Programmer's Manual

Linux 32-bit API
"libmp2032.so"

For POS and Kiosk printers

Version 2.0

Index

Introducing the API 3
Installing the API – Parallel and Serial Ports 4
Installing the API – USB 5
Declaring the API in the Application 6
API Functions 9
- SetSerialBaudRate() 9
- InitializePort() 10
- SelectCodePage() 11
- FormatTX() 12
- BematechTX() 13
- CommandTX() 14
- CheckDocInserted() 15
- AuthenticateDoc() 16
- Read_Status() 17
- GraphicChar() 19
- ClosePort() 20
- ConfigPrinterModel() 21
- ConfigSlipSize() 22
- EnableLongSlip() 23
- EnableWaitPrinting() 24
- WaitPrinting() 25
- ReadDrawerStatus() 26
- CutPaper() 27
- Print_UPCA_Barcode() 28
- Print_UPCE_Barcode() 29
- Print_EAN13_Barcode() 30
- Print_EAN8_Barcode() 31
- Print_CODE39_Barcode() 32
- Print_ITF_Barcode() 33
- Print_CODABAR_Barcode() 34
- Print_CODE93_Barcode() 35
- Print_CODE128_Barcode() 36
- Print_ISBN_Barcode() 37
- Print_MSI_Barcode() 38
- Print_PLESSEY_Barcode() 39
- Print_PDF417_Barcode() 40
- ConfigureBarcode() 41
- EnableRetractableFcn() 42
- ConfigureRetractableFcn() 43
- CheckPaperInPresenter() 44

API Function Summary / Glossary 47

Introducing the API

An API (Application Program Interface) is a software interface that enables an application program to communicate with the operating system (such as Linux) and other services (such as printing and communications) provided by the OS.

The Bematech Linux API allows a Linux application program to send data to or receive status from Bematech POS Printers (such as the MP-20 CI) and Bematech Kiosk Printers (such as the PB-20 AC, PB-20 TH, KC-4112).

The Bematech Linux API is provided in the form of a Linux .so (Shared Object, similar to a DLL – Dynamic Link Library under the Windows® OS) – a file that contains programmed routines that can be called from the application. The file is called *libmp2032.so* and is described below. The libmp2032 shared object contains functions to abstract most of the printers' functionalities, such as printing bar codes, bitmaps, formatted text, status reading, printer setup etc.

The Linux Shared Object (libmp2032.so)

The libmp2032.so shared object may be used with many programming languages under most Linux systems. For easier installation, Bematech provides pre-compiled binaries of this shared object in the RPM (Red Hat Package Manager) format. These packages are certified to be installed under Red Hat Linux versions 8.0 and above (including Red Hat Enterprise Linux version 3), and under SuSE Linux versions 9.0 and above. A complete list of supported versions can be found on Bematech's Web site.

After installation of the corresponding RPM packages, the libmp2032.so file will be located under the /usr/lib directory, and a C include file named mp2032.h will be available under the /usr/include/mp2032 directory.

The programming languages for Linux usually offer ways of accessing the functions of a shared object. The most common is the C programming language, and the include file is the way a C compiler is aware of the implemented functions in the shared object. Under other programming languages, a special modules is usually needed to make a "bridge" between the language compiler (or interpreter, depending on the language) and the libmp2032.so shared object. In any case, though, the programmer is required to know how to use the API by knowing its function names, return values and parameters. The programmer will usually need to know the following (in this order):

a) The name of the shared object;

b) The name of the function in the shared object;

c) The type of the function return value (although this is not always required);

d) Passing parameters.

Its operation is defined through functions, which are accessed by the application. The API functions implemented in the libmp2032.so are described throughout this manual.

Installing the API – Parallel and Serial Ports

Prior to installing and using the shared object, check your computer's SETUP for the settings of the LPT port where the printer will be connected to.

This setting should be the STANDARD (SPP) / One-way mode.

Since the Linux OS is very standardized with regards to file locations, there is no need for manual installation of the shared object. The shared object and other necessary files will be installed automatically when using the provided RPM packages. Bematech will eventually provide manually installable packages in .tar.gz format. Please check out Bematech's Web site from time to time and watch out Bematech's Developer Newsletter.

Please consult our "Driver Installation and Test Application Usage Manual" for "Thermal Printers - Linux Spooler Driver" for detailed installation instructions.

Installing the API – USB

Currently the Linux API driver does not fully support USB. Please check out Bematech's Web site from time to time and watch out Bematech's Developer Newsletter for a new release supporting USB in the near future.

The USB support in the shared object, when available, will be automatically installed together with the provided RPM package. There is a possibility of some extra libraries be required for USB support, and they will be provided by Bematech together with the other RPM packages.

Declaring the API in the Application

The Bematech Linux shared object for POS / Kiosk Printers, independently of the Linux platform, must be declared with its specific functions for the programming language being used, so that the compiler / interpreter is aware of the functions implemented in the libmp2032 shared object.

Each programming language has its own different way of declaring an external shared object. Always see the programming manual for your language to verify this procedure. Bematech will be constantly working for releasing modules for the most popular programming languages, if they require one.

