

# Machine Readable Travel Documents (MRTD) support in uFCoder library

Document version 1.0

## Table of contents

<b>Introduction</b>	<b>3</b>
<b>Library functions for MRTD support</b>	<b>4</b>
MRTD_MRZDataToMRZProtoKey	4
MRTD_MRZSubjacentToMRZProtoKey	5
MRTDAppSelectAndAuthenticateBac	6
MRTDFileReadBacToHeap	7
<b>ePassport MRTD Example</b>	<b>7</b>
<b>Revision history</b>	<b>12</b>



## Library functions for MRTD support

### MRTD\_MRZDataToMRZProtoKey

#### Function description

In order to get MRZ Proto Key needed in subsequent steps, you can call this function and pass it null terminated strings containing document number, document holder date of birth and document expiration date. After successful function execution MRZ Proto Key will be stored in a `mrz_proto_key` 25-byte array.

#### Function declaration (C language)

```
UFR_STATUS MRTD_MRZDataToMRZProtoKey(const char *doc_number,
                                       const char *date_of_birth,
                                       const char *date_of_expiry,
                                       uint8_t mrz_proto_key[25]);
```

#### Parameters

<code>doc_number</code>	Pointer to a null terminated string containing exactly 9 characters document number.
<code>date_of_birth</code>	Pointer to a null terminated string containing exactly 6 characters representing the date of birth in the "YYMMDD" format.
<code>date_of_expiry</code>	Pointer to a null terminated string containing exactly 6 characters representing expiration date in the "YYMMDD" format.
<code>mrz_proto_key</code>	This byte array will contain calculated MRZ proto-key after successful function execution. This array must have allocated at least 25 bytes prior calling this function.

### MRTD\_MRZSubjacentToMRZProtoKey

#### Function description

In order to get MRZ Proto Key needed in subsequent steps, in case of the TD3 MRZ format (88 totally character long), you can call this function and pass it null terminated string containing MRZ subjacent row. Example of the TD3 MRZ format printed on the eMRTD document looks like this:



### Function declaration (C language)

## Parameters

MRTDAppSelectAndAuthenticateBac

## Function description

Use this function to authenticate to the eMRTD NFC tag using BAC. This function establish security channel for communication. Security channel is maintained using `send_sequence_cnt` parameter and channel session keys are `ksenc` (for encryption) and `ksmac` (for calculating MAC).

### Function declaration (C language)

## Parameters

<b>mrz_proto_key</b>	MRZ proto-key acquired using prior call to MRTD_MRZDataToMRZProtoKey() or MRTD_MRZSubjacentToMRZProtoKey() function.
----------------------	--

<b>ksenc</b>	This array must have allocated at least 16 bytes prior calling this function. This array will contain session encryption key after successful function execution.
<b>ksmac</b>	This array must have allocated at least 16 bytes prior calling this function. This array will contain session key for calculating MAC after successful function execution.
<b>send_sequence_cnt</b>	After successful execution of this function, pointer to this 64-bit value should be saved and forwarded at every subsequent call to MRTDFileReadBacToHeap() and/or other functions for reading eMRTD.

## MRTDFileReadBacToHeap

### Function description

Use this function to read files from the eMRTD NFC tag. You can call this function only after successfully established security channel by the previously called

MRTDAppSelectAndAuthenticateBac() function. Session keys ksenc and ksmac, and also parameter send\_sequence\_cnt are acquired by the previously called

MRTDAppSelectAndAuthenticateBac() function. After the successful call to this function, \*output points to the file data read from a eMRTD file specified by the file\_index parameter. Buffer, in which the data is stored, is automatically allocated on memory heap during function execution. Maximum amount of data allocated can be 32KB. There is programmer responsibility to cleanup allocated data (i.e. by calling free(), the standard C function) after use.

### Function declaration (C language)

```
UFR_STATUS MRTDFileReadBacToHeap(const uint8_t *file_index,
                                  uint8_t **output,
                                  uint32_t *output_length,
                                  const uint8_t ksenc[16],
                                  const uint8_t ksmac[16],
                                  uint64_t *send_sequence_cnt);
```

### Parameters

<b>file_index</b>	Parameter that specifies the file we want to read from the eMRTD. This is pointer to byte array contains exactly two bytes designating eMRTD file. Those two bytes are file identificator (FID) and there is a list of FIDs: EF.COM = {0x01, 0x1E}
-------------------	--

