



www.ultramagicard.com

Magicard API documentation

Application Programming Interface for Magicard Printer Drivers

Table of Contents

Overview	1
API Functions	2
EnableStatusReporting	2
DisableStatusReporting	4
FeedCard	5
EjectCard	6
WaitForPrinter	7
GetLastPrinterMessage	8
GetLastEnduroMessage	9
GeneralCommand	10
GetPrinterStatus	11
GetEnduroInfo	12
SetEjectMode	14
EncodeMagStripe	15
ReadMagStripe	18
Deprecated Functions	20
RequestMagData	20
ReadMagData	21
Driver Validation	22
Typical Application Flowchart	23
Sample Code	24
Error Code Numerical Values	29

Figures

Figure 1 – Generic information flowchart 1

Figure 2 – Typical application flowchart23

Overview

The purpose of this application programming interface (API) is to allow applications to synchronize their operations with the operation of the printer.

For example, an application may need to know when the printer has finished printing on a card.

Until now, it was not possible to retrieve such information in an accurate manner due to the way the Windows print spooler works.

With this API, an application is finally able to wait until the printer has finished its work.

Special functions are also available to 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 the API calls themselves may need to pass commands to the printer via the spooler. In this situation, the API 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:

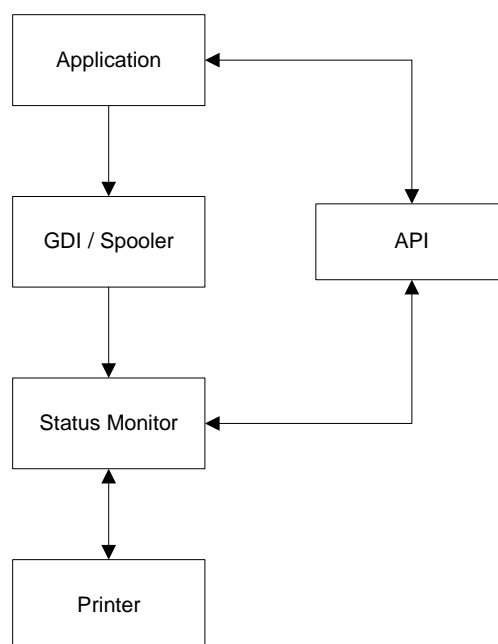


Figure 1 – Generic information flowchart

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

NB . With the API, USB communications only is supported

API Functions

EnableStatusReporting

Initialises the API, and also 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	0	The status monitor will not change its current behaviour regarding printer errors
CONFIG_QUIET	1	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.

Remarks

None.

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 assume one of the following values:

FEED_CHIPCARD	1	Places the card in the contact chip encoding station.
FEED_CONTACTLESS	2	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. The API itself is ANSI.

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. The API itself is ANSI.

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);
```

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

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

The timeout length cannot be changed - it is fixed at 30 seconds.

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 this function 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 the operations that the application started are 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.

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.

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

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 an Enduro 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	0	Normal printer operation - cards are ejected when action is complete
SEM_EJECT_OFF	1	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	0	ISO Encoding
EMS_ENCODING_SPEC_JIS2	1	JIS2 Encoding (not Rio/Tango)

iVerify

Specifies whether verification is required

EMS_VERIFY_OFF	0	Verification is off
EMS_VERIFY_ON	1	Verification is on

iCoercivity

Specifies the coercivity of the encoding

EMS_COERCIVITY_DEFAULT	0	Default Coercivity
EMS_COERCIVITY_HICO	1	High Coercivity
EMS_COERCIVITY_LOCO	2	Low Coercivity

iBitsPerChar

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

EMS_BITSPERCHAR_DEFAULT	0	Default bits per character
EMS_BITSPERCHAR_1	1	1 bit per character
EMS_BITSPERCHAR_5	2	5 bits per character
EMS_BITSPERCHAR_7	3	7 bits per character

iBitsPerInch

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

EMS_BITSPERINCH_DEFAULT	0	Default bits per inch
EMS_BITSPERINCH_75	1	75 bits per inch
EMS_BITSPERINCH_210	2	210 bits per inch

iParity

Specifies the parity for the encoding (ISO only)

EMS_PARITY_DEFAULT	0	Default parity
EMS_PARITY_OFF	1	Parity off
EMS_PARITY_ODD	2	Odd parity
EMS_PARITY_EVEN	3	Even parity

iLRC

Specifies the LRC for the encoding (ISO only)

EMS_LRC_DEFAULT	0	Default LRC
EMS_LRC_OFF	1	LRC off
EMS_LRC_ODD	2	Odd LRC
EMS_LRC_EVEN	3	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	0	ISO Encoding
EMS_ENCODING_SPEC_JIS2	1	JIS2 Encoding (not Rio Tango)

Structures