Under the C programming language, a include file is used. It will be installed as /usr/include/mp2032/mp2032.h when you install Bematech's provided RPM packages. The include file is at the same time an interface for the programming language and a specification file for the programmer, since it shows the function names, parameters and return values.

Include File for the C Programming Language

```
/*****************************************************************************
 * Project: Bematech API - Thermal Kiosk Printers / Thermal Mini-Printers
 * File: mp2032.h
 * Version: $Name:  $
 * Revision: $Id: mp2032.h,v 1.6 2004/10/08 19:55:47 celso Exp $
 *
 * Device: Thermal mini-printers and kiosk thermal printers
 * Bematech models:
 * . kiosk printers: KC-2580, KF-6580, KC-3800 P, KC-1800, KB-1800, KC-4112
 * . mini-printers: MP-20CI, MP-20MI, MP-20TH, MP-2000TH, MP-2000CI, MP-2100TH
 *
 * Author: Celso Kopp Webber <celso@webbertek.com.br>
 * . Developed by Webbertek(R) for Bematech S/A
 *
 * Description of this file contents:
 * . API specification/functions prototypes - header file
 *****************************************************************************/

#ifndef MP2032_H
  #define MP2032_H

// TODO: pra que serve este MAXSIZE que estava no .h da Bematech?
#define MAXSIZE 256


/*****************************************************************************
 * Constants and Macros
 *****************************************************************************/

/* MS-Windows compatibility */
/* Basic data types provided in windows.h */
#ifndef ALLREADY_HAVE_WINDOWS_TYPE
  #ifndef UCHAR
    #define UCHAR unsigned char
  #endif
  #ifndef UINT
    #define UINT unsigned int
  #endif
#endif

/* The boolean type */
#ifndef BOOL
  #define BOOL           int
  #define TRUE           1
  #define FALSE          0
#endif


/*****************************************************************************
 * Functions prototypes (English API interface)
 *****************************************************************************/

int InitializePort(char *pcPort);
```

```
int FormatTX(char *BufMp20TX, short LetterType, short Italic, short Underlined,
             short Expanded, short Emphasized);

/* This functions has the same name in Portuguese */
/* int BematechTX(char *BufTrans); */

int CommandTX(char *BufTrans, short BufTransSize);

int CheckDocInserted();

int AuthenticateDoc(char *String, int Timeout);

unsigned int Read_Status();

int GraphicChar(char *Character, int CharacterSize);

int ClosePort();

int ConfigPrinterModel(int Model);

int ConfigSlipSize(int Lines);

int EnableLongSlip(int Enable);

int EnableWaitPrinting(int Enable);

void WaitPrinting();

int ReadDrawerStatus();

int CutPaper(int Mode);

int Print_UPCA_Barcode(char *Code);

int Print_UPCE_Barcode(char *Code);

int Print_EAN13_Barcode(char *Code);

int Print_EAN8_Barcode(char *Code);

int Print_CODE39_Barcode(char *Code);

int Print_ITF_Barcode(char *Code);

int Print_CODABAR_Barcode(char *Code);

int Print_CODE93_Barcode(char *Code);

int Print_CODE128_Barcode(char *Code);

int Print_ISBN_Barcode(char *Code);

int Print_MSI_Barcode(char *Code);

int Print_PLESSEY_Barcode(char *Code);

int Print_PDF417_Barcode(int ECC, int Height, int Width, int Columns, char *Code);

int ConfigureBarcode(int Height, int Width, int CharPosition, int Font, int Margin);

int EnableRetractableFcn(int Flag);

int ConfigureRetractableFcn(int Timeout);

int CheckPaperInPresenter();

void SetSerialBaudRate(UINT baudrate);


/**************************************************************************
 * Functions prototypes (Brazilian Portuguese API interface)
 **************************************************************************/

int IniciaPorta(char *pcPorta);

int FormataTX(char *BufMp20TX, short TpoLtra, short Italic, short Sublin,
```

```
              short Expand, short Enfat);
int BematechTX(char *BufTrans);
int ComandoTX(char *BufTrans, short TamBufTrans);
int DocumentInserted();
int AutenticaDoc(char *String, int Tempo);
unsigned int Le_Status();
int CaracterGrafico(char *Caracter, int TamCaracter);
int FechaPorta();
int ConfiguraModeloImpressora(int Modelo);
int ConfiguraTamanhoExtrato(int iNumeroLinhas);
int HabilitaExtratoLongo(int Habilita);
int HabilitaEsperaImpressao(int Habilita);
void EsperaImpressao();
int Le_Status_Gaveta();
int AcionaGuilhotina(int iModo);

int ImprimeCodigoBarrasUPCA(char *cCodigo);
int ImprimeCodigoBarrasUPCE(char *cCodigo);
int ImprimeCodigoBarrasEAN13(char *cCodigo);
int ImprimeCodigoBarrasEAN8(char *cCodigo);
int ImprimeCodigoBarrasCODE39(char *cCodigo);
int ImprimeCodigoBarrasITF(char *cCodigo);
int ImprimeCodigoBarrasCODABAR(char *cCodigo);
int ImprimeCodigoBarrasCODE93(char *cCodigo);
int ImprimeCodigoBarrasCODE128(char *cCodigo);
int ImprimeCodigoBarrasISBN(char *cCodigo);
int ImprimeCodigoBarrasMSI(char *cCodigo);
int ImprimeCodigoBarrasPLESSEY(char *cCodigo);
int ImprimeCodigoBarrasPDF417(int iCorrecaoErros, int iAltura, int iLargura, int iColunas, char * cCodigo);
int ConfiguraCodigoBarras(int Altura, int Largura, int PosicaoCaracteres, int Fonte, int Margem);

int HabilitaPresenterRetratil(int iFlag);
int ProgramaPresenterRetratil(int iTempo);
int VerificaPapelPresenter();

#endif   /* MP2032_H */
```

API Functions

Here you will find all functions available in the libmp2032 API, which will enable you to send or receive data to and from the Bematech kiosk or POS printer.

