www.ultramagicard.com

Magicard API documentation

Application Programming Interface for Magicard Printer Drivers

Table of Contents

Overview 1
Sample Code 2
Typical Application Flowchart 2
API Functions 3
EnableStatusReporting 3
DisableStatusReporting 5
FeedCard 6
EjectCard 7
WaitForPrinter 8
GetLastPrinterMessage 9
GetLastEnduroMessage 10
GeneralCommand 11
GetPrinterStatus 12
GetPrinterInfo 13
SetEjectMode 15
EncodeMagStripe 16
ReadMagStripe 18
DecodeMagData 20
GetPrinterType 21
GetConnectionType 22
FlipCard 23
CleanPrinter 24
RestartPrinter 25
PrintTestCard 26
SetSmartMode 27
SetSmartLocation 28
EraseCard 29

SetEraseSpeed ... 30
GetAPIVersion ... 31
ErrorResponse ... 32
Deprecated Functions ... 33
RequestMagData ... 33
ReadMagData ... 34
GetEnduroInfo ... 35
Driver Validation ... 36

Figures

Figure 1 – Generic information flowchart ... 1
Figure 2 – Typical application flowchart ... 2

Overview

The purpose of this application programming interface (API) is to allow applications to:

- initiate printer operation by sending commands
- synchronize with the operation of the printer.
- interrogate the printer

For example, an application may need to know when the printer has finished printing a card. Without feedback from the printer, it is not possible to retrieve such information in an accurate manner due to the way the Windows print spooler works. With the API, this feedback is available and an application is able to wait until the printer has completed its operation.

Specific API functions are also available to control the printer e.g. place cards in the correct position for contact and contactless chip encoding and to eject cards from the printer.

The process of controlling the printer using the APIs **must** be a serial one – in other words, it is **not possible** to load the spooler with multiple prints and then control their flow to the printer by API calls, since these API calls themselves may need to pass commands to the printer via the spooler. In this situation, the commands would be placed in the spooler after the batch prints, and would be out of synchronization with the print they were trying to control, so control would be lost.

Therefore for each print job, the APIs should be used to control the card positioning, then the image data for that single card should be loaded to the spooler. The API is then used to position the next card for the next print job, and so on.

Generically speaking, the flow of information between the application and the status monitor is represented in the following diagram:

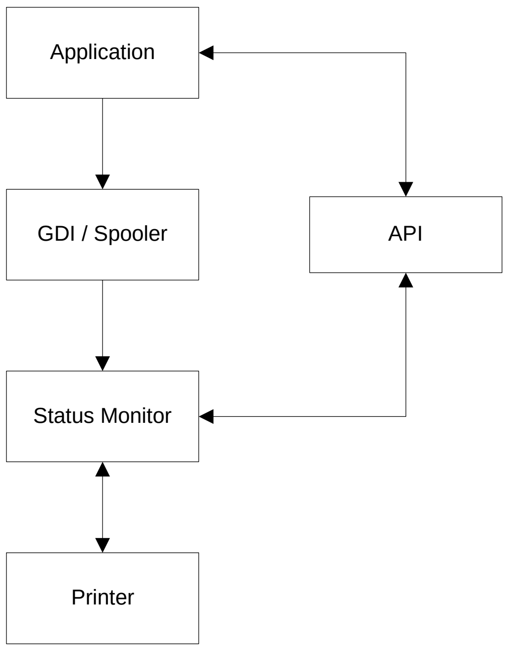

Figure 1 – Generic information flowchart

The API itself can be used by loading MagAPI.dll. A 'C' header file, a .def file and .lib static library file are also provided.

NB. The API is originally designed for USB communications and all timings are determined for this method of connection. Whilst it is possible to use ethernet, due to the variable nature of the timings using such comms, timings may vary considerably.

Sample Code

Sample code is available on request.