```
Error! Reference source not found.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;
```

Members:

- 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 `ReadMagData` which must be used in conjunction with `RequestMagData`.

Deprecated Functions

These functions have been replaced by *EncodeMagStripe* and *ReadMagStripe* but are maintained here for backwards compatibility.

RequestMagData

Instructs the printer to feed a card and obtain the magnetic stripe data from it.

```
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.

ReadMagData

Instructs the printer to send magnetic data (previously obtained from the card) 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".

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.
}
```

Typical Application Flowchart

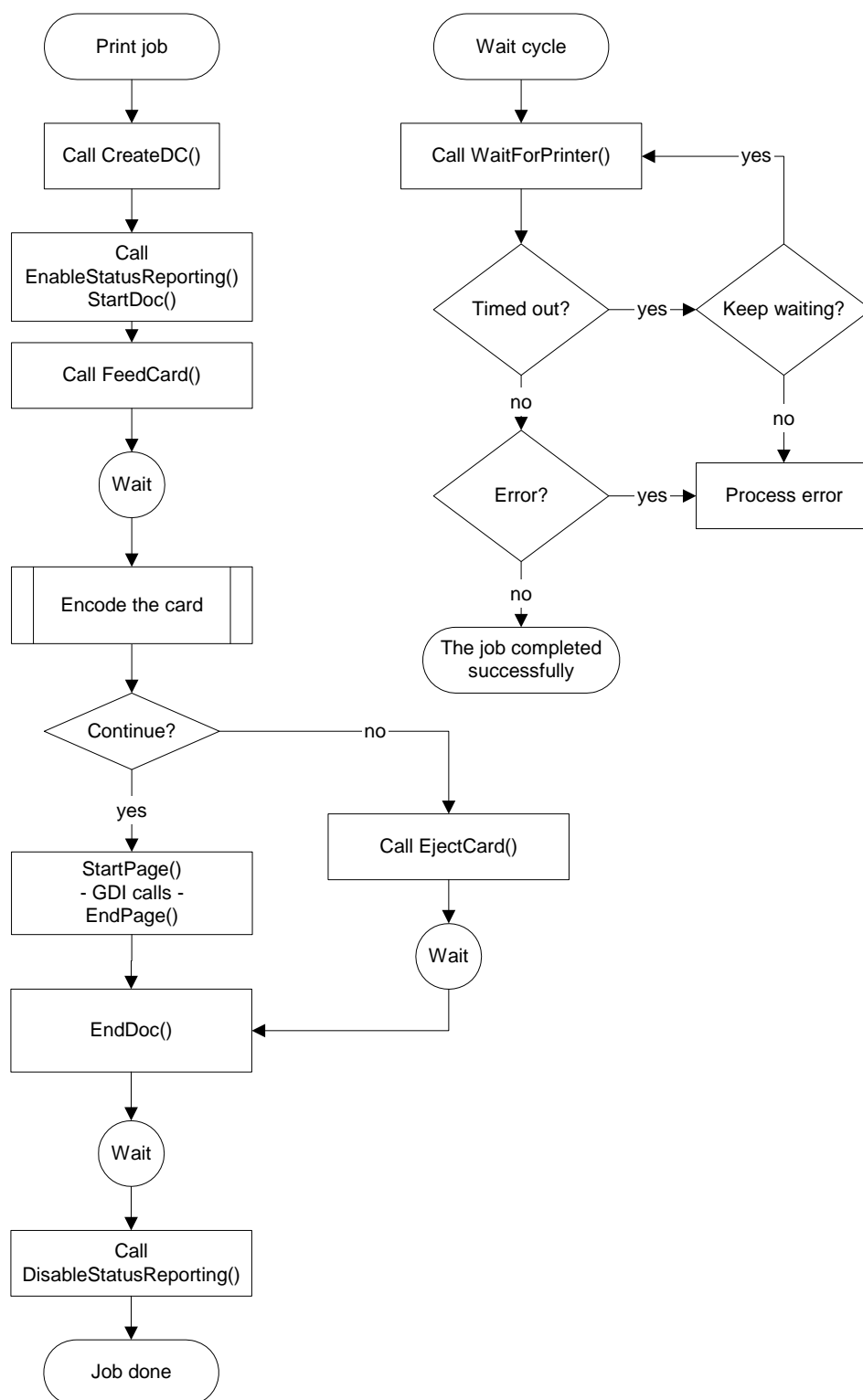


Figure 2 – Typical application flowchart

Sample Code

