LS_OKAY)
        '{
        '    if (CheckReply(Reply, "LSChangeStampPosition"))
        '    {
        '        return Reply;
        '    }
        '}


        '----------- Set attesa introduzione documento -----------
        Reply = CtsLs.LSDisableWaitDocument(hConnect, 0, stParAppl.WaitTimeout)
        If Reply <> CtsLs.LsReply.LS_OKAY Then
            If CheckReply(Reply, "LSDisableWaitDocument") Then
                Return Reply
            End If
        End If


        '----------- Set speed document -----------
        Reply = CtsLs.LSSetUnitSpeed(hConnect, 0, stParAppl.LowSpeed)
        If Reply <> CtsLs.LsReply.LS_OKAY Then
            If CheckReply(Reply, "LSSetSpeedUnit") Then
                Return Reply
            End If
        End If


        '----------- Set Light Intensity --------------------------
        Reply = CtsLs.LSSetLightIntensity(hConnect, 0, stParAppl.LightIntensity)
        If Reply <> CtsLs.LsReply.LS_OKAY Then
            If CheckReply(Reply, "LSSetLightIntensity") Then
                Return Reply
            End If
        End If


        '----------- Only for Ultra Violet type -----------
        If (UnitCfg(1) And MASK_SCANNER_UV) = MASK_SCANNER_UV Then
            Reply = CtsLs.LSModifyPWMUltraViolet(hConnect, 0,
stParAppl.PercentPWM_UV, stParAppl.Contrast_UV, stParAppl.Threshold_UV)
            If Reply <> CtsLs.LsReply.LS_OKAY Then
                If CheckReply(Reply, "LSModifyPWMUltraViolet") Then
                    Return Reply
                End If
            End If
        End If


        ' Settaggio dei parametri fissi per LS_AutoDocHandle
        dirBase = Marshal.StringToHGlobalAnsi(PathAppl + SAVE_DIRECTORY_IMAGE)
        Filename = Marshal.StringToHGlobalAnsi(NAME_IMAGE)

        CodelineType = CtsLs.CodeLineType.NO_READ_CODELINE
        TypeOfDecod = stParAppl.TypeOfDecod
        C_x = C_y = C_w = C_h = 0
        If stParAppl.CodelineMICR <> CtsLs.CodeLineType.NO_READ_CODELINE Then
            CodelineType = stParAppl.CodelineMICR
        ElseIf stParAppl.CodelineOCR <> CtsLs.CodeLineType.NO_READ_CODELINE Then
            CodelineType = stParAppl.CodelineOCR
            C_x = stParAppl.Codeline_Sw_x
            C_y = stParAppl.Codeline_Sw_y
            C_w = stParAppl.Codeline_Sw_w
            C_h = stParAppl.Codeline_Sw_h
            TypeOfDecod -= DECODE_OCR
        ElseIf stParAppl.BarcodeType <> CtsLs.CodeLineType.NO_READ_CODELINE Then
            CodelineType = stParAppl.BarcodeType
            C_x = stParAppl.Barcode_Sw_x
            C_y = stParAppl.Barcode_Sw_y
            C_w = stParAppl.Barcode_Sw_w
            C_h = stParAppl.Barcode_Sw_h
            TypeOfDecod -= DECODE_OCR
        End If

        Reply = CtsLs.LSAutoDocHandle(hConnect, 0, stParAppl.FrontStamp,
CShort(PrintValidate), CShort(CodelineType), stParAppl.ScanMode, _
         CShort(CtsLs.Feeder.FEED_AUTO), CShort(CtsLs.Sorter.SORTER_POCKET_1),
stParAppl.NumDoc, stParAppl.ClearAlignImage, stParAppl.Side, 0, _
         stParAppl.SaveImage, dirBase, Filename, C_x, C_y, C_w, _
         C_h, CShort(CtsLs.Unit.UNIT_MM), 0, stParAppl.FileFormat,
stParAppl.Qual, CShort(CtsLs.FileAttribute.SAVE_OVERWRITE), _
         1, CShort(If(stParAppl.WaitTimeout, CtsLs.Wait.WAIT_YES,
CtsLs.Wait.WAIT_NO)), stParAppl.BeepOnError, 0, IntPtr.Zero, IntPtr.Zero)
        If Reply <> CtsLs.LsReply.LS_OKAY Then
            If CheckReply(Reply, "LSAutoDocHandle") Then
                Return Reply
            End If
        End If


