

# uFR e-Fiscalisation manual

## v1.4

# Table of contents

<b>About</b>	<b>4</b>
<b>Usage</b>	<b>4</b>
Reader Open examples	5
Android (internal NFC)	6
NexGo (contact card side slot)	7
Sunmi (PSAM slot)	8
uFR Online (UDP)	9
uFR Series reader via OTG	9
<b>Implementation</b>	<b>10</b>
Include uFCoder in Android Studio project	10
Build Gradle configuration	10
<b>Android example walkthrough</b>	<b>11</b>
Reader open	11
Get token	12
Custom APDU commands	13
Custom APDU command - Extended APDU	14
<b>uFCoder library API example - get token minimal snippet</b>	<b>17</b>
<b>uFCoder library API example - send APDU minimal snippet</b>	<b>18</b>
<b>uFCoder library HTTP service example</b>	<b>19</b>
Disclaimer	19
Usage	20
HTTP example - Get token	20
HTTP example - APDU commands	21
Starting service from another application	22
<b>References</b>	<b>22</b>
Google Play	22
Youtube	22
SDK examples	23
Documentation	23



## Revision history

24

## About

As of version **v5.0.61** of uFCoder library, support for e-fiscalization has been introduced. Communication with **Nexgo** and **Sunmi** devices is supported by utilizing the **ReaderOpenEx()** function from our API. Developers who are working on implementation of authorization via smart cards and Generic Identity Device Specification (GIDS) specification, can utilize uFCoder API on Android devices with NFC support, without writing additional code for interaction with internal NFC and sending APDU commands. Additionally, uFR Series hardware can be used with Android devices, it is necessary to use uFCoder API to make this possible. As such - NFC support on Android device(s) is not mandatory because of uFR Series hardware.

Sending and receiving APDU commands is also supported for e-fiscalization. These are our API functions that should be used:

- APDUHexStrTransceive
- SetISO14443\_4\_Mode (for uFR Series readers only)
- s\_block\_deselect (for uFR Series readers only)

Examples of sending/receiving APDU commands can be found in section [Custom APDU commands](#). Along with the short demonstration of fetching the token for e-fiscalisation under the [Get token](#) section.

## Usage

Advanced parameters of the **ReaderOpenEx()** function from our API are used for establishing communication with the device. Currently, our API supports: Android internal NFC along with Nexgo and Sumni, uFR Online series readers, uFR series readers. Parameters for interacting with the supported devices are as following:

- ReaderOpenEx **5 0 0 0** - Android internal NFC (Also supported on Sunmi V2 Pro)
- ReaderOpenEx **5 0 1 0** - NexGo contact card side slot
- ReaderOpenEx **5 0 4 0** - NexGo PSAM1 slot
- ReaderOpenEx **5 0 5 0** - NexGo PSAM2 slot
- ReaderOpenEx **5 0 8 0** - Sunmi PSAM slot
- ReaderOpenEx **1 ip\_address 85 0** - uFR Online UDP connection
- ReaderOpenEx **1 ip\_address 84 0** - uFR Online TCP/IP connection
- ReaderOpen - will use FTDI to try and open the device if it's connected via OTG cable (recommended for use with uFR series readers)

ReaderOpenEx function relies on the following parameters, To determine type of communication that will be initiated, ReaderOpenEx function relies on the following parameters:

- Reader type
- Port name
- Port interface
- Additional argument.

Parameter "port name" and "additional argument", when specified as **0**, depending on platform, should instead be passed as an empty string literal.

