# Driver Installation and Test Application Usage Manual

# Linux 32-bit API "libmp2032.so"

For POS and Kiosk printers

Driver Version: 1.0, release 1

BemaTECH INTERNATIONAL

# 1 PACKAGE DESCRIPTIONS

These *drivers* are available in RPM packages, which is a widely adopted package format in various Linux distributions.

The following packages are available:

| Package Name | Description |
|---|---|
| 1. bemathermal-apidrv | API driver compatible with MP2032.dll (Bematech Windows API) for kiosk and mini-printers. This package contains a shared library (**/usr/lib/libmp2032.so**) and a test application (**/usr/bin/test-mp2032**). Specific communication libraries (serial and parallel ports) are required (please see "Package Dependencies" below). |
| 2. bemathermal-apidrv-devel | Version of the library containing the files needed for application development. Contains a C language header file (**/usr/include/mp2032/mp2032.h**), and the source code for the provided test application (located under **/usr/src/mp2032**). |

## 1.1 Related Files

The following files refer to the packages above:

| Package # | File Name | Description |
|---|---|---|
| 1 | bemathermal-apidrv-1.0-1.i386.rpm | RPM instalable package containing the Bematech MP2032 shared library. Users' applications (non spooler) should be dinamically linked against this library (**libmp2032.so**). |
| 2 | bemathermal-apidrv-devel-1.0-1.i386.rpm | RPM installable package containing files needed for application development using the MP2032 library (**libmp2032.so**). This module is destinated to application developers only. |

## 1.2 Package Dependencies

For each provided RPM package, some extra packages may be needed for installation:

| **Package Name** | **Packages Needed for Installation** |
|---|---|
| 1. bemathermal-apidrv | -- No package dependencies -- |
| 2. bemathermal-apidrv-devel | bemathermal-apidrv |

# 2 INSTALLATION INSTRUCTIONS

## 2.1 Package Installation

Package installation can be done using any graphical utility which is suitable for installing RPM packages. Nevertheless, the most simple and direct way of installing RPM packages is using the **rpm** command line utility.

Please follow the steps below to install the provided packages successfully:

1. Open a terminal (command line shell) as the "**root**" user;
2. Copy the necessary RPM files to a temporary directory. On the following example, we considered that the RPM packages are located on a diskette, and we are going to copy them to the **/root** directory:

```
# mkdir /mnt/floppy (if needed)
# mount /dev/fd0 /mnt/floppy
# cp /mnt/floppy/bemathermal-apidrv-1.0-1.i386.rpm /root
# umount /mnt/floppy
```

3. Install the packages using the **rpm** command. On the following example, we considered that the necessary RPM packages were already copied to the **/root** directory on the target machine:

```
# cd /root
# rpm -ivh bemathermal-apidrv-1.0-1.i386.rpm
Preparing...                ########################################### [100%]
   1:bemathermal-apidrv     ########################################### [100%]
```

## 2.2 Adjusting Permissions

The API driver, and so the applications linked against it, read and write directly to the physical device where the printer is connected to.

Under Linux this means that any user wanting to use such an application will need read and write access to the device files corresponding to the device port. The most common configuration under the various Linux distributions is:

| Device Type | Device Files | Default Permissions |
|---|---|---|
| Parallel | /dev/lp* | `crw-rw----  1 root lp` |
| Serial | /dev/ttyS[0-9]* | `crw-rw----  1 root uucp` |
| USB | /dev/ttyUSB* | `crw-rw----  1 root uucp` |

Considering that on most environments the real users do not run applications as the “**root**” user, there is a chance you will need to adjust some permissions.

There are several ways to do this, the most common being:

1. To add the user to the group owner of the device file in question. Example: add user “**bob**” to group “**lp**” to give him access to the parallel ports; or
2. To change permissions of the device file in question so that anyone can write to it; or
3. To change user or group ownership of the device file in question to correspond to the desired user or group of users of the device file.

The most simpler and elegant solution is the first one, where the system administrator simply adds an user to the group owner of the desired device file, and everything is set up. The example below adds user “**bob**” to group “**lp**”, thus giving him access to the parallel ports:

```
# gpasswd –a bob lp
Added user bob to group lp
```

The second solution is not recommended since it leaves the system too open for accesses from undesired users.