        ' Salvo il nr. di documenti da processare
        If stParAppl.NumDoc <> 0 Then
            DocToProcess = stParAppl.NumDoc
        Else
            DocToProcess = 1000000
        End If


        If Save_FrontImage <> IntPtr.Zero Then
            CtsLs.LSFreeImage(0, Save_FrontImage)
            Save_FrontImage = IntPtr.Zero
        End If
        If Save_RearImage <> IntPtr.Zero Then
            CtsLs.LSFreeImage(0, Save_RearImage)
            Save_RearImage = IntPtr.Zero
        End If

        '-----------GetDocData--------------------------------------
        Dim pBitmap As Bitmap = Nothing

        '   if( Reply == LS_OKAY )
        '   {
        '         while( TRUE )
        While Reply = CtsLs.LsReply.LS_OKAY
            Marshal.WriteByte(BufFrontFile, 0)
            Marshal.WriteByte(BufRearFile, 0)
            BufFrontImage = IntPtr.Zero
            BufRearImage = IntPtr.Zero
            BufFrontUVImage = BufFrontGrayUVImage = IntPtr.Zero
            NoImage = IntPtr.Zero
            NrDoc = 0
            DocToRead = 0
            NrPrinted = 0

            Marshal.WriteByte(BufCodelineSW, 0)
            Marshal.WriteByte(BufCodelineHW, 0)
            Marshal.WriteByte(BufBarcode, 0)

            'BufFrontNettoFile,
            'BufBackNettoFile,
            'BufbackNettoImage,
            Reply = CtsLs.LSGetDocData(hConnect, 0, NrDoc, BufFrontFile,
BufRearFile, IntPtr.Zero, _
             IntPtr.Zero, BufFrontImage, BufRearImage, BufFrontUVImage, NoImage,
BufCodelineSW, _
             BufCodelineHW, BufBarcode, IntPtr.Zero, DocToRead, NrPrinted,
IntPtr.Zero, _
             IntPtr.Zero)
            ' Se e` doppia sfogliatura non termino
            If ((Reply <> CtsLs.LsReply.LS_OKAY) AndAlso (Reply <>
CtsLs.LsReply.LS_SORTER1_FULL) AndAlso (Reply <>
CtsLs.LsReply.LS_DOUBLE_LEAFING_WARNING)) Then
                If Reply <> CtsLs.LsReply.LS_FEEDER_EMPTY Then
                    CheckReply(Reply, "LS_GetDocData")
                End If
                Exit While
            End If


            'metto il test per le altre codeline
            If (TypeOfDecod And DECODE_OCR) = DECODE_OCR Then
                Dim ro As CtsLs.READOPTIONS
                Dim CodelineOpt As Byte() = New Byte(3) {}
                Dim len_codeline As Integer

                ro.PutBlanks = 1
                ro.TypeRead = "N"c
                If stParAppl.CodelineOCR =
CtsLs.CodeLineType.READ_CODELINE_SW_E13B_X_OCRB Then
                    ro.TypeRead = "X"c
                    CodelineOpt(0) =
CByte(CtsLs.CodeLineType.READ_CODELINE_SW_E13B)
                    CodelineOpt(1) =
CByte(CtsLs.CodeLineType.READ_CODELINE_SW_OCRB_ITALY)
                    CodelineOpt(2) = CByte(AscW(ControlChars.NullChar))
                Else
                    CodelineOpt(0) = CByte(stParAppl.CodelineOCR)
                    CodelineOpt(2) = CByte(AscW(ControlChars.NullChar))
                End If

                len_codeline = CInt(CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH)