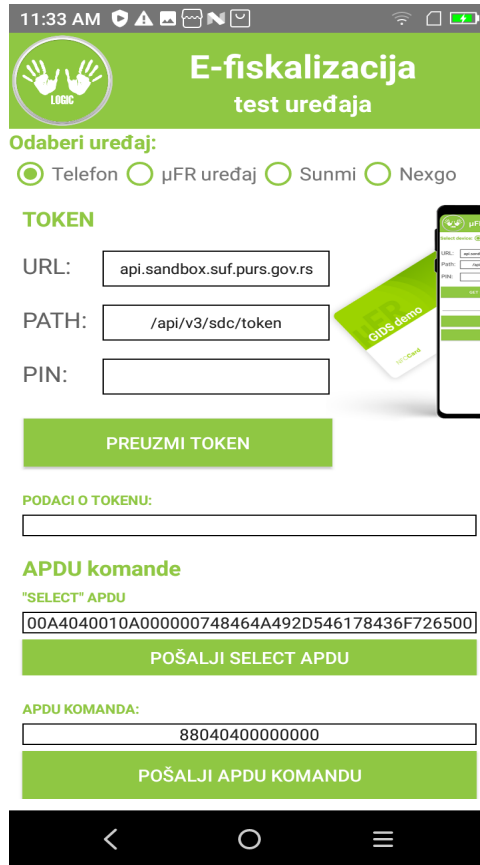
## Reader Open examples

For uFR Series hardware, on Android, using the basic *ReaderOpen()* function, while the device is connected via OTG cable, will result in almost immediate port open with FTDI.

Exception to this would be uFR Online series hardware, which contains multiple methods of communication and requires usage of *ReaderOpenEx()* method, this method is also required for Sunmi/Nexgo devices.

Examples of opening the communication with the supported devices:

## Android (internal NFC)



11:33 AM

**E-fiskalizacija**  
test uređaja

**Odaberi uređaj:**

☒ Telefon ☐ μFR uređaj ☐ Sunmi ☐ Nexgo

**TOKEN**

URL:

PATH:

PIN:

**PREUZMI TOKEN**

**PODACI O TOKENU:**

**APDU komande**

"SELECT" APDU

**POŠALJI SELECT APDU**

**APDU KOMANDA:**

**POŠALJI APDU KOMANDU**

Selecting the 'Telefon' will call the following method:

`uFCoder.ReaderOpenEx(5, "", 0, "");`

Video demonstration of using Android device with internal NFC reader: <https://youtu.be/b2QofrpRwHo>

## NexGo (contact card side slot)

**E-fiskalizacija**  
test uređaja

**Odaberi uređaj:**

☐ Telefon ☐ μFR uređaj ☐ Sunmi ☒ Nexgo

**READER OPEN** ☒ Dodatne opcije

Reader type: 5 Port name: 0

Interface: 1 Arg: 0

**TOKEN**

URL:

PATH:

PIN:

**Reader connected successfully**

**PODACI O TOKENU:**

Debug Mode

Selecting 'Nexgo' by default will have these parameters as input, clicking on 'Reader Open' with this input will be calling the ReaderOpenEx() method as:

`uFCoder.ReaderOpenEx(5, "", 8, "");`

Video demonstration of using Nexgo reader can be found here: <https://youtu.be/34rpoPHeHCA>

## Sunmi (PSAM slot)


Selecting 'Sunmi' by default will have these parameters as input, clicking on 'Reader Open' with this input will be calling the ReaderOpenEx() method as:

```
uFCoder.ReaderOpenEx(5, "", 8, "");
```

Video demonstration of using Sunmi V2 Pro reader can be found here: <https://youtu.be/Ms3h-eqWk5A>



## uFR Online (UDP)



## E-fiskalizacija test uređaja

**Odaberi uređaj:**

☐ Telefon
 ☒ μFR uređaj
 ☐ Sunmi
 ☐ Nexgo

☒ Dodatne opcije

Reader type:	1	Port name:	192.168.1.8
Interface:	U	Arg:	0

**TOKEN**

URL:   
 PATH:   
 PIN:

**PODACI O TOKU**

Reader connected successfully

**APDU komande**

For uFR Online most important parameters are 'port\_name' which is the IP address of the device on the network, and 'interface' which specifies the type of connection. This is the example of UDP connection, and by clicking on 'Reader Open' it will call ReaderOpenEx() method as:

```
uFCoder.ReaderOpenEx(1, "192.168.1.8", 85, "");
```