	<p>EF.DG1 = {0x01, 0x01}</p> <p>EF.DG2 = {0x01, 0x02}</p> <p>EF.DG3 = {0x01, 0x03}</p> <p>EF.DG4 = {0x01, 0x04}</p> <p>EF.DG5 = {0x01, 0x05}</p> <p>EF.DG6 = {0x01, 0x06}</p> <p>EF.DG7 = {0x01, 0x07}</p> <p>EF.DG8 = {0x01, 0x08}</p> <p>EF.DG9 = {0x01, 0x09}</p> <p>EF.DG10 = {0x01, 0x0A}</p> <p>EF.DG11 = {0x01, 0x0B}</p> <p>EF.DG12 = {0x01, 0x0C}</p> <p>EF.DG13 = {0x01, 0x0D}</p> <p>EF.DG14 = {0x01, 0x0E}</p> <p>EF.DG15 = {0x01, 0x0F}</p> <p>EF.DG16 = {0x01, 0x10}</p> <p>EF.SOD = {0x01, 0x1D}</p>
<b>*output</b>	After the successful call to this function, this pointer points to the file data read from a eMRTD file specified by the file_index parameter. Buffer, in which the data is stored, is automatically allocated during function execution. Maximum amount of data allocated can be 32KB. There is programmer responsibility to cleanup allocated data (i.e. by calling free(), the standard C function) after use.
<b>output_length</b>	After the successful call to this function, this pointer is points to the size of the file data read from a eMRTD file specified by the file_index parameter.
<b>ksenc</b>	Session encryption key acquired using prior call to MRTDAppSelectAndAuthenticateBac() function.
<b>ksmac</b>	Session key for calculating MAC acquired using prior call to MRTDAppSelectAndAuthenticateBac() function.
<b>send_sequence_cnt</b>	This pointer should point to a 64-bit value initialized by the previously successful call to MRTDAppSelectAndAuthenticateBac() function. Pointer to this 64-bit value should be saved and forwarded at every subsequent call to this function and/or other functions used for reading eMRTD.



## ePassport MRTD Example

This example you can download from:

[https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-ePassport\\_mrttd.git](https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-ePassport_mrttd.git)

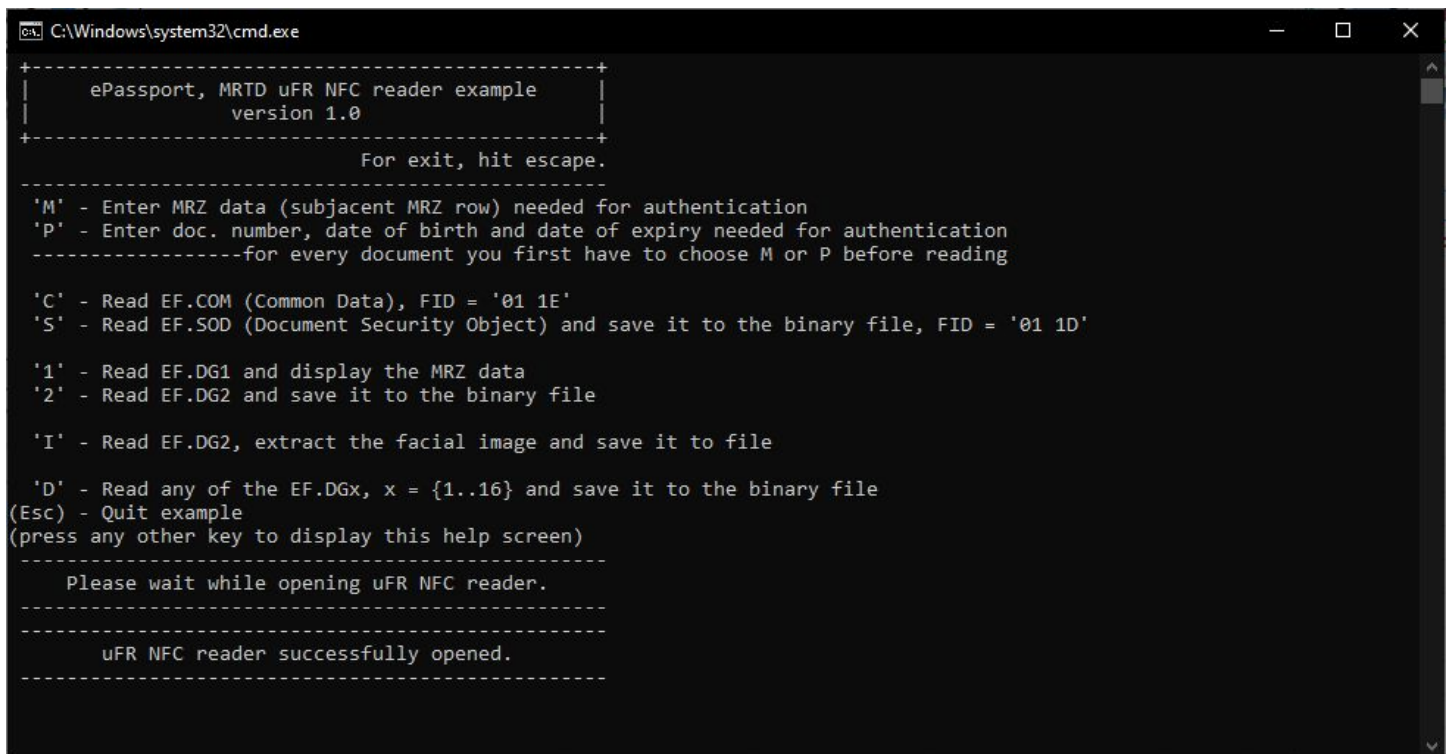
or clone the entire eclipse cdt project using:

`git clone --recursive https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-ePassport_mrttd.git`  
command.