                Reply = CtsLs.LSCodelineReadFromBitmap(0, BufFrontImage,
CodelineOpt, stParAppl.Unit_measure, stParAppl.Codeline_Sw_x,
stParAppl.Codeline_Sw_y, _
                 stParAppl.Codeline_Sw_w, stParAppl.Codeline_Sw_h, ro,
BufCodelineSW, len_codeline)
                If Reply <> CtsLs.LsReply.LS_OKAY Then
                    CheckReply(Reply, "LSCodelineReadFromBitmap")
                    ' Set Ok for not exit from the loop
                    Reply = CtsLs.LsReply.LS_OKAY
                End If
            End If
            If (TypeOfDecod And DECODE_BARCODE) = DECODE_BARCODE Then
                Dim len_barcode As Integer =
CInt(CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH)

                Reply = CtsLs.LSReadBarcodeFromBitmap(0, BufFrontImage,
CByte(stParAppl.BarcodeType), CInt(stParAppl.Barcode_Sw_x),
CInt(stParAppl.Barcode_Sw_y), CInt(stParAppl.Barcode_Sw_w), _
                 CInt(stParAppl.Barcode_Sw_h), BufBarcode, len_barcode)
                If Reply <> CtsLs.LsReply.LS_OKAY Then
                    CheckReply(Reply, "LSReadBarcodeFromBitmap")
                    ' Set Ok for not exit from the loop
                    Reply = CtsLs.LsReply.LS_OKAY
                End If
            End If
            If (TypeOfDecod And DECODE_PDF417) = DECODE_PDF417 Then
```

```
                Dim len_barcode As Integer =
CInt(CtsLs.CodeLineType.MAX_CODE_LINE_LENGTH)

                Reply = CtsLs.LSReadPdf417FromBitmap(0, BufFrontImage,
BufBarcode, len_barcode, 0, 0, _
                 0, 0, 0)
                If Reply <> CtsLs.LsReply.LS_OKAY Then
                    CheckReply(Reply, "LSReadPdf417FromBitmap")
                    ' Set Ok for not exit from the loop
                    Reply = CtsLs.LsReply.LS_OKAY
                End If
            End If


            If stParAppl.ScanMode =
CShort(CtsLs.ScanMode.SCAN_MODE_256GR100_AND_UV) OrElse stParAppl.ScanMode =
CShort(CtsLs.ScanMode.SCAN_MODE_256GR200_AND_UV) OrElse stParAppl.ScanMode =
CShort(CtsLs.ScanMode.SCAN_MODE_256GR300_AND_UV) Then
                ' Build the mergered gray UV image
                If BufFrontUVImage <> IntPtr.Zero Then
                    'stParAppl.Threshold_UV,


                    'if( stParAppl.SaveImage == IMAGE_SAVE_ON_FILE ||
stParAppl.SaveImage == IMAGE_SAVE_BOTH )
                    '{
                    '    // Check if I must save the image
                    '    if( stParAppl.FileFormat ==
(short)CtsLs.FileType.FILE_JPEG )
                    '    {
                    '        FullFName, "%s\\%s%04dGUV.jpg", dirBase, Filename,
(NrDoc - 1));
                    '        Reply = LSSaveJPEG(hWnd, BufFrontGrayUVImage,
stParAppl.Qual, FullFName);
                    '    }
                    '    else if( stParAppl.FileFormat ==
(short)CtsLs.FileType.FILE_BMP )
                    '    {
                    '        sprintf(FullFName, "%s\\%s%04dGUV.bmp", dirBase,
Filename, (NrDoc - 1));
                    '        Reply = LSSaveDIB(hWnd, BufFrontGrayUVImage,
FullFName);
                    '    }
                    '    else if( stParAppl.FileFormat ==
(short)CtsLs.FileType.FILE_CCITT_GROUP4 )
                    '    {
                    '        sprintf(FullFName, "%s\\%s%04dGUV.tif", dirBase,
Filename, (NrDoc - 1));
                    '        Reply = LSSaveTIFF(hWnd, BufFrontGrayUVImage,
FullFName, FILE_TIF, SAVE_OVERWRITE, 1);
                    '    }
                    '}
                    CtsLs.LSMergeImageGrayAndUV(0, BufFrontImage,
BufFrontUVImage, 0, 0, BufFrontGrayUVImage)
                End If
            End If