Typical Application Flowchart

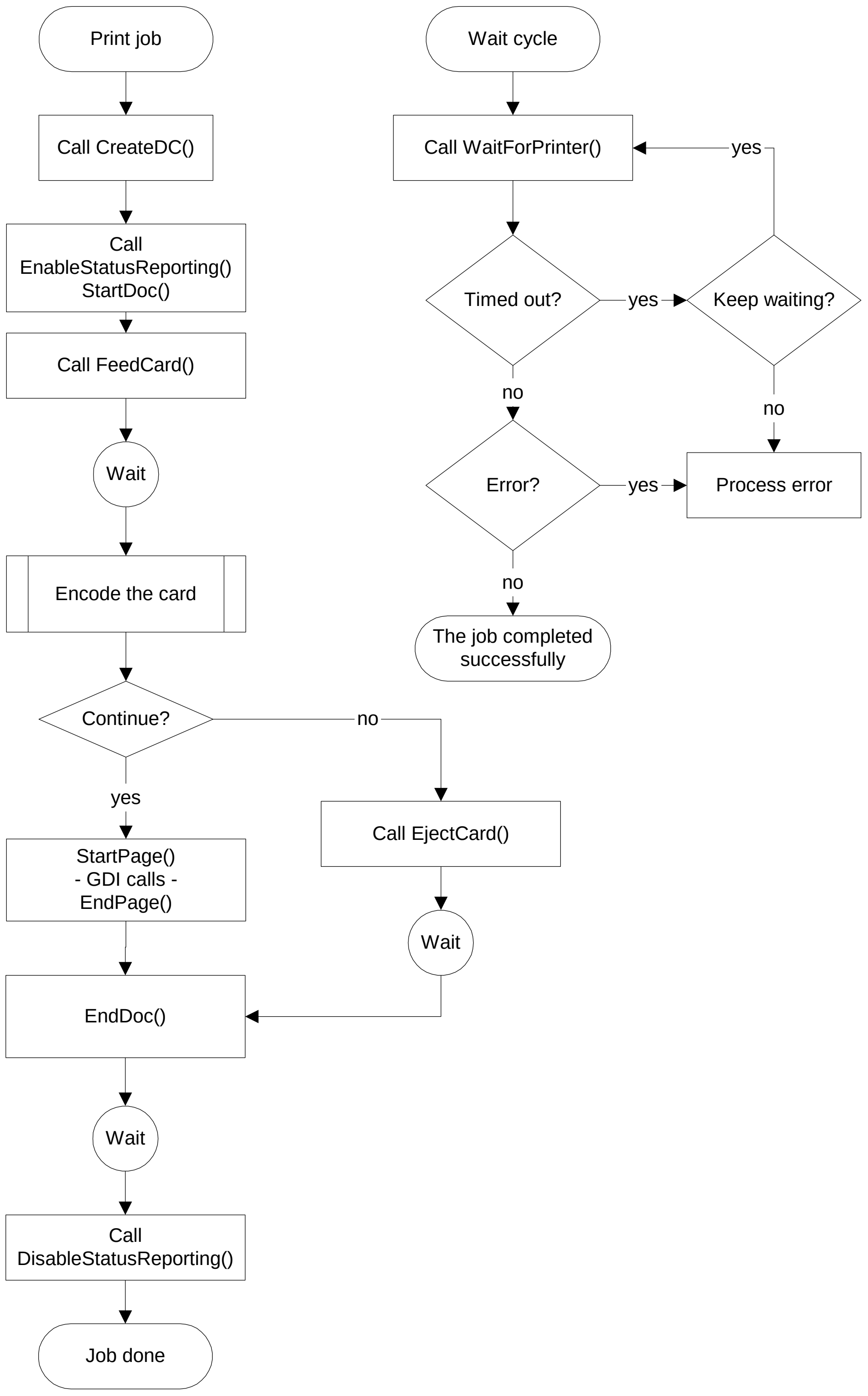

Figure 2 – Typical application flowchart

API Functions

EnableStatusReporting

Initialises the API and its communications channel with the printer and status monitor.

```
int EnableStatusReporting(HDC     hDC,
                          HANDLE  *phSession,
                          DWORD   dwFlags);
```

Parameters

hDC

A device context handle for the printer driver the application is using.

phSession

A pointer to a variable that will receive a handle that identifies the newly established session with the status monitor. This handle must be closed with *DisableStatusReporting*.

dwFlags

Defines how the status monitor will deal with all errors from now on. It can assume one of the following values:

(NB Only relevant on printers which support status monitor)

CONFIG_NORMAL	The status monitor will not change its current behaviour regarding printer errors
CONFIG_QUIET	No status monitor is displayed