The last one, although not as elegant as the first one, is also acceptable, since just the selected users will have access to the device files. It deserves notice that there is a small chance of some programs to break because of this change[1].

For example, if a device has to be used simultaneously by the API driver and by the spooler driver, the system printing spooler will need to have access to the device file where the printer is connected to. The printing spooler usually runs as group "**lp**" when using parallel ports, so if you change ownership of the parallel ports this will probably make the spooler to malfunction.

[1] On Red Hat Linux systems the PAM_console module comes configured for giving ownership over various device files for the user logged in to the physical machine console. The desired device files could be included by editing the **/etc/security/console.perms** file.

# 3 TEST APPLICATION USAGE

The provided test application (**test-mp2032**) is a very simple, although complete, test application for the MP2032 API driver under Linux.

This application provides the user the opportunity of calling any of the API functions. Although this is a little bit technical, it allows the programmer to check functionality before coding the final application.

## 3.1 Running the Test Application

1. The first and mandatory step is to install the drivers as described in the previous section, and to give necessary access to the devices files for the user you'd like to run the application with. If you are unsure about how to configure the permissions correctly (as described in section 2.2), you have the option to run the test application as the "**root**" user. Be warned, though, that most commands and actions you take will be done without any warnings or confirmation requests, thus posing a damage possibility to your installed Linux system;
2. To run the test application, open any "text" terminal (command line shell) as the user who has read/write access to the device file corresponding to the port your equipment is connected to;
3. Under the shell prompt, type the name of the test application program (**test-mp2032**):

```
$ test-mp2032
...
```

4. You'll get a screen similar to the example below. On the following examples, any text typed by the user is highlighted in yellow. Any important messages from the application are highlighted in green:

```
[user@machine user]$ test-mp2032

Bematech MP2032 API - Test Application

 1 - InitializePort()           17 - Print_UPCA_Barcode()
 2 - FormatTX()                 18 - Print_UPCE_Barcode()
 3 - BematechTX()               19 - Print_EAN13_Barcode()
 4 - CommandTX()                20 - Print_EAN8_Barcode()
 5 - CheckDocInserted()         21 - Print_CODE39_Barcode()
 6 - AuthenticateDoc()          22 - Print_ITF_Barcode()
 7 - Read_Status()              23 - Print_CODABAR_Barcode()
 8 - GraphicChar()              24 - Print_CODE93_Barcode()
 9 - ClosePort()                25 - Print_CODE128_Barcode()
10 - ConfigPrinterModel()       26 - Print_ISBN_Barcode()
11 - ConfigSlipSize()           27 - Print_MSI_Barcode()
12 - EnableLongSlip()           28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()       29 - Print_PDF417_Barcode()
14 - WaitPrinting()             30 - ConfigureBarcode()
15 - ReadDrawerStatus()         31 - EnableRetractableFcn()
16 - CutPaper()                 32 - ConfigureRetractableFcn()
                                33 - CheckPaperInPresenter()
99 - Exit                       34 - SetSerialBaudRate()
                                35 - SelectCodePage()
Your option:
```

5. The user interface is very simple. You have just to type in the number corresponding to the API function you'd like to test. According to the API specification, the only function that can be used before initializing the device port is the "Set SerialBaudRate()" function. So, before doing anything, we have to "open" the port. In the next example we'll "open" the first parallel port on the machine:

```

 1 - InitializePort()           17 - Print_UPCA_Barcode()
 2 - FormatTX()                 18 - Print_UPCE_Barcode()
 3 - BematechTX()               19 - Print_EAN13_Barcode()
 4 - CommandTX()                20 - Print_EAN8_Barcode()
 5 - CheckDocInserted()         21 - Print_CODE39_Barcode()
 6 - AuthenticateDoc()          22 - Print_ITF_Barcode()
 7 - Read_Status()              23 - Print_CODABAR_Barcode()
 8 - GraphicChar()              24 - Print_CODE93_Barcode()
 9 - ClosePort()                25 - Print_CODE128_Barcode()
10 - ConfigPrinterModel()       26 - Print_ISBN_Barcode()
11 - ConfigSlipSize()           27 - Print_MSI_Barcode()
12 - EnableLongSlip()           28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()       29 - Print_PDF417_Barcode()
14 - WaitPrinting()             30 - ConfigureBarcode()
15 - ReadDrawerStatus()         31 - EnableRetractableFcn()
16 - CutPaper()                 32 - ConfigureRetractableFcn()
                                33 - CheckPaperInPresenter()
99 - Exit                       34 - SetSerialBaudRate()
                                35 - SelectCodePage()
Your option: 1


Enter device port name (ex: /dev/parport0; /dev/ttyS0; etc): /dev/parport0
```

