MAGTEK®
SECURITY FROM THE INSIDE

MagTek Universal SDK

For MMS Devices

Test Console Manual (Linux)

May 2024

Manual Part Number:
D998200580-101

REGISTERED TO ISO 9001:2015

Table 0.1 – Revisions

Rev Number	Date	Notes
100	February 1, 2024	Initial release.
101	May 9, 2024	- Added instructions for building the test console at section 1.1 and 1.2. - Added instructions for configuration files at section 3. - Updated to show Event Driven transaction mode at sections 3.1, 3.2, 3.3, and 3.4.

SOFTWARE LICENSE AGREEMENT

IMPORTANT: YOU SHOULD CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE INSTALLING THE SOFTWARE PACKAGE. YOUR INSTALLATION OF THE SOFTWARE PACKAGE PRESUMES YOUR ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ASSOCIATED DOCUMENTATION TO THE ADDRESS ON THE FRONT PAGE OF THIS DOCUMENT, ATTENTION: CUSTOMER SUPPORT.

TERMS, CONDITIONS, AND RESTRICTIONS

MagTek, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software."

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with MagTek products. LICENSEE MAY NOT COPY, MODIFY, OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble, or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass, or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without the prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except for archival purposes or to load for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions, or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded are free from defects in material or workmanship under normal use.

THE SOFTWARE IS PROVIDED AS IS. LICENSOR MAKES NO OTHER WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE, OR INABILITY TO USE, THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void, or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall inure to the benefit of MagTek, Incorporated, its successors or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS, AND RESTRICTIONS, AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL VERBAL AND WRITTEN COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO MAGTEK, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ADDRESS LISTED IN THIS DOCUMENT, OR E-MAILED TO SUPPORT@MAGTEK.COM.

DEMO SOFTWARE / SAMPLE CODE: Unless otherwise stated, all demo software and sample code are to be used by Licensee for demonstration purposes only and MAY NOT BE incorporated into any production or live environment. The PIN Pad sample implementation is for software PIN Pad test purposes only and is not PCI compliant. To meet PCI compliance in production or live environments, a third-party PCI compliant component (hardware or software-based) must be used.

Table of Contents

1 Introduction

This document provides instructions to use the MTUSDKNET Test Console. This test EMV transactions, sends configuration files and updates firmware.

It is part of a larger library of documents designed to assist Secure Card Readers implementers, which includes the following documents available from MagTek:

- ***D998200570 MAGTEK UNIVERSAL SDK PROGRAMMER'S MANUAL (LINUX)***
- ***D998200489 DYNAPROX PROGRAMMER'S MANUAL (COMMANDS)***
- ***D998200383 DYNAFLEX PRODUCTS PROGRAMMER'S MANUAL (COMMANDS)***

1.1 Building Default Test Console

This section is for building the default test console.

Before building, all dependencies must be present. See instructions for dependences in document **D998200570 MAGTEK UNIVERSAL SDK PROGRAMMER'S MANUAL (LINUX)**

1) Unzip the package.
2) Using a terminal, run make in the same folder the package is unzipped to build the mtscra and mtusdk libraries and the test console.

```
$ make
```

3) Go to the folder: mtusdk_test.

```
$ cd mtusdk_test
```

4) The output for the test consoles are available:

`mtusdk_test`	(For SCRA devices)
`mtusdk_test.mms`	(For DynaFlex devices)

5) The static library `libmtusdk.a` will be placed in the `/mtusdk_test` folder.
6) The shared object libraries `libmtscra.so` and `libmtmms.so` (SCRA and DynaFlex respectively) will be placed in the `/mtusdk_test` folder.
7) These libraries must be present to rebuild and run the test console.

1.2 Building Adjusted Test Console

This section is for building an adjusted test console after the initial default build. To make adjustments, follow the steps below.

1) After the default build of the test console, changes may be needed to start a transaction.
2) Modifying the Payment Method according to the device to be tested.
   - Open file `main.cpp`
   - Search for **`get_input`** and adjust the call to function **`test_emv()`**

```
get_input:
        int inchar = std::getchar();
        if (inchar == '1')
        {
                bQuickChip = true;
                test_emv(dev, deviceType == MMS ? (PAYMENT_METHOD_CONTACTLESS +
PAYMENT_METHOD_NFC) : PAYMENT_METHOD_CONTACT);
        }
        else if (inchar == '2')
        {
                bQuickChip = false;
                test_emv(dev, deviceType == MMS ? (PAYMENT_METHOD_CONTACTLESS +
PAYMENT_METHOD_NFC) : PAYMENT_METHOD_CONTACT);
                }
```