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.
MAGICARD_DRIVER_NOTCOMPLIANT	This driver version is not supported.
MAGICARD_LOCALCOMM_ERROR	Failed to open the client communications pipe.
MAGICARD_REMOTECOMM_ERROR	Failed to open the monitor communications pipe.
MAGICARD_OPENPRINTER_ERROR	Failed to open the printer the DC belongs to.
MAGICARD_SPOOLER_NOT_EMPTY	There are print jobs queued for this printer instance.
MAGICARD_REMOTECOMM_IN_USE	The monitor communications pipe is already in use.
MAGICARD_LOCALCOMM_IN_USE	The client communications pipe is already in use.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

The print spooler must be empty before calling this function, since the status monitor's behaviour will already be different when the next print job begins. Otherwise, the function will fail with the MAGICARD_SPOOLER_NOT_EMPTY error code.

Failure to open any of the communications pipes usually means that the printer driver is configured to print to a port that is not supported.

It may also mean that the status monitor is not installed or that the system refuses to start it for some reason.

Alternatively, any of the two "in use" error codes usually mean that there is another application using the API already. The exact error code depends on the operating system and on the sequence of calls made by either application.

To help in determining if the API is indeed "in use", an inline function has been added – MAGICARD_Is_Status_Reporting_In_Use(int iError) – which returns TRUE if the error is either MAGICARD_REMOTECOMM_IN_USE or MAGICARD_LOCALCOMM_IN_USE.

DisableStatusReporting

Closes the communications channel with the printer, returns the status monitor to its normal behaviour and releases all resources used.

```
int DisableStatusReporting(HANDLE hSession);
```

Parameters

hSession
The session handle returned by *EnableStatusReporting*.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

FeedCard

Instructs the printer to feed a card to one of the chip encoding positions available.

```
int FeedCard(HANDLE    hSession,
             DWORD     dwMode,
             int       iParam,
             LPTSTR    lpszJobName);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

dwMode

The desired card position. It can be one of the following values:

FEED_CHIPCARD	Places the card in the contact chip encoding station.
FEED_CONTACTLESS	Places the card in range of the contactless chip encoder antenna.

iParam

An optional positive integer parameter that is to be appended to the end of the printer command used to feed the card.
If it is zero, nothing is appended. If it is positive, its value is used.

lpszJobName

The name of the secondary print job that is created by the API.

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

There are ANSI and UNICODE versions of this function, with A and W suffixes. A macro is conditionally defined in the header to point to the correct function version.

EjectCard

Instructs the printer to eject any card that may be present in the mechanism.

```
int EjectCard(HANDLE   hSession,
              LPTSTR   lpszJobName);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

lpszJobName

The name of the secondary print job that is created by the API when spooling is enabled.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

There are ANSI and UNICODE versions of this function, with A and W suffixes. A macro is conditionally defined in the header to point to the correct function version.

WaitForPrinter

Waits until the status monitor reports that the printer is no longer busy or until a time-out period elapses.

```
int WaitForPrinter(HANDLE hSession);
int WaitForPrinterReady(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.
MAGICARD_TIMEOUT	A 30-second period has elapsed without receiving any status information from the status monitor.
MAGICARD_PRINTER_ERROR	The printer has aborted the operation, due to an error.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

WaitForPrinter will exit upon the printer being no longer busy (i.e. ready), or on an error.

WaitForPrinterReady will not exit on an error.

These functions may timeout during lengthy operations. It is up to the application to determine how long it is going to wait, by repeating calls to the functions, before deciding that the printer is not responding.

The timeout length are fixed at 30 seconds (approx.).

If a printer error is reported, the application may call the *GetLastPrinterMessage* function to retrieve the error message sent by the printer.

Even if the application aborts the print job, it should still call these functions after every printer operation that can cause status information to be returned: *EndDoc*, *FeedCard* and *EjectCard.* This ensures that the status monitor is not asked to resume its normal behaviour before physical operations of the printer that the application has initiated have completed.

GetLastPrinterMessage

Retrieves a string containing the last status message sent by the (now obsolete) Rio/Tango printer.