The library functions were initially programmed in Portuguese and for compatibility purposes most of the original function names are being preserved. While helping on compatibility, this brings some “cryptic” names for foreign programmers.

With this in mind, a glossary will be provided at the end of this document containing the function names and what those names actually mean in English.

The examples shown below for each function were taken directly from the test application provided by the RPM packages for the libmp2032 shared object. Please refer to the test application source code located in /usr/src/mp2032/test-mp2032.c for further details.

SetSerialBaudRate()

This function has the purpose of setting the serial port baud rate. It must be used BEFORE calling InitializePort.

Parameter:

Port: integer with the port baud rate in bits per second (bps).

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int baudrate;

  printf("\nEnter serial baud rate (1200/2400/4800/9600/19200/38400/57600/115200): ");
  scanf("%d", &baudrate);
  SetSerialBaudRate(baudrate);
}
```

Return for this function is void.

InitializePort()

This function has the purpose of opening the communication port (serial COM or parallel LPT) where the printer is connected. It should be used at the start of the Application, just after SetSerialBaudRate.

Parameter:

Port: STRING with the name of the communication port.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *buffer;

  buffer=(char *) malloc(51);
  printf("\n\nEnter device port name (ex: /dev/parport0; /dev/ttyS0; etc): ");
  scanf("%50s", buffer);
  result=InitializePort(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

Less then or equal to 0 (zero) : unable to open the communication port.
1 (one) : communication port opened ok.

SelectCodePage()

This function selects the equipment's internal character table for text printing. While most of the character tables (some times called code pages) support the full ASCII table, it is possible that a specific code page will support only special characters in the future.

Basically, each code page contains a different set of "special" characters. For example, the code page "437" contains a set of line drawing characters, useful for drawing "windows" in text mode. Other code pages usually contain international accented characters. Please see the table below for the available code pages at the time of this writing. It is possible that this set of code pages will grow in the future, so your equipment's manual should always be used as a reference together with this manual.

Another feature of this function is that any text sent to the printer using the "text functions" FormatTX() and BematechTX() will have its accented characters converted from the native character set of the operating system to the accented characters of the corresponding code page. For instance, if your Linux systems uses UTF-8 natively (default under Red Hat 8.0 and later), and if you select your printer to use Code Page 850 (default setting), all accented characters will be automatically converted from UTF-8 to Code Page 850. This way, your application can receive accented characters from user input and print this text correctly using the FormatTX() and BematechTX() functions.

Parameter:

CodePage: integer indicating the desired code page, according to the table below.

Parameter	Code Page (Character Table)
0 (dot matrix models)	Code Page 850
1 (dot matrix models)	Abicomp (an older Brazilian standard)
2	Code Page 850
3	Code Page 437
4	Code Page 860
5	Code Page 858

Please note that dot matrix models use the parameter "0" for Code Page 850, while thermal models use the parameter "2" for the same code page. The default code page also differs from dot matrix and thermal printer models: Abicomp for dot matrix and Code Page 850 for thermal.

The default setting for accented character conversion is always "Code Page 850", since this is the default setting on most printer models.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
```

FormatTX()

This function is for sending text to the printer with formatting dictated by the parameters:

LetterType:
1 = condensed
2 = normal
3 = elite

Italic:
1 = enable italic mode
0 = disable italic mode

Underlined:
1 = enable underlined mode
0 = disable underlined mode

Expanded:
1 = enable expanded mode
0 = disable expanded mode

Bold:
1 = enable bold mode
0 = disable bold mode

User mode:

FormatTX(" Text to be Printed ", LetterType, Italic, Underlined, Expanded, Bold)

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter text to be printed using FormatTX(): ");
  scanf("%254s", buffer);
  strcat(buffer, "\x0A");  /* added LF at buffer end to flush printing */
  printf("\nLetter type (0=condensed;1=normal;2=elite)");
  scanf("%d", &letter_type);
  printf("\nItalic? (1=yes; 0=no)");
  scanf("%d", &italic);
  printf("\nUnderline? (1=yes; 0=no)");
  scanf("%d", &underline);
  printf("\nExpanded? (1=yes; 0=no)");
  scanf("%d", &expanded);
  printf("\nEmphasized? (1=yes; 0=no)");
  scanf("%d", &emphasized);
  result=FormatTX(buffer,letter_type,italic,underline,expanded,emphasized);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

1 (one) : Successful. Function carried out without any problems.
0 (zero) : Communication Error.

BematechTX()

This function is used for printing text, sending a set with several lines.