If you want quick run only, download the project and start binary executable from the appropriate folder:

- for a 32-bit Windows start the win32\_release\run\_me.cmd
- for a 64-bit Windows start the win64\_release\run\_me.cmd
- for a 32-bit Linux start linux32\_release/ePassport\_mrttd
- for a 64-bit Linux start linux64\_release/ePassport\_mrttd

Software example requires uFR reader device to be attached and configured to the PC. No other application or service using uFR reader should be running on the computer. After successful start of the “ePassport MRTD Example” you will get screen like on figure 2.



```

C:\Windows\system32\cmd.exe
+-----+
| ePassport, MRTD uFR NFC reader example |
|          version 1.0                   |
+-----+
|                                     |
|           For exit, hit escape.       |
|-----|
'M' - Enter MRZ data (subjacent MRZ row) needed for authentication
'P' - Enter doc. number, date of birth and date of expiry needed for authentication
-----for every document you first have to choose M or P before reading

'C' - Read EF.COM (Common Data), FID = '01 1E'
'S' - Read EF.SOD (Document Security Object) and save it to the binary file, FID = '01 1D'

'1' - Read EF.DG1 and display the MRZ data
'2' - Read EF.DG2 and save it to the binary file

'I' - Read EF.DG2, extract the facial image and save it to file

'D' - Read any of the EF.DGx, x = {1..16} and save it to the binary file
(Esc) - Quit example
(press any other key to display this help screen)
+-----+
| Please wait while opening uFR NFC reader. |
|-----|
|                                     |
|           uFR NFC reader successfully opened. |
|-----|
  
```

figure 2: “ePassport MRTD Example” start screen

Now, you should choose one of the ‘M’ or ‘P’ options as stated in the application usage instructions on the screen.

If you chose ‘M’ option, you will be prompted with text:





```
EF.COM has been successfully read. File length is ?? bytes
Raw data: 60 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx ...
Parsing the EF.COM raw data:
LDS version is 01.07
UNICODE version is 04.00.00
Existing data groups list:
  Found: EF.DG1
  Found: EF.DG2
  Found: EF.DG3
  Found: EF.DG14
```

-----

Raw data in this example are masked and they can have any arbitrary value. Only the raw data tag has been present and it will be always the same (0x60). When you read your own document, you will get its actual raw data here. More about LDS version and UNICODE version you can read in the [ICA0 9303, part 10 document](#).

LDS and UNICODE version are followed by the data groups list that ePassport contains. Only DG1 and DG2 are mandatory. All the other data groups can be either present or not in the particular MRTD.

**'S'** - this option reads the document security object (EF.SO<sub>0</sub> elementary file) and save it to the binary file which path and name you have to enter when you prompted. Document security object contains digital signature in the standard [PKCS#7 CMS](#) format. Presence of the EF.SO<sub>0</sub> on the MRTD is mandatory.

**'1'** - this option reads the EF.DG1, parse it and displays raw and parsed data in the following format:

```
EF.DG1 has been successfully read. File length is ?? bytes
Raw data:
61 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx
xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx ...
Simple parsing the EF.DG1 raw data:
  Document code: P   (ePassport)
  Issuing State or organization: ???
  Name of holder: SURNAME  FIRST_NAME
  Document number: ??????????
  Nationality: ???
  Date of birth (dd.MM.yyyy.): ??..??..????..
  Sex: ?????
  Date of expiry (dd.MM.yyyy.): ??..??..????..
  Optional data: ????????????????
```

-----

Raw data in this example are masked and they can have any arbitrary value. Only the raw data tag has been present and it will be always the same (0x61). When you read your own document, you will get its actual raw data here.

**'2'** - this option reads the EF.DG2 and save it to the binary file which path and name you have to enter when you prompted. EF.DG2 contains document holder facial image and it is mandatory. EF.DG2 beside facial image could contain biometric facial features to. More about EF.DG2 content you can read in the [ICAO 9303, part 10 document](#).

**'I'** - this option reads the EF.DG2 to. In this case only the facial image is extracted from the MRTD file and saved to the file which path and name you have entered. Image format is automatically detected and the file extension is set according to it. There are two possible image file formats defined for this context: JPEG or JP2 (i.e. jpeg 2000).

**'D'** - this option reads any of the elementary data group (EF.DG) files from the MRTD and save it to the binary file which path and name you have to enter when you prompted. After this option has been chosen you will be prompted for EF.DG index. Index can be from the range 1 to 16 (e.g. 1 for EF.DG1 and 14 for EF.DG14). Elementary file you wanted to read must be listed in the EF.COM data groups list.

Reading of some optional elementary files, especially those containing biometric data, requires special security mechanisms which is outside the scope of this document.

Current version of the "ePassport MRTD Example" is 1.0 and depends on the uFCoder library version 5.0.12 and uFR firmware version 5.0.22.

## Revision history

Date	Version	Comment
2019-09-19	1.0	Base document