```
int GetLastPrinterMessage(HANDLE   hSession,
                          LPSTR    lpszBuffer,
                          LPDWORD  pdwBufferSize);
```

Parameters

hSession
The session handle returned by *EnableStatusReporting*.

lpszBuffer
A pointer to the buffer that is going to receive the status message.

pdwBufferSize
A pointer to a variable that contains the buffer size.
If the buffer is too small, the function fails and places the required buffer size in this location.

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.
MAGICARD_DRIVER_NOTCOMPLIANT	The request was made to an Enduro Printer.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This command is only valid for Rio/Tango printers. For Enduro, Rio Pro and Pronto printers, use *GetLastEnduroMessage*.

The printer-specific error code is embedded in the string returned, normally at its end, in the form “(cxxxx)”, where the x’s represent digits.

GetLastEnduroMessage

Retrieves a string containing the last status message sent by Enduro, Rio Pro and Pronto printers (and their OEM derivatives).

```
int GetLastEnduroMessage(HANDLE   hSession,
                         LPTSTR   lpszBuffer,
                         LPDWORD  pdwBufferSize);
```

Parameters

hSession
: The session handle returned by *EnableStatusReporting*.

lpszBuffer
: A pointer to the buffer that is going to receive the status message.

pdwBufferSize
: A pointer to a variable that contains the buffer size.
If the buffer is too small, the function fails and places the required buffer size in this location.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.
`MAGICARD_DRIVER_NOTCOMPLIANT`	The request was made to a Rio/Tango Printer.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

For Rio and Tango printers, use *GetLastPrinterMessage*.

The printer-specific error code is embedded in the string returned, normally at its end, in the form “MMMM:mmmm”, where MMMM = Major Error code, mmmm = Minor Error Code

GeneralCommand

Sends the given command string to the printer.

```
int GeneralCommand(HANDLE  hSession,
                   LPSTR   lpszCommandString);
```

Parameters

hSession
The session handle returned by *EnableStatusReporting*.

lpszCommandString
The command string to be sent to the printer.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This function allows the user to send a command string to the printer. It is unidirectional to the printer only i.e. It does not support retrieving data from the printer.

GetPrinterStatus

Obtains the current status of the printer.

```
int GetPrinterStatus(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

`STATUS_READY`	Printer is Ready
`STATUS_BUSY`	Printer is Busy
`STATUS_ERROR`	Printer is in Error
`STATUS_OFFLINE`	Printer is Offline
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

GetPrinterInfo

Returns the printer configuration information from Enduro, Rio Pro and Pronto printers (and their OEM derivatives)

```
    int GetEnduroInfo(HANDLE        hSession,
                      PRINTER_INFO  *pPrinterInfo);
```

OR

```
    int GetPrinterInfo(HANDLE       hSession,
                       PRINTER_INFO *pPrinterInfo);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

pPrinterInfo

Pointer to a PrinterInfo structure which is to be filled with the configuration information

Structures

```
#define SERIAL_SIZE 20
typedef struct
{
   BOOL  bPrinterConnected;
   DWORD eModel;
   char  sModel[30];
   DWORD ePrintheadType;
   char  sPrinterSerial[SERIAL_SIZE];
   char  sPrintheadSerial[SERIAL_SIZE];
   char  sPCBSerial[SERIAL_SIZE];
   TCHAR sFirmwareVersion[SERIAL_SIZE];

   char  sDummy[SERIAL_SIZE - sizeof(DWORD)];
   DWORD iES_Density;

   DWORD iHandFeed;
   DWORD iCardsPrinted;
   DWORD iCardsOnPrinthead;
   DWORD iDyePanelsPrinted;
   DWORD iCleansSinceShipped;
   DWORD iDyePanelsSinceClean;
   DWORD iCardsSinceClean;
   DWORD iCardsBetweenCleans;

   DWORD iPrintHeadPosn;
   DWORD iImageStartPosn;
   DWORD iImageEndPosn;
   DWORD iMajorError;
   DWORD iMinorError;
   char  sTagUID[20];
   DWORD iShotsOnFilm;
   DWORD iShotsUsed;
   char  sDyeFilmType[20];
   DWORD iColourLength;
   DWORD iResinLength;
   DWORD iOvercoatLength;
   DWORD eDyeFlags;
   DWORD iCommandCode;
   DWORD iDOB;
```

```
      DWORD eDyeFilmManuf;
      DWORD eDyeFilmProg;
   } PRINTER_INFO;
```

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.
`MAGICARD_DRIVER_NOTCOMPLIANT`	The request was made to a Rio/Tango Printer.

The structure is loaded with the complete response by a printer to a ‘request for information’ command.

SetEjectMode

Sets the eject mode of the printer according to the passed parameter.