6. If everything goes well, we'll get an "ok" status at the top of the screen. This status represents the return code from the function, so this code can have different meanings, depending on which function you're trying to call:

```
** STATUS = Command successfull OR Drawer opened OR Paper is not in presenter.

Bematech MP2032 API - Test Application

 1 - InitializePort()          17 - Print_UPCA_Barcode()
 2 - FormatTX()                18 - Print_UPCE_Barcode()
 3 - BematechTX()              19 - Print_EAN13_Barcode()
 4 - CommandTX()               20 - Print_EAN8_Barcode()
 5 - CheckDocInserted()        21 - Print_CODE39_Barcode()
 6 - AuthenticateDoc()         22 - Print_ITF_Barcode()
 7 - Read_Status()             23 - Print_CODABAR_Barcode()
 8 - GraphicChar()             24 - Print_CODE93_Barcode()
 9 - ClosePort()               25 - Print_CODE128_Barcode()
10 - ConfigPrinterModel()      26 - Print_ISBN_Barcode()
11 - ConfigSlipSize()          27 - Print_MSI_Barcode()
12 - EnableLongSlip()          28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()      29 - Print_PDF417_Barcode()
14 - WaitPrinting()            30 - ConfigureBarcode()
15 - ReadDrawerStatus()        31 - EnableRetractableFcn()
16 - CutPaper()                32 - ConfigureRetractableFcn()
                               33 - CheckPaperInPresenter()
99 - Exit                      34 - SetSerialBaudRate()
                               35 - SelectCodePage()
Your option:
```

7. Using the functions is straight forward: you call the function by it's number and the test application asks you all necessary parameters for it. It's your responsibility to call the functions in a sensible sequence, since the application is just an interface for the API. The example below calls the "FormatTX()" function, which requires a few parameters:

```
11 - ConfigSlipSize()          27 - Print_MSI_Barcode()
12 - EnableLongSlip()          28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()      29 - Print_PDF417_Barcode()
14 - WaitPrinting()            30 - ConfigureBarcode()
15 - ReadDrawerStatus()        31 - EnableRetractableFcn()
16 - CutPaper()                32 - ConfigureRetractableFcn()
                               33 - CheckPaperInPresenter()
99 - Exit                      34 - SetSerialBaudRate()
                               35 - SelectCodePage()
Your option: 2


Enter text to be printed using FormatTX(): test-for-bematech-printers
```

```
Letter type (0=condensed;1=normal;2=elite)1

Italic? (1=yes; 0=no)1

Underline? (1=yes; 0=no)1

Expanded? (1=yes; 0=no)1

Emphasized? (1=yes; 0=no)1
```

8. When you finish entering the last parameter, the function executes immediately. The function's return code of the always appear at the top of the screen. One of the most interesting features of the printer is supported by the API. It's possible to force an application to block until all buffer contents are already printed. To do this, you have to call "EnableWaitPrinting()":

```
 1 - InitializePort()           17 - Print_UPCA_Barcode()
 2 - FormatTX()                 18 - Print_UPCE_Barcode()
 3 - BematechTX()               19 - Print_EAN13_Barcode()
 4 - CommandTX()                20 - Print_EAN8_Barcode()
 5 - CheckDocInserted()         21 - Print_CODE39_Barcode()
 6 - AuthenticateDoc()          22 - Print_ITF_Barcode()
 7 - Read_Status()              23 - Print_CODABAR_Barcode()
 8 - GraphicChar()              24 - Print_CODE93_Barcode()
 9 - ClosePort()                25 - Print_CODE128_Barcode()
10 - ConfigPrinterModel()       26 - Print_ISBN_Barcode()
11 - ConfigSlipSize()           27 - Print_MSI_Barcode()
12 - EnableLongSlip()           28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()       29 - Print_PDF417_Barcode()
14 - WaitPrinting()             30 - ConfigureBarcode()
15 - ReadDrawerStatus()         31 - EnableRetractableFcn()
16 - CutPaper()                 32 - ConfigureRetractableFcn()
                                33 - CheckPaperInPresenter()
99 - Exit                       34 - SetSerialBaudRate()
                                35 - SelectCodePage()
Your option: 13


Enable waiting for end of printing (1=yes; 0=no): 1
```