            ' Show Codeline
            If Marshal.ReadByte(BufCodelineHW) <> 0 Then
```

```
                tbCodeline.Text = Marshal.PtrToStringAnsi(BufCodelineHW)
            Else
                tbCodeline.Text = ""
            End If

            ' Show the immage
            If Form1.stParAppl.Side = CByte(CtsLs.Side.SIDE_ALL_IMAGE) OrElse
Form1.stParAppl.Side = CByte(CtsLs.Side.SIDE_FRONT_IMAGE) Then
                Dim pBmp As CtsLs.BITMAPINFOHEADER =
Marshal.PtrToStructure(BufFrontImage, GetType(CtsLs.BITMAPINFOHEADER))

                pBitmap = New Bitmap(pBmp.biWidth, pBmp.biHeight,
PixelFormat.Format24bppRgb)

'               BitmapData bmpData = pBitmap.LockBits(new Rectangle(0, 0,
pBitmap.Width, pBitmap.Height), ImageLockMode.WriteOnly, pBitmap.PixelFormat);
                Dim xx As Integer, yy As Integer
                Dim diff As Int32, WidthBytes As Int32 = pBmp.biWidth
                diff = WidthBytes Mod 4
                If diff <> 0 Then
                    WidthBytes += (4 - diff)
                End If
                Dim row As Int32, col As Int32
                row = pBmp.biHeight - 1
                col = pBmp.biWidth - 1
                For yy = 0 To pBmp.biHeight - 1
                    For xx = 0 To pBmp.biWidth - 1
                        Dim Pixel As Byte = Marshal.ReadByte(CInt(BufFrontImage)
+ 1064 + ((yy * WidthBytes) + xx))
                        Dim Pixel24 As Color
                        Pixel24 = Color.FromArgb((Pixel * 256 * 256) + (Pixel *
256) + Pixel)
                        pBitmap.SetPixel(xx, (row - yy), Pixel24)
                    Next
                Next

                If pbImage.Image IsNot Nothing Then
                    '    //pbImage.Image = null;
                    pbImage.Image.Dispose()
                End If
                pbImage.Image = pBitmap
            End If

            ' Refresh the form
            Application.DoEvents()

            ' Enable the button for show the rear
            If Form1.stParAppl.Side = CByte(CtsLs.Side.SIDE_ALL_IMAGE) Then
                btRear.Enabled = True
            End If

            ' Free the aree
            '                pBitmap.Dispose();


            ' Free the previous image memory and save the current
            If Save_FrontImage <> IntPtr.Zero Then
                CtsLs.LSFreeImage(0, Save_FrontImage)
            End If
            If BufFrontGrayUVImage <> IntPtr.Zero Then
```

```
                Save_FrontImage = BufFrontGrayUVImage
                CtsLs.LSFreeImage(0, BufFrontImage)
                CtsLs.LSFreeImage(0, BufFrontUVImage)
            ElseIf BufFrontImage <> IntPtr.Zero Then
                Save_FrontImage = BufFrontImage
            End If

            If Save_RearImage <> IntPtr.Zero Then
                CtsLs.LSFreeImage(0, Save_RearImage)
            End If
            Save_RearImage = BufRearImage


            If (System.Threading.Interlocked.Decrement(DocToProcess)) = 0 Then
                Exit While
            End If


            NrCheque += 1
        End While
        ' Fine while( TRUE )
        ' Free of local variable
        Marshal.FreeHGlobal(Filename)
        Marshal.FreeHGlobal(dirBase)
        Marshal.FreeHGlobal(BufBarcode)
        Marshal.FreeHGlobal(BufCodelineHW)
        Marshal.FreeHGlobal(BufCodelineSW)
        Marshal.FreeHGlobal(BufRearFile)
        Marshal.FreeHGlobal(BufFrontFile)