```
#include "stdafx.h"
#include "MagAPI.h"
#include "API_Test.h"
#include "commdlg.h"
#include <windows.h>
#include <winpool.h>
#include <stdio.h>

#####

PFNENABLESTATUSREPORTING pFnEnableStatusReporting;
PFNDISABLESTATUSREPORTING pFnDisableStatusReporting;
PFNWAITFORPRINTER pFnWaitForPrinter;
PFNGETLASTPRINTERMESSAGE pFnGetLastPrinterMessage;
PFNFEEDCARDA pFnFeedCardA;
PFNEJECTCARDA pFnEjectCardA;

#####

#define MAX_STRING 100

// Global Variables:
HINSTANCE      hInst;                // current instance
TCHAR          szTitle[MAX_STRING];  // The title bar text
TCHAR          szWindowClass[MAX_STRING]; // the main window class name

#####

int APIENTRY _tWinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPTSTR lpCmdLine,
    int nCmdShow)
{
    HANDLE hSession;
    int apiResult;
    PRINTDLG pd = {0};
    char szMessage[MAX_PATH];
    DWORD dwMsgSize;
    char DLLPath[MAX_PATH];
    DWORD LenDLLPath;

    //Access the Printer Driver Directory
    if (GetPrinterDriverDirectory(NULL,
                                  TEXT("Windows NT x86"),
                                  1,
                                  (LPBYTE)DLLPath,
                                  MAX_PATH,
                                  &LenDLLPath) == NULL)
    {
        MessageBox(HWND_DESKTOP,
                   "Cannot Access Printer Driver Directory",
                   "API Test - Error",
                   MB_OK | MB_ICONSTOP);
        return 0;
    }

    _tcscat(DLLPath, "\\Magicard_RioTango24982\\MagAPI.DLL");

    //Load the Library
    HINSTANCE hLib = LoadLibrary(DLLPath);
    if (hLib == NULL)
```

```

{
    MessageBox(HWND_DESKTOP,
        "Cannot Load MagAPI.DLL",
        "API Test - Error",
        MB_OK | MB_ICONSTOP);
    return 0;
}