9. With this feature enabled, you can print some reasonable amount of text and immediatelly call "WaitPrinting()" to block until everything is on paper. This is useful for an application to "know" when to ask the user to tear off the paper (or to pull off the printed slip or ticket). In the next example, we'll print the **/etc/group** file, which is large enough to fill in the printer buffer. Remember that immediately after

printing the text we should call "WaitPrinting()" for the application to block:

```
 1 - InitializePort()          17 - Print_UPCA_Barcode()
 2 - FormatTX()                18 - Print_UPCE_Barcode()
 3 - BematechTX()              19 - Print_EAN13_Barcode()
 4 - CommandTX()               20 - Print_EAN8_Barcode()
 5 - CheckDocInserted()        21 - Print_CODE39_Barcode()
 6 - AuthenticateDoc()         22 - Print_ITF_Barcode()
 7 - Read_Status()             23 - Print_CODABAR_Barcode()
 8 - GraphicChar()             24 - Print_CODE93_Barcode()
 9 - ClosePort()               25 - Print_CODE128_Barcode()
10 - ConfigPrinterModel()      26 - Print_ISBN_Barcode()
11 - ConfigSlipSize()          27 - Print_MSI_Barcode()
12 - EnableLongSlip()          28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()      29 - Print_PDF417_Barcode()
14 - WaitPrinting()            30 - ConfigureBarcode()
15 - ReadDrawerStatus()        31 - EnableRetractableFcn()
16 - CutPaper()                32 - ConfigureRetractableFcn()
                               33 - CheckPaperInPresenter()
99 - Exit                      34 - SetSerialBaudRate()
                               35 - SelectCodePage()
Your option: 3


Enter file name to be printed using BematechTX(): /etc/group
```

10. Now we quickly call "WaitPrinting()":

```
Your option: 14    < now the application blocks until all data is printed >

Bematech MP2032 API - Test Application

 1 - InitializePort()          17 - Print_UPCA_Barcode()
 2 - FormatTX()                18 - Print_UPCE_Barcode()
 3 - BematechTX()              19 - Print_EAN13_Barcode()
 4 - CommandTX()               20 - Print_EAN8_Barcode()
 5 - CheckDocInserted()        21 - Print_CODE39_Barcode()
 6 - AuthenticateDoc()         22 - Print_ITF_Barcode()
 7 - Read_Status()             23 - Print_CODABAR_Barcode()
 8 - GraphicChar()             24 - Print_CODE93_Barcode()
 9 - ClosePort()               25 - Print_CODE128_Barcode()
10 - ConfigPrinterModel()      26 - Print_ISBN_Barcode()
11 - ConfigSlipSize()          27 - Print_MSI_Barcode()
12 - EnableLongSlip()          28 - Print_PLESSEY_Barcode()
13 - EnableWaitPrinting()      29 - Print_PDF417_Barcode()
14 - WaitPrinting()            30 - ConfigureBarcode()
15 - ReadDrawerStatus()        31 - EnableRetractableFcn()
16 - CutPaper()                32 - ConfigureRetractableFcn()
                               33 - CheckPaperInPresenter()
99 - Exit                      34 - SetSerialBaudRate()
                               35 - SelectCodePage()
Your option:
```

11. To stop working with the API you should always call "ClosePort()". But just in case the programmer forgets to call this function, the API driver

releases the port automatically to the operating system.