Parameter:

Text: STRING with text to be printed.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *buffer;
  char *bigbuffer;

  buffer=(char *) malloc(61);
  printf("\n\nEnter file name to be printed using BematechTX(): ");
  scanf("%60s", buffer);
  file=fopen(buffer, "r");
  if (file == NULL)
    show_error(-10);
  else
  {
    buffer=realloc(buffer, BUFFER_SIZE);
    bigbuffer=NULL;
    qtty=0;
    while (! feof(file))
    {
      bigbuffer=(char *) realloc(bigbuffer, qtty+BUFFER_SIZE);
      result=fread(buffer, 1, BUFFER_SIZE, file);
      memcpy(bigbuffer+qtty, buffer, result);
      qtty+=result;
    }
    bigbuffer=realloc(bigbuffer, qtty+1);
    bigbuffer[qtty]='\0';
    result=BematechTX(bigbuffer);
    show_error(result);
    free(bigbuffer);
  }
  free(buffer);
```

Response for this function is given by an integer, where the response is:

1 (one) : Successful. Function carried out without any problems.
0 (zero) : Communication error.

CommandTX()

This function is used for sending printer specific commands to the printer, such as control commands.

Parameters:

Command : STRING with the command you want to perform.
Command Size : INTEGER with the size of the command to be sent.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char *buffer;
  char *bigbuffer;

  buffer=(char *) malloc(61);
  printf("\n\nEnter file name to be printed using CommandTX(): ");
  scanf("%60s", buffer);
  file=fopen(buffer, "r");
  if (file == NULL)
    show_error(-10);
  else
  {
    buffer=realloc(buffer, BUFFER_SIZE);
    bigbuffer=NULL;
    qtty=0;
    while (! feof(file))
    {
      bigbuffer=(char *) realloc(bigbuffer, qtty+BUFFER_SIZE);
      result=fread(buffer, 1, BUFFER_SIZE, file);
      memcpy(bigbuffer+qtty, buffer, result);
      qtty+=result;
    }
    result=CommandTX(bigbuffer, qtty);
    show_error(result);
    free(bigbuffer);
  }
  free(buffer);
```

Response for this function is given by an integer, where the response is:

1 (one) : Successful. Function carried out without any problems.
0 (zero) : Communication error.

CheckDocInserted()

This function checks presence of a document for validation (used with dot-matrix printers that can perform validation)

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;

  result=CheckDocInserted();
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

1 (one) : There is a document inserted for validation.
0 (zero) : There is no document inserted for validation.

AuthenticateDoc()

This function allows easier validation of documents, with no need for the application to send control commands to the printer. Just send the text you want written for validation and the validation waiting time in milliseconds.

Parameters:

Text : STRING that will be used in validation.
Time : INTEGER with waiting time, in milliseconds.

Method for use:

AuthenticateDoc("Validation Text", 5000)

where:

5000 = 5 seconds wait for document insertion

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter text to be printed using AuthenticateDoc(): ");
  scanf("%254s", buffer);
  printf("\nTimeout for inserting document (in ms): ");
  scanf("%d", &timeout);
  result=AuthenticateDoc(buffer, timeout);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:
1 (one) : Successful. Function carried out without any problems.
0 (zero) : Time-Out. Document was not inserted for validation.

Read_Status()

This function takes care of reading printer status such as On Line, Off Line, Turned Off, No Paper, Paper near end and head up.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("\n\nStatus is: %d\n", Read_Status());
}
```

Feedback for this function is through an integer, where each printer status will be defined by a value. It is important to perform this function in each printer status and store the values for future comparison. The function will return a value for ON-LINE, a value for OFF LINE, and so on.

- For the Parallel Port

	POS Printer	Kiosk Printer
ON-LINE	144	144
OFF-LINE	79	79
End of Paper	40	40
Low Paper and ON-LINE	None	48
Low Paper and OFF-LINE	None	32
Head-up	None	128
Off or Cable Disconnected	0	0

- For the Serial Port

	POS Printer	Kiosk Printer
ON-LINE	24	24
OFF-LINE	0	0
Off or Cable Disconnected	0	None
No Paper	32	32
Low Paper and ON-LINE	None	5
Low Paper and OFF-LINE	None	4
Head-up	None	9
Printer On Error or Off/Offline	3	3

-For the USB Port

	POS Printer	Kiosk Printer
ON-LINE	24	24
OFF-LINE	0	0
Off or Cable Disconnected	68	68
No Paper	32	32
Low Paper and ON-LINE	5	5
Low Paper and OFF-LINE	4	4
Head-up	9	9
Printer Buffer Full	64	64

Cut Error	65	65
High Head Temperature	66	66
Paper Jam	67	67

GraphicChar()

This function prints a bit image created. For more information on bit images please check the printer user's manual.

Example:

```
/*
                    GRAPHIC
              1 2 3 4 5 6 7 8 9
bit 7 = 128   *               *
bit 6 = 064   * *             *
bit 5 = 032   * * *           *
bit 4 = 016   * * * *         *
bit 3 = 008   * * * * *       *
bit 2 = 004   * * * * * * *   *
bit 1 = 002   * * * * * * * *   *
bit 0 = 001   * * * * * * * * * *
*/
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(36);
  /* command for enabling 9x9 bitmap graphics */
  memcpy(buffer, "\x1B\x5E\x12\x00", 4);
  result=CommandTX(buffer, 4);
  show_error(result);
  memcpy(buffer,
"\xFF\x00\x00\x00\x7F\x00\x00\x00\x3F\x00\x00\x00\x1F\x00\x00\x00\x0F\x00\x00\x00\x07\x00\x00\x00\x
03\x00\x00\x00\x01\x00\x00\x00\xFF\x00\x00\x00", 36);
  result=GraphicChar(buffer, 36);
  show_error(result);
  memcpy(buffer, "\x0D\x0A", 2);
  result=CommandTX(buffer, 2);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

1 (one) : Successful. Function carried out without any problems.
0 (zero) : Communication error.

ClosePort()

This function is needed to close the communication port previously opened, releasing the port for other tasks.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;

  result=ClosePort();
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

1 (one) : Successful. Function carried out without any problems.
0 (zero) : Error closing communication port.

Note: The function ClosePort() must be used only when closing the Application, i.e., when finishing a print job.

ConfigPrinterModel()

This function is used where there is a need for international accented characters. Its use is optional but this function must be called if the printer is of the MP-20 CI, MP-20 MI or MP-20 S models and international characters are needed. If this function is not called the API assumes the default.

INTEGER type parameters:

0 (zero) : models MP-20 TH, MP-2000 CI, MP-2000 TH and Kiosk Printers (default).
1 (one) : models MP-20 CI, MP-20 MI e MP-20 S.

Default is 0 (zero).

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  int model;

  printf("\n\nEnter printer model (0=thermal; 1=dot matrix): ");
  scanf("%d", &model);
  result=ConfigPrinterModel(model);
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

1 (one) : OK.
-2 (minus two) : Invalid Parameter.

ConfigSlipSize()

Note: Use it only when a Kiosk Printer is installed.

This function sets up the number of lines printed in the statement prior to ejection in “log” mode. The number of lines may vary from 1 to 150 lines. Default is 90 lines.

Parameter:

Size: INTEGER with the size of the statement.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  int slipsize;

  printf("\n\nEnter number of lines per slip: ");
  scanf("%d", &slipsize);
  result=ConfigSlipSize(slipsize);
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

0 (zero):	Communication Error.
1 (one):	OK.
-2 (minus two):	Invalid Parameter.

Important

This function will have no effect if the function EnableLongSlip (Enable Ticket Length - see below) is not performed.

EnableLongSlip()

Note: Use it only when a Kiosk Printer is installed.

This function enables or disables the number of lines set up in the function ConfigSlipSize. If this function is performed with “enable”, the number of lines will not be Default (90 lines) - it will be the number sent in the function ConfigSlipSize.

INTEGER type parameter:

0 (zero) : Disabled.
1 (one) : Enabled.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  int longslip;

  printf("\n\nEnable long slip (1=yes; 0=no): ");
  scanf("%d", &longslip);
  result=EnableLongSlip(longslip);
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

 0 (zero) : Communication Error.
 1 (one) : OK.
-2 (minus two) : Invalid Parameter.

EnableWaitPrinting()

This function enables or disables sending of character ETX (03h), which keeps the printer busy until the entire contents of the reception buffer have been printed. The ETX character must be enabled for the *EnableWaitPrinting* (wait for end of print job) function to work.

Integer type parameter:

0 (zero) : Disabled.
1 (one) : Enabled.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  int waitprinting;

  printf("\n\nEnable waiting for end of printing (1=yes; 0=no): ");
  scanf("%d", &waitprinting);
  result=EnableWaitPrinting(waitprinting);
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

1 (one) : OK.
-2 (minus two) : Invalid Parameter.

WaitPrinting()

This function halts the Application until all data sent to the printer is printed. Since this function uses the ETX command functionality, it will only work after the ETX command is enabled by the *EnableWaitPrinting* function.

Parameter:

None

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  WaitPrinting();
}
```

This function has no returning bytes.

ReadDrawerStatus()

This function returns the status of the cash drawer (opened / closed).

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;

  result=ReadDrawerStatus();
  show_error(result);
}
```

Return from this function is given by an integer, where the response is:

0 (zero) : Communication Error.
1 (one) : Cash drawer opened.
2 (two) : Cash drawer closed.

Note: In previous versions of MP-2000, cash drawer status are not available if serial communication is being used. See our Technical Support for any questions.

CutPaper()

This function activates the cutter for partial or full cutting.

Integer type parameters:

0 (zero) : Partial Cut (if available).
1 (one) : Full Cut.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  int cuttingmode;

  printf("\n\nEnter desired cutting mode (0=partial; 1=full): ");
  scanf("%d", &cuttingmode);
  result=CutPaper(cuttingmode);
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

0 (zero) : Communication Error.
1 (one) : OK.
-2 (minus two) : Invalid Parameter.

Note: Printers equipped with the presenter (such as Kiosk Printers) will not perform partial cuts.

Print_UPCA_Barcode()

This function prints the UPCA barcode.

Parameter:

Code: STRING with 11 digits size from 0 to 9.