//Get the Proc Addresses
pFnEnableStatusReporting = (PFNENABLESTATUSREPORTING) GetProcAddress(hLib, "EnableStatusReporting");
pFnDisableStatusReporting = (PFNDISABLESTATUSREPORTING) GetProcAddress(hLib, "DisableStatusReporting");
pFnFeedCard = (PFNFEEDCARDA) GetProcAddress(hLib, FEEDCARDPROC);
pFnEjectCard = (PFNEJECTCARDA) GetProcAddress(hLib, EJECTCARDPROC);
pFnWaitForPrinter = (PFNWAITFORPRINTER) GetProcAddress(hLib, "WaitForPrinter");
pFnGetLastPrinterMessage = (PFNGETLASTPRINTERMESSAGE) GetProcAddress(hLib, "GetLastPrinterMessage");

//
// Get a printer DC.
//
ZeroMemory(&pd, sizeof(pd));
pd.lStructSize = sizeof(PRINTDLG);
pd.hInstance = hInstance;
pd.hwndOwner = HWND_DESKTOP;
pd.Flags = PD_RETURNDC;
pd.nCopies = 1;
pd.nFromPage = 0xFFFF;
pd.nToPage = 0xFFFF;
d.nMinPage = 1;
d.nMaxPage = 0xFFFF;

if (PrintDlg(&pd))
{
    int iEsc = ESC_IS_API_CAPABLE;
    int escRes;

    escRes = ExtEscape(pd.hDC, QUERYYESCSUPPORT, sizeof(iEsc), (LPCSTR)&iEsc, 0, NULL);

    if (escRes <= 0)
    {
        MessageBox (HWND_DESKTOP,
            TEXT("Invalid Driver Version"),
            TEXT("API Test - Error"),
            MB_OK | MB_ICONSTOP);

        goto exit;
    }

    //
    // Initialize the API.
    //
    // Ideally, the spooler is empty at this point.
    //
    apiResult = (*pFnEnableStatusReporting)(pd.hDC, &hSession, CONFIG_QUIET);

    if (apiResult != ERROR_SUCCESS)
    {
        dwMsgSize = MAX_PATH;
        wsprintf(szMessage,
            "EnableStatusReporting() failed with code %i, error %u",
            apiResult,
            GetLastError());
        MessageBox(HWND_DESKTOP,
            szMessage,
            TEXT("API _test - Error"),
            MB_OK | MB_ICONSTOP);

        goto exit;
    }
}

```

```
//
// Start the printing job as usual.
//
DOCINFO di;
di.cbSize = sizeof(DOCINFO);
di.lpszDocName = "API Test - Printing";
di.lpszOutput = NULL;

StartDoc(pd.hDC, &di);

//
// Feed a card into the printer, from the hopper.
//
apiResult = (*pFnFeedCard)(hSession, FEED_CHIPCARD, 0, TEXT("Feeding Card"));

//
// Process the result of the last action.
//
do
{
    apiResult = (*pFnWaitForPrinter)(hSession);
} while (apiResult == MAGICARD_TIMEOUT);

if (apiResult == MAGICARD_PRINTER_ERROR)
{
    dwMsgSize = MAX_PATH;
    (*pFnGetLastPrinterMessage)(hSession, szMessage, &dwMsgSize);
    MessageBox(HWND_DESKTOP,
                szMessage,
                TEXT("API Test - Printer error"),
                MB_OK | MB_ICONSTOP);
}
else if (apiResult == ERROR_SUCCESS)
{
    if (MessageBox(HWND_DESKTOP,
                    TEXT("Card in place. Proceed with printing?"),
                    TEXT("API Test"),
                    MB_YESNO | MB_ICONQUESTION) == IDYES)
    {
        //
        // Front of Card
        //
        StartPage(pd.hDC);

        char szFrontMessage[] = "This is a test card";
        TextOut(pd.hDC, 40, 40, szFrontMessage, lstrlen(szFrontMessage));

        HPEN newPen1 = CreatePen(PS_SOLID, 5, RGB(255, 0, 0));
        HPEN oldPen = (HPEN)SelectObject(pd.hDC, newPen1);
        Rectangle(pd.hDC, 60, 160, 320, 380);
        DeleteObject(newPen1);