```
int SetEjectMode (HANDLE    hSession,
                  int       iMode);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

iMode

The eject mode being selected

SEM_EJECT_ON	Normal printer operation - cards are ejected when action is complete
SEM_EJECT_OFF	Cards are **not** ejected when action is complete

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

The eject mode returns to normal (eject mode on) if the printer is powered off

EncodeMagStripe

Encodes data to the magnetic stripe on the card

```
int EncodeMagStripe (HANDLE hSession,
                     int     iTrackNo,
                     int     iCharCount,
                     char    *lpszData,
                     int     iEncodingSpec,
                     int     iVerify,
                     int     iCoercivity,
                     int     iBitsPerChar,
                     int     iBitsPerInch,
                     int     iParity,
                     int     iLRC);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

iTrackNo

The number of the track being written (1, 2 or 3)

iCharCount

The number of characters to be written to the track (including start and end sentinels)

**lpszData*

Pointer to a buffer containing the data to be written

iEncodingSpec

The encoding method to be used

EMS_ENCODING_SPEC_ISO	ISO Encoding
EMS_ENCODING_SPEC_JIS2	JIS2 Encoding (not Rio/Tango)

iVerify

Specifies whether verification is required

EMS_VERIFY_OFF	Verification is off
EMS_VERIFY_ON	Verification is on

iCoercivity

Specifies the coercivity of the encoding

EMS_COERCIVITY_DEFAULT	Default Coercivity
EMS_COERCIVITY_HICO	High Coercivity
EMS_COERCIVITY_LOCO	Low Coercivity

iBitsPerChar

Specifies the number of bits per character for the encoding (ISO only)

EMS_BITSPERCHAR_DEFAULT	Default bits per character

`EMS_BITSPERCHAR_1`	1 bit per character
`EMS_BITSPERCHAR_5`	5 bits per character
`EMS_BITSPERCHAR_7`	7 bits per character

iBitsPerInch

Specifies the number of bits per inch for the encoding (ISO only)

`EMS_BITSPERINCH_DEFAULT`	Default bits per inch
`EMS_BITSPERINCH_75`	75 bits per inch
`EMS_BITSPERINCH_210`	210 bits per inch

iParity

Specifies the parity for the encoding (ISO only)

`EMS_PARITY_DEFAULT`	Default parity
`EMS_PARITY_OFF`	Parity off
`EMS_PARITY_ODD`	Odd parity
`EMS_PARITY_EVEN`	Even parity

iLRC

Specifies the LRC for the encoding (ISO only)

`EMS_LRC_DEFAULT`	Default LRC
`EMS_LRC_OFF`	LRC off
`EMS_LRC_ODD`	Odd LRC
`EMS_LRC_EVEN`	Even LRC

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

JIS2 encoding is not permitted with Rio/Tango printers

ReadMagStripe

Reads data from the magnetic stripe on the card

```
int ReadMagStripe (HANDLE    hSession,
                   MSVDATA   *pMSV,
                   int       iEncodingSpec);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

pMSV

Pointer to a MSV Data structure which will be filled with the magnetic stripe data.

iEncodingSpec

The encoding method in use

`EMS_ENCODING_SPEC_ISO`	ISO Encoding
`EMS_ENCODING_SPEC_JIS2`	JIS2 Encoding (not Rio Tango)

Structures

```
typedef struct
{
    DWORD   msv_id;
    DWORD   msg_len;
    DWORD   tk1_pass;
    DWORD   tk2_pass;
    DWORD   tk3_pass;
    DWORD   tk1_len;
    DWORD   tk2_len;
    DWORD   tk3_len;
    RAW_DATA raw;
} MSVDATA;

typedef struct
{
    char tk1[172];  // ISO max is 79  (7bpc, 210bpi)
    char tk2[172];  // ISO max is 40  (5bpc, 75bpi)
    char tk3[172];  // ISO max is 107 (5bpc, 210bpi)
} RAW_DATA;
```