Note: The verifier digit will be automatically added.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_UPCA_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_UPCA_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_UPCE_Barcode()

This function prints the UPCE barcode.

Parameter:

Code: STRING with 6 digits size from 0 to 9.

Note: The verifier digit will be automatically added.

Example:

```
#include <mp2032/mp2032.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_UPCE_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_UPCE_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_EAN13_Barcode()

This function prints the EAN13 barcode.

Parameter:

Code: STRING with 12 digits size from 0 to 9.

Note: The verifier digit will be automatically added.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_EAN13_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_EAN13_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

0 (zero)	: Communication Error.
1 (one)	: OK.
-1 (minus one)	: Run Error.
-2 (minus two)	: Invalid Parameter.

Print_EAN8_Barcode()

This function prints the EAN8 barcode.

Parameter:

Code: STRING with 7 digits size from 0 to 9.

Note: The verifier digit will be automatically added.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_EAN8_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_EAN8_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_CODE39_Barcode()

This function prints the CODE39 barcode.

Parameter:

Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.

The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	15
1	9
2	6

Bar Width is 1 (default).

Notes:

- The verifier digit will be automatically added.
- The barcode accepts digits from 0 to 9, letters from A to Z (Either Caps or Small Letters) and characters "space", "$", "%", "+", "-", "." and "/".
- Upper case and lower case characters cannot be used at the same time.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_CODE39_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_CODE39_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_ITF_Barcode()

This function prints the ITF barcode.

Parameter:

Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.

The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	30
1	20
2	14

Bar Width is 1 (default).

Note: Accepts digits from 0 to 9.
Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_ITF_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_ITF_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

 0 (zero)   : Communication Error.
 1 (one)   : OK.
-1 (minus one) : Run Error.
-2 (minus two) : Invalid Parameter.

Print_CODABAR_Barcode()

This function prints the CODABAR barcode.

Parameter:

Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.

The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	20
1	12
2	8

Bar Width is 1 (default).

Notes:

- The verifier digit will be automatically added.
- Accepts digits from 0 to 9, letters A, B, C and D (either CAPS or Small Letters) and characters "$", "+", "-", ".", "/" and ":".

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_CODABAR_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_CODABAR_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_CODE93_Barcode()

This function prints the CODE93 barcode.

Parameter:
Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.

The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	15
1	9
2	6

Bar Width is 1 (default).

Notes:

- The verifier digit will be automatically added.
- Accepts digits from 0 to 9, letters A to Z (either CAPS or small letters) and characters "space", "$", "%", "+", "-", "." e "/".

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_CODE93_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_CODE93_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

 0 (zero)   : Communication Error.
 1 (one)   : OK.
-1 (minus one) : Run Error.
-2 (minus two) : Invalid Parameter.

Print_CODE128_Barcode()

This function prints the CODE128 barcode.

Parameter:

Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.

The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	42
1	28
2	16

Bar Width is 1 (default).

Notes:

- Uses characters from the ASCII table, from 1d up to 127d.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_CODE128_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_CODE128_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_ISBN_Barcode()

This function prints the ISBN barcode.

Parameter:

Code: STRING with the ISBN.

The ISBN (International Standard Book Number) is a 10-digit product number, used by publishers, booksellers and libraries for ordering, listing and stock control purposes. It is provided by the ISBN Agency.

The ISBN is composed of the following structure: G-PPPPP-BBB-C AAAAA where G is a Group Number, PPPPP is a 5-digit Publisher Identifier group, BBB is a 3-digit Book Number group, C is an optional Check digit and AAAAA is an optional 5-digit add-on number.

The optional check digit can go from 0 to 9 as well as the “X” letter, used when the check digit must represent the value “10”.

The optional add-on number can be used for example in pricing information in publications – the use of such number is regulated depending on the country. In the United States the number is mandatory and must show the suggested price or the number 90000 if no price is suggested.

With this structure we can have the following examples:

ISBN with check digit and without add-on: 0-330-28987-1

ISBN without check digit and without add-on: 0-330-28987

ISBN with check digit and add-on number: 1-56592-292-X 90000

ISBN without check digit and with add-on: 1-56592-292 01599

When sending the string to the API the application must send the whole ISBN numbers, including hyphens – the printer or the API also do not calculate the check digit. The example below shows how to send the “ISBN with check digit and add-on number” shown above.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_ISBN_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_ISBN_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_MSI_Barcode()

This function prints barcode MSI.

Parameter:

Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.
The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	16
1	10
2	7

Bar Widh is 1 (default).

Notes:
- The verifier digit will be automatically added.
- Uses digits from 0 to 9.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_MSI_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_MSI_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

 0 (zero)   : Communication Error.
 1 (one)   : OK.
-1 (minus one) : Run Error.
-2 (minus two) : Invalid Parameter.

Print_PLESSEY_Barcode()

This function prints barcode PLESSEY.

Parameter:

Code: STRING with the code to be generated. The number of characters in the string will vary according to the width of the print field and the width of the barcode.

The table below shows the correlation between the bar width and the number of characters that can be printed in a 80mm kiosk printer.

Bar Width	Number of Characters
0	13
1	7
2	4

Bar Width is 1 (default).

Notes:

- The verifier digit will be automatically added.
- The barcode uses digits from 0 to 9 and letters A to Z (either CAPS or small letters). Uppercase and lowercase characters can't be combined in the same code.

Example:

```
int main()
{
  int result;
  char *buffer;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_PLESSEY_Barcode(): ");
  scanf("%254s", buffer);
  result=Print_PLESSEY_Barcode(buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

Print_PDF417_Barcode()

This function prints barcode PDF417.