   - For DynaWave
     Set the payment method as `PAYMENT_METHOD_CONTACTLESS`.

```
test_emv(dev, PAYMENT_METHOD_CONTACTLESS);
```

   - For mDynamo
     Set the payment method as `PAYMENT_METHOD_CONTACT`.

```
test_emv(dev, PAYMENT_METHOD_CONTACT);
```

   - For DynaProx
     Set the payment method as `PAYMENT_METHOD_CONTACTLESS` + `PAYMENT_METHOD_NFC`

```
test_emv(dev, deviceType == MMS ? (PAYMENT_METHOD_CONTACTLESS + PAYMENT_METHOD_NFC) :
PAYMENT_METHOD_CONTACT);
```

   - For DynaFlex/DynaFlex II/DynaFlex II PED
     Set the payment method as `PAYMENT_METHOD_CONTACT` + `PAYMENT_METHOD_CONTACTLESS` + `PAYMENT_METHOD_NFC`
     MSR may also be added if needed. `PAYMENT_METHOD_MSR`

```
test_emv(dev, deviceType == MMS ? (PAYMENT_METHOD_MSR + PAYMENT_METHOD_CONTACT +
PAYMENT_METHOD_CONTACTLESS + PAYMENT_METHOD_NFC) : PAYMENT_METHOD_CONTACT);
```

3) Save changes to `main.cpp` and rebuild.
   For SCRA devices `$ make -f makefile`
   or
   For DynaFlex devices `$ make -f makefile.mms`

2 How to Connect to MTUSDK Test Console

To connect via an interface, follow these steps:

2.1 Connect via USB Interface

1) Connect the device to the USB port of the Linux computer.
2) The first time, allow a moment for the host to recognize the device.
3) The console automatically connects to the first USB DynaFlex or DynaProx it finds.

2.2 Connect via Serial Interface

1) Connect the device to the Serial port of the Linux computer.
2) Edit `main.cpp` file.
3) Below the `reopen:` label, set the proper Serial connection type by uncommenting the "`dev = CoreAPI`" that matches the port to be used.

```
reopen:

      auto devices = CoreAPI::getDeviceList();
      IDevice* dev = NULL;

      // USB device detected ?
      if (devices.size() > 0)
      {
            // open first USB device
            dev = devices.at(0);
            //

      }
      else
      {
            //dev = CoreAPI::getDevice(MTU_DEVICE_TYPE::SCRA,
MTU_DEVICE_CONNECTION_TYPE::MTU_SERIAL, "port=/dev/ttyS1");


            // open a DynaProx from ttyS1
            //dev = CoreAPI::getDevice(MTU_DEVICE_TYPE::MMS,
MTU_DEVICE_CONNECTION_TYPE::MTU_SERIAL, "port=/dev/ttyS1");

#ifdef TEST_MMS
            // open a DynaProx from serial0
            dev = CoreAPI::getDevice(MTU_DEVICE_TYPE::MMS,
MTU_DEVICE_CONNECTION_TYPE::MTU_SERIAL, "port=/dev/serial0");
#else
            // open a mDynamo from serial0
            dev = CoreAPI::getDevice(MTU_DEVICE_TYPE::SCRA,
MTU_DEVICE_CONNECTION_TYPE::MTU_SERIAL, "port=/dev/serial0");
#endif

            // open a mDynamo from USB to RS232 port (first one)
            //dev = CoreAPI::getDevice(MTU_DEVICE_TYPE::SCRA,
MTU_DEVICE_CONNECTION_TYPE::MTU_SERIAL, "port=/dev/ttyUSB0");
      }
```

4) After editing `main.cpp`, rebuild the test console.
   For MMS devices use: `make -f makefile.mms`.
   The output file is `mtusdk_test.mms`.

For SCRA devices use: `make -f makefile`
The output file is `mtusdk_test`.

3 How to use the MTUSDK Test Console

The following instructions are for using the MTUSDK Test console on a Linux operating system.
In these examples, lines are removed and replaced with ". . ." for readability.

Make sure any image, config on JSON files to be demonstrated are located in the current folder of the console app. For this sample console, the following files must be present. The samples provided in the package are only for testing the console. These files ultimately need to be configured for your use case.

Copy from Sample\
    cfg0000000.bin
    image.bmp
    magensa.json

Paste to Sample\mtusdk_test
    cfg0000000.bin
    image.bmp
    magensa.json

3.1 Launch Test Console

1) For testing DynaFlex family of devices (MMS – mtusdk library).

```
$ sudo ./mtusdk_test.mms
```

2) For testing V5 family of devices (MTSCRA – mtscra library).

```
$ sudo ./mtusdk_test
```

Note: Many of the selection options are not shown in the test console when a V5 device is attached.