        Return Reply
    End Function
```

15. Reply codes

LS functions reply codes are coded as requested by environment standard and all the possible reply codes are identified by a symbolic name as shown by the table below.

Symbolic Name	Class	Description	Possible Reasons
LS_OKAY		Command correctly executed	
LS_SYSTEM_ERROR	E	The service was unable to execute command due to a system error	System allocation memory error
LS_USB_ERROR	E	The service was unable to execute a command due to a USB error	Error condition coming from USB Manager or USB exchange data
LS_PERIPHERAL_NOT_FOUND	E	Peripheral not found by the service	Peripheral is switched off or not connected to the USB port
LS_HARDWARE_ERROR	E	Peripheral hardware error	Peripheral hardware is not operating correctly
LS_PERIPHERAL_OFF_ON	E	Peripheral has been switched off and on again	The peripheral has been switched off and on again between the last and the current command.
LS_RESERVED_ERROR	E	Peripheral reservation error	Another program is using the peripheral.
LS_PAPER_JAM	E	Document jammed	Document jammed
LS_INVALID_COMMAND	E	Invalid command	The current command contains invalid parameters specification or is inconsistent with the previous command sequence
LS_DATA_LOST	E	Error during codeline data transfer	Application buffer not sufficient for contained all the character decoded
LS_COMMAND_IN_EXECUTION_YET	E	A command is already in execution	Command refused because other command has been launched
LS_COMMAND_SEQUENCE_ERROR	E	The command's sequence is not correct	The command is executed in wrong sequence.
LS_BARCODE_ERROR	E	Barcode string is in error.	The barcode string cannot be read.
LS_INVALID_HANDLE	E	The handle is invalid	The handle passed to image manipulation function like LSConvertImageToBW is invalid.
LS_NO_LIBRARY_LOAD	E	Library Img_util.dll and/or CtsDecod.dll	Install the ***complete software distribution kit*** or verified that the

		are not present or not correctly installed	library Img_util.dll and/or CtsDecod.dll are present in the same directory of the library LsApi.dll.
LS_JPEG_ERROR	E	JPEG image not created	Folder for store the images probably not found.
LS_BMP_ERROR	E	BMP image not created	Folder for store the images probably not found.
LS_TIFF_ERROR	E	TIFF image not created	Folder for store the images probably not found.
LS_ALREADY_OPEN	W	The peripheral is already connected	Do LS_Close, or use the other command.
LS_STRING_TRUNCATED	W	Buffer insufficient for contain the library release	Selected feeder (autofeeder or scan feeder) is empty.
LS_FEEDER_EMPTY	W	Feeder is empty	Selected feeder (autofeeder or scan feeder) is empty.
LS_DATA_TRUNCATED	W	Data truncated	Data transferred during command execution has been truncated because the destination buffer was too small.
LS_DOC_PRESENT	W	Document present	You close or open a document handling session with a document already present inside document handling path.
LS_DOUBLE_LEAFING_ERROR	W	Document double feeding	More than one document has been feed by autofeeder.
LS_DOUBLE_LEAFING_WARNING	W	Document double feeding	More than one document has been feed by autofeeder.
LS_SORTER1_FULL	W	Sorter 1 full	Too many documents in sorter one.
LS_SORTER2_FULL	W	Sorter 2 full	Too many documents in sorter two.
LS500_SORTERS_BOTH_FULL	W	Sorters 1 and 2 full	Too many documents in sorter one and two.
LS_BADGETIMEOUT	W	Badge read timeout	Timeout expired during read badge.
LS_NO_OTHER_DOCUMENT	W	There are not other document	The queue is empty and loop handle is finished.
LS_TRY_TO_RESET	W	Try to execute a Reset command	The peripheral is in error
LS_NO_SENSE	W	The peripheral is busy	Re-execute the command
LS_QUEUE_FULL	W	The document's queue is full.	To many documents is scanned.
LS_RETRY	W	Retry the GetDocData command.	There are other documents in the queue.