<u>Members:</u>

msv_id:	Unique ID to distinguish this message
msg_len:	Size of message, including this
tk1_pass:	TRUE if Track 1 passed; FALSE if failed or not tested
tk2_pass:	Same for Track 2
tk3_pass:	Same for Track 3
tk1_len:	Number of bytes returned for Track 1 from start sentinel to LRC inclusive
tk2_len:	Same for Track 2
tk3_len:	Same for Track 3
raw:	Raw data for each track

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.
`MAGICARD_TIMEOUT`	A 30-second period has elapsed without receiving any magnetic stripe data.
`MAGICARD_PRINTER_ERROR`	The printer has aborted the operation, due to an error.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This function performs a complete read of the data encoded on the magnetic stripe, unlike the deprecated function *ReadMagData,* which must be used in conjunction with *RequestMagData*.

DecodeMagData

Instructs the printer to decode the magnetic data previously obtained from the printer

```
void DecodeMagData(HANDLE    hSession,
                   MSVDATA   *pMSV);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

pMSV

Pointer to a MSV Data structure which contains the magnetic stripe data.

Structures

See *ReadMagStripe*

Remarks

This function performs a conversion of the data read from the magnetic stripe on the card, removing parity bits and unpacking Binary Coded Data. The unpacked data is returned in the same MSV structure passed as input.

GetPrinterType

Returns an identifier of the printer type.

```
int GetPrinterType(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

`PRINTER_RIOTANGO`	Rio/Tango Family (Rio, Tango, X-Series)
`PRINTER_AOTA`	AOTA Family (Alto, Opera, Temp, Avalon)
`PRINTER_ENDURO`	Enduro Family (Enduro, Pronto, Rio Pro and derivatives)

GetConnectionType

Returns an identifier of the connection to the printer

```
int GetConnectionType(HANDLE hSession);
```

Parameters

hSession
The session handle returned by *EnableStatusReporting*.

Return Values

PORT_UNKNOWN	Connection Unknown
PORT_USB	USB Connection
PORT_ETHERNET	Ethernet Connection
PORT_FILE	Printer output is directed to FILE:

FlipCard

Instructs the printer to rotate the card using the flipper mechanism.

```
int FlipCard(HANDLE hSession);
```

Parameters

hSession
The session handle returned by *EnableStatusReporting*.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This command is not valid on a single sided printer (e.g. Pronto, Avalon, Alto)

CleanPrinter

Instructs the printer to execute a cleaning cycle.

```
int CleanPrinter(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

RestartPrinter

Instructs the printer to perform a reset cycle.

```
int RestartPrinter(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

PrintTestCard

Instructs the printer to print an internal test pattern.

```
int PrintTestCard(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

SetSmartMode

Sets the smart card encoding position of the printer according to the passed parameter.

```
int SetSmartMode (HANDLE  hSession,
                  int     iMode);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

iMode

The smart encoding position being selected

`SMART_MODE_DEFAULT`	The default (in the encoder) position is to be used
`SMART_MODE_PLATEN`	The encoding position is on the printer platen

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This setting is persistent and is retained by the printer through a power on/reset

SetSmartLocation

Defines the smart card encoding position according to the passed parameter.

```
int SetSmartLocation(HANDLE  hSession,
                     int     iParam);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

iParam

The smart encoding position

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This setting is persistent and is retained by the printer through a power on/reset

EraseCard

Instructs the printer to perform an erase cycle on a rewritable card.

```
int EraseCard(HANDLE  hSession,
              int     iBottomLeftX,
              int     iBottomLeftY,
              int     iTopRightX,
              int     iTopRightY)
```

Parameters

hSession

The session handle returned by EnableStatusReporting.

iBottomLeftX

iBottomLeftY

iTopRightX

iTopRightY

Coordinates of the bottom left and top right corners of the area on the card to be erased

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

If all four coordinates are set to 0, the whole card is erased.

SetEraseSpeed

Sets the speed to be used in the erase cycle on a rewritable card

```
int SetEraseSpeed(HANDLE    hSession,
                  int       iMode)
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

iMode

The erase speed being selected.