Parameters:

E, H, W, N, "String" where:

E = Error correction level – Integer between 0 to 8

Error correction is done introducing redundancies in the PDF417 code. The higher the error correction level the easier it is to retrieve the code. However, printing will become larger and the quantity of data will be lower.

H = Height – Integer from 1 to 8

Code character height (pitch). 1 pitch = 0,125 mm height

W = Width – Integer from 1 to 4

Code character width (pitch). 1 pitch = 0,125 mm height

N = Number of columns printed in the line – Integer from 1 to 30

"0" (zero) uses the maximum number of columns the device allows for the informed width (pitch). If not fitting into the line, the printer will automatically adjust to the maximum number of columns allowed in the line.

String - String with the barcode content to be generated.

Example:

```
int main()
{
  int result;
  char *buffer;
  int ecc;
  int height;
  int width;
  int columns;

  buffer=(char *) malloc(256);
  printf("\n\nEnter bar code text for Print_PDF417_Barcode(): ");
  scanf("%254s", buffer);
  printf("\nEnter ECC level (0 - 8): ");
  scanf("%d", &ecc);
  printf("\nEnter height (1 - 8): ");
  scanf("%d", &height);
  printf("\nEnter width (1 - 4): ");
  scanf("%d", &width);
  printf("\nEnter number of columns (0 - 575): ");
  scanf("%d", &columns);
  result=Print_PDF417_Barcode(ecc, height, width, columns, buffer);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

ConfigureBarcode()

This function sets up the barcodes, establishing Height, Width and character position.

Parameters:

H, W, P, F, M where

H = Height – Integer from 1 to 255 (default is 162) – where the printer barcode will have a height of *H* x 0.125mm. The default (162) represents a 20.25mm (0.8 inches) height.

W = Width – Integer from 0 to 2.

Width = 0 (thin bars)
Width = 1 (medium bars) - default
Width = 2 (thick bars)

P = Position of the HRI – Integer from 0 to 3. The HRI is the “human readable information” to be printed with the code.

Position = 0 (will not print HRI)
Position = 1 (will print HRI above the code)
Position = 2 (will print HRI below the code) - default
Position = 3 (will print HRI both above and below the code)

F = Font of the HRI – Integer from 0 to 1.

Font = 0 (normal)
Font = 1 (condensed)

M = Left Margin position - Integer from 0 to 575 (dots pitch). This can be interpreted as the distance from the left margin to the beginning of the barcode, given by *M* x 0.125mm. The default is 0 (zero).

Example:

```
int main()
{
  int result;
  int height;
  int width;
  int position;
  int font;
  int margin;

  printf("\nEnter height (1 - 255): ");
  scanf("%d", &height);
  printf("\nEnter width (0=thin; 1=medium; 2=thick): ");
  scanf("%d", &width);
  printf("\nEnter characters position (0=none; 1=above; 2=below; 3=both): ");
  scanf("%d", &position);
  printf("\nEnter font (0=normal; 1=condensed): ");
  scanf("%d", &font);
  printf("\nEnter margin (0 - 575): ");
  scanf("%d", &margin);
  result=ConfigureBarcode(height, width, position, font, margin);
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

 0 (zero)   : Communication Error.
 1 (one)   : OK.
-1 (minus one) : Run Error.
-2 (minus two) : Invalid Parameter.

EnableRetractableFcn()

Enables or Disables the retractable function of the Presenter.

Parameter:

iFlag: Variable INTEGER type, 1 (one) digit size. Where:

1 = Enabled
0 = Disabled

Example:

```
int main()
{
  int result;
  int retractiblepresenter;

  buffer=(char *) malloc(256);
  printf("\nEnable retractible presenter (1=yes; 0=no): ");
  scanf("%d", &retractiblepresenter);
  result=EnableRetractableFcn(retractiblepresenter);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

0 (zero)	: Communication Error.
1 (one)	: OK.
-1 (minus one)	: Run Error.
-2 (minus two)	: Invalid Parameter.

ConfigureRetractableFcn()

Programs the waiting time before paper retraction, in case the paper is not removed from the Presenter exit.

Parameter:

iTime: Variable INTEGER type, from 0 (zero) to 60 seconds, establishing waiting time. Default = 5 (five) seconds. If the time is 0 (zero) the waiting time will be infinite.

Example:

```
int main()
{
  int result;
  int timeout;

  buffer=(char *) malloc(256);
  printf("\nEnter timeout for retractible presenter (0 - 60): ");
  scanf("%d", &timeout);
  result=ConfigureRetractableFcn(timeout);
  show_error(result);
  free(buffer);
}
```

Response for this function is given by an integer, where the response is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-1 (minus one)  : Run Error.
-2 (minus two)  : Invalid Parameter.
```

CheckPaperInPresenter()

Verify if paper is positioned inside the presenter.