        EndPage(pd.hDC);

        if (MessageBox(HWND_DESKTOP,
                        TEXT("Do you want to print a double-sided card?"),
                        TEXT("API Test"),
                        MB_YESNO | MB_ICONQUESTION) == IDYES)
        {
            //
            // Back of Card
            //
            StartPage(pd.hDC);

            char szBackMessage[] = "Reverse Side";
            TextOut(pd.hDC, 40, 40, szBackMessage, lstrlen(szBackMessage));
        }
    }
}
```

```
        HPEN newPen2 = CreatePen(PS_SOLID, 5, RGB(0, 0, 255));
        SelectObject(pd.hDC, newPen2);
        Rectangle(pd.hDC, 60, 160, 320, 380);
        DeleteObject(newPen2);

        EndPage(pd.hDC);
    }

    SelectObject(pd.hDC, oldPen);

    EndDoc(pd.hDC);

    //
    // Wait until the print job ends.
    //
    do
    {
        apiResult = (*pFnWaitForPrinter)(hSession);
    } while (MAGICARD_TIMEOUT == apiResult);

    if (apiResult == MAGICARD_PRINTER_ERROR)
    {
        dwMsgSize = MAX_PATH;
        (*pFnGetLastPrinterMessage)(hSession, szMessage, &dwMsgSize);
        MessageBox(HWND_DESKTOP,
            TEXT("Printer Error"),
            TEXT("API Test - Printer error"),
            MB_OK | MB_ICONSTOP);
    }
}

else
{
    //
    // Eject the card from inside the printer.
    //
    apiResult = (*pFnEjectCard)(hSession, TEXT("Ejecting Card"));

    do
    {
        apiResult = (*pFnWaitForPrinter)(hSession);
    } while (MAGICARD_TIMEOUT == apiResult);

    if (apiResult == MAGICARD_PRINTER_ERROR)
    {
        dwMsgSize = MAX_PATH;
        (*pFnGetLastPrinterMessage)(hSession, szMessage, &dwMsgSize);
        MessageBox(HWND_DESKTOP,
            szMessage,
            TEXT("API Test - Printer error"),
            MB_OK | MB_ICONSTOP);
    }

    EndDoc(pd.hDC);
}

//
// Tell the monitor that it can interact normally with the user.
//
apiResult = (*pFnDisableStatusReporting)(hSession);

if (apiResult != ERROR_SUCCESS)
{
    dwMsgSize = MAX_PATH;
    wsprintf(szMessage,
        "DisableStatusReporting() failed with code %i, error %u",
```

```
        apiResult, GetLastError());
    MessageBox(HWND_DESKTOP,
        szMessage,
        TEXT("API _test - Error"),
        MB_OK | MB_ICONSTOP);
}

//
// Clean up.
//
if (NULL != pd.hDevMode)
{
    GlobalFree(pd.hDevMode);
}
if (NULL != pd.hDevNames)
{
    GlobalFree(pd.hDevNames);
}

exit:
    DeleteDC(pd.hDC);

    FreeLibrary(hLib);

    MessageBox(HWND_DESKTOP,
        TEXT("All done."),
        TEXT("API Test"),
        MB_OK | MB_ICONINFORMATION);
}
}
```

Error Code Numerical Values

MAGICARD_TIMEOUT	-1
MAGICARD_ERROR	-2
MAGICARD_PRINTER_ERROR	-3
MAGICARD_DRIVER_NOTCOMPLIANT	-4
MAGICARD_OPENPRINTER_ERROR	-5
MAGICARD_REMOTECOMM_ERROR	-6
MAGICARD_LOCALCOMM_ERROR	-7
MAGICARD_SPOOLER_NOT_EMPTY	-8
MAGICARD_REMOTECOMM_IN_USE	-9
MAGICARD_LOCALCOMM_IN_USE	-10