ERASE_SPEED_THOROUGH	Slower speed, for more thorough erasing
ERASE_SPEED_QUICK	Higher speed erasing

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

GetAPIVersion

Returns the version ID of the MagAPI DLL.

```
int GetAPIVersion(HANDLE       hSession,
                  API_VERSION *pAPIVersion);
```

Parameters

hSession
: The session handle returned by *EnableStatusReporting*.

pAPIVersion
: Pointer to a an API Version structure which is to be filled with the version information

Structures

```
typedef struct tag_VERSION
{
    DWORD Major;
    DWORD Minor;
    DWORD Build;
    DWORD Private;
} API_VERSION;
```

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The structure is loaded with the four character build version of the API DLL.

ErrorResponse

Sends a response to an error condition to the printer.

```
int ErrorResponse(HANDLE  hSession,
                  int     iParam)
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

iParam

The error response to be sent

ERROR_CAN	Sends a 'Cancel' response
ERROR_OKY	Sends an 'Okay' response

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Deprecated Functions

These functions have been superseded but are maintained here for backwards compatibility.

RequestMagData

Instructs the printer to feed a card and read the magnetic stripe data

```
int RequestMagData(HANDLE hSession);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

Return Values

`ERROR_SUCCESS`	The operation completed successfully.
`MAGICARD_ERROR`	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This function is used in conjunction with the *ReadMagData* function.

RequestMagData instructs the printer to feed the card and obtain the magnetic stripe data from it; then *ReadMagData* instructs the printer to transmit the data to the PC.

All 3 tracks of data are read from the card.

If a printer error is reported, the application may call the *GetLastPrinterMessage* function to retrieve the error message sent by the printer.

This function, in conjunction *ReadMagData*, has been replaced by *EncodeMagStripe* and *ReadMagStripe*.

ReadMagData

Instructs the printer to supply magnetic data (previously read from the card by a call to *RequestMagData*) to the PC.

```
int ReadMagData(HANDLE   hSession,
                MSVDATA  *pMSV);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

pMSV

Pointer to a MSV Data structure which will be filled with the magnetic stripe data.

Structures

See API function *ReadMagStripe*

Return Values

ERROR_SUCCESS	The operation completed successfully.
MAGICARD_ERROR	Win API error or a parameter is invalid.

The application may also check the result of the Win API function *GetLastError* to obtain further information about any error that has occurred.

Remarks

This function is used in conjunction with the *RequestMagData* function.

RequestMagData instructs the printer to feed the card and obtain the magnetic stripe data from it; then *ReadMagData* instructs the printer to transmit the data to the PC.

It is important that your compiler calculates the correct size of this structure, which is 548 bytes. If not using ‘C’, bear in mind that a char is a byte of 8 bits, and an int is a signed integer of 4 bytes. This structure is “packed”, i.e. there are no pad bytes.

Each 8-bit byte of raw data contains one sample of either 5 or 7-bit data.

The printer is big-endian, so the integer components will require byte-reversal on a little-endian host (e.g. a PC), i.e. the bytes in each integer, e.g. “ABCD”, will arrive as “DCBA”.

This function, in conjunction *RequestMagData*, has been replaced by *EncodeMagStripe* and *ReadMagStripe*.

GetEnduroInfo

Returns the printer configuration information from Enduro, Rio Pro and Pronto printers (and their OEM derivatives)

```
int GetEnduroInfo(HANDLE        hSession,
                  PRINTER_INFO  *pPrinterInfo);
```

Parameters

hSession

The session handle returned by *EnableStatusReporting*.

pPrinterInfo

Pointer to a PrinterInfo structure which is to be filled with the configuration information

Remarks

This function has been replaced by the *GetPrinterInfo* function.

See *GetPrinterInfo* for more information.

Driver Validation

The application may verify if the currently installed printer driver supports this API by interrogating the driver for the presence of the ESC_IS_API_CAPABLE driver escape.

The following code shows a method of doing this, assuming that hDC is a handle for a device context belonging to the driver being interrogated:

```
int iEsc = ESC_IS_API_CAPABLE;
int escRes;

escRes = ExtEscape(hDC,
                   QUERYESCSUPPORT,
                   sizeof(iEsc),
                   (LPCSTR)&iEsc,
                   0,
                   NULL);

if (escRes > 0)
{
     // The driver supports API calls.
}
```