For more details on uFR Online series device and it's configuration, read more in 'uFR Online - Quick Start Guide' document, that can be found here: <https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git>  
 Video on how to open uFR Online can be found here: <https://youtu.be/L75l2ReA61c>

## uFR Series reader via OTG

Simplest method for interacting with a uFR series reader via OTG cable would be to call the ReaderOpen() method. This method is called just by clicking on the 'Reader Open' button with 'Dodatne opcije' checkbox unchecked and 'μFR uređaj' selected. By doing that, the following method is called:

```
uFCoder.ReaderOpen()
```

This method requires no additional parameters, and relies on FTDI API to establish communication with the device. Video on how to open uFR via OTG can be found here: <https://youtu.be/28af6oulv1c>

## Implementation

This section will describe basic implementation of uFCoder library in Android studio and showcase function calls necessary for getting the token.

### Include uFCoder in Android Studio project

Follow the instructions below to include the library in your Android project.

1. Create new or open existing Android project.
2. Open project folder in File Explorer.
3. Navigate to "<MyProject>/app" directory.
4. Download "[ufr-lib](#)" and rename it as "libs" inside the "/app" directory so the final path is "<MyProject/app/libs>"

### Build Gradle configuration

Follow the instructions below to configure the build gradle for using the library.

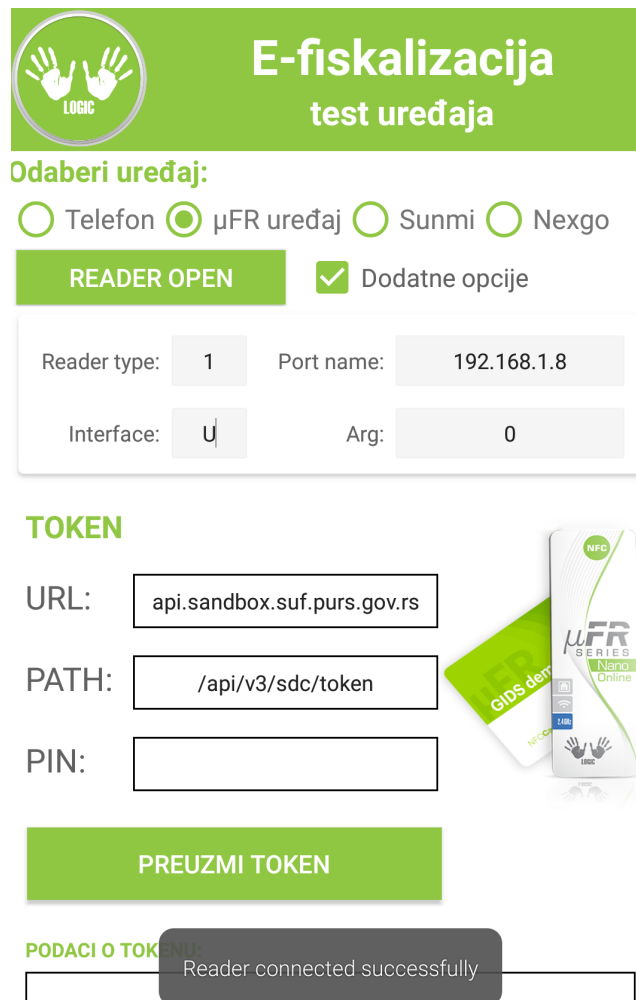
1. Open build.gradle file (the one under 'app')
2. Add **implementation files** under "*dependencies*":
  - a. Add **('libs/android/uFCoder.aar')** - contains support only for Android internal NFC;
  - b. Add **('libs/android\_sunmi/uFCoder.aar')** - which supports Sunmi devices & Android internal NFC;
  - c. Add **('libs/android