3.2 EMV Transaction

1) To perform an EMV transaction, select operation 1 or 2.

```
Select an operation :
 1. Transaction (Quick Chip)
 2. Transaction (Full)
 3. Send configuration file
 4. Update firmware
 5. Display a message
 6. Send an image and display
 9. Enable event driven transaction
 a. Disable event driven transaction
 0. Quit
```

2) The console calls `startTransaction()`
3) Continue the transaction by tapping a contactless card.
4) Message events are shown to tap and remove the card.

```
 <- (Display Message) WELCOME
[WELCOME]
 <- (Device Notification)
AA008104830018038204020102008482001418038101008201018382000854415020434
15244
 <- (Display Message) TAP CARD
[TAP CARD]
 <- (Device Notification)
AA008104830018038204020101008482001718038101008201018382000B52454D4F56
452043415244
 <- (Display Message) REMOVE CARD
[REMOVE CARD]
 <- (Device Notification) AA00810483000101820420010400
 <- (Transaction Status) card_detected
 <- (Device Notification)
```

5) In the event handler `OnEvent()`, an ARQC event is fired as:
   `EVENT_TYPE_AuthorizationRequest`

```
 <- (Request Authorization)
01E9F98201E5DFDF540AFFFF9876543210200015DFDF550182DFDF2507423632434135
46FA8201C3708201BFDFDF530100DFDF4D273B353434333030303034303033343535
3D3030. . .
```

6) The ARQC is forwarded to Magensa MPPGv4 webservice using the credentials within the file `magensa.json`.
7) If the selected operation was #2 Transaction (Full), the console sends the ARPC to the device by calling `sendAuthorization()`.

```
SendAuthorization - FF7413DFDF2507423531433646543FA0670048A023030
```

3.3 Send Configuration Files

1) To send a configuration file, select operation 3. File `cfg00000000.bin` must be in the current folder.

```
Select an operation :
 1. Transaction (Quick Chip)
 2. Transaction (Full)
 3. Send configuration file
 4. Update firmware
 5. Display a message
 6. Send an image and display
 9. Enable event driven transaction
 a. Disable event driven transaction
 0. Quit
```

2) The console calls `sendFile()` to send `cfg00000000.bin` (Terminal configuration).

   From this point, the SDK sends the file in packets to the device. After each packet is received by the device, the SDK is notified and triggers the callback `OnProgress()`. The console displays the progress event.

```
 <- (Operation Status) operation_done,0000D812,Operation Done,00000000
 <- (Device Response) AA0081048203D812820400000000
  progress (33)
  . . .
  progress (100)
  result (0)
```

3) The console shows the end result of each upload as `result (0)`. The result is handled within SDK event function `OnResult()`. 0 = success.

```
 <- (Operation Status) operation_done,0000D812,Operation Done,00000000
 <- (Device Response) AA0081048203D812820400000000
  progress (33)
  . . .
  progress (100)
  result (0)
sendfile(0000000000, filedata) -> 0
```

3.4 Update Firmware

1) To update the firmware on the device, select operation 4.

```
Select an operation :
 1. Transaction (Quick Chip)
 2. Transaction (Full)
 3. Send configuration file
 4. Update firmware
 5. Display a message
 6. Send an image and display
 9. Enable event driven transaction
 a. Disable event driven transaction
 0. Quit
```

2) The console calls `updateFirmware()`.

   From this point, the SDK sends the firmware in packets to the device. After each packet is received by the device, the SDK is notified and triggers the callback `OnProgress()`. The console displays each progress event.

```
  progress (1)
  . . .
  progress (68)
  progress (99)
  progress (100)
```

3) DO NOT POWER CYCLE OR DETACH FROM HOST during this time.

4) When the update is complete, the console displays as shown below. 0 is success.

```
  progress (100)
  result(0)
updateFirmware(1, filedata) -> 0
```

3.5 Other Operations

To perform other operations, press the specific key.

- 5 Shows a message on the device's display. Only for devices with display.
- 6 Sends and shows an image on the device's display. File `image.bmp` must be in the current folder. Only for devices with display.
- 9 Enables event driven transaction mode. Device must be reset to take effect. In this mode, the device sends a notification on detection of the payment method. The app then starts a transaction.
- a Disables event driven transaction mode.
- 0 Quits the test console.

```
Select an operation :
 1. Transaction (Quick Chip)
 2. Transaction (Full)
 3. Send configuration file
 4. Update firmware
 5. Display a message
 6. Send an image and display
 9. Enable event driven transaction
 a. Disable event driven transaction
 0. Quit
```