Parameter:

None

Example:

```
int main()
{
  int result;

  result=CheckPaperInPresenter();
  show_error(result);
}
```

Response for this function is given by an integer, where the response is:

-1 (minus on) : Run error.
0 (zero) : Problems in the verification of the paper in presenter.
1 (one) : Paper located in presenter.
2 (two) : Paper not located in presenter.
3 (three) : Unknown error.

Read_Extended_Status ()

Read and returns 5 bytes status information (USB and serial interface only)

Parameter:
statusBuffer: 5 Bytes array to receive raw binary status information from the printer, according to the following:

Status Byte 1: Communications and Buffer

Bit no.	7	6	5	4	3	2	1	0
Value	1	Buffer status		0	On / Off line	Error overrun	0	0

Bit 2: 0: data received will be printed
1: data received will be lost

Bit 3: 0: on-line
1: off-line

Bits 5 & 6: 00: buffer empty
01: buffer less 1/3 full
10: buffer more 1/3 full
11: buffer more ¾ full

Status Byte 2: Sensors and Error

Bit no.	7	6	5	4	3	2	1	0
Value	1	Error	Paper Sensor	Presenter Sensor	Head-up sensor	Paper near and sensor	0	1

Bit 2: 0: paper near and sensor with paper
1: paper near and sensor without paper

Bit 3: 0: head-up locked (head is down)
1: head-up unlocked (head is up)

Bit 4: 0: presenter sensor without paper
1: presenter sensor with paper

Bit 5: 0: paper sensor with paper
1: paper sensor without paper

Bit 6: 0: printer without error
1: printer with error

Status Byte 3: Error detail

Bit no.	7	6	5	4	3	2	1	0
Value	1	0	Non recoverable error	1	Paper cutter error	0	0	0

Bit 3: 0: cutter without error
1: cutter with error

Bit 5: 0: error is recoverable – a reset may solve the problem, but a test receipt must be sent after the reset to double check system operation
1: error is non-recoverable – service intervention is necessary

Status Byte 4: Printhead and Paper Operation

Bit no.	7	6	5	4	3	2	1	0
Value	1	0	Internal paper jam	1	Paper eject error	Head temperature	0	1

Bit 2: 0: normal printhead temperature
1: printhead temperature above normal

Bit 3: 0: paper with no presenting errors
1: printer with paper presenting error - error in paper ejection

Bit 5: 0: printer with no internal paper jams
1: printer with internal paper jam error – paper did not reach presenter

Status Byte 5: Firmware Version

Bit no.	7	6	5	4	3	2	1	0
Value	0	Major firmware version digit			Minor firmware version digit			

Example:

```
int main()
{
  char status[5];

  Read_Extended_Status(status);
  if ( status[1] & 64 ) //bit 6 (status byte 2) : printer has error (s)
  {
       if ( status[2] & 8 )//bit 3 (status byte 3) : cutter error
       {
              //cutter error //do something here
       }
       …
  }

}
```

The return for this function is given by an integer, where the return is:

```
 0 (zero)       : Communication Error.
 1 (one)        : OK.
-2 (minus two) : Invalid Parameter.
-5 (minus five): Indicates wrong printer model
```

API Function Summary / Glossary

Name	Function	Page
AuthenticateDoc	Validates document	16
BematechTX	Sends a text / ASCII string	13
CheckDocInserted()	Checks for presence of a document for validation	15
CheckPaperInPresenter	Checks for paper inside the presenter	44
ClosePort	Closes Communication Port	20
CommandTX	Sends Commands	14
ConfigPrinterModel	Configures the Printer Model	21
ConfigSlipSize	Sets the maximum ticket length in log mode	22
ConfigureBarcode	Configures Barcodes	41
ConfigureRetractableFcn	Programs wait time before retracting	43
CutPaper	Activates cutter	27
EnableLongSlip	Enables / disables custom ticket length in log mode	23
EnableRetractableFcn	Enables retractable presenter function	42
EnableWaitPrinting	Enables the wait for the end of print job	24
FormatTX	Sends Text	11
GraphicChar	Sends Bit image	19
InitializePort	Opens the Communication Port	10
Print_CODABAR_Barcode	Prints Codabar Barcode	34
Print_CODE128_Barcode	Prints Code128 Barcode	36
Print_CODE39_Barcode	Prints Code39 Barcode	32
Print_CODE93_Barcode	Prints Code93 Barcode	35
Print_EAN13_Barcode	Prints EAN13 Barcode	30
Print_EAN8_Barcode	Prints EAN8 Barcode	31
Print_ISBN_Barcode	Prints ISBN Barcode	37
Print_ITF_Barcode	Prints ITF Barcode	33
Print_MSI_Barcode	Prints MSI Barcode	38
Print_PDF417_Barcode	Prints PDF417 Barcode	40
Print_PLESSEY_Barcode	Prints Plessey Barcode	39
Print_UPCA_Barcode	Prints UPCA Barcode	28
Print_UPCE_Barcode	Prints UPCE Barcode	29
Read_Status	Reads Printer Status	17
ReadDrawerStatus	Reads cash drawer status	26
SelectCodePage	Selects the printer's internal character table	11
WaitPrinting	Waits for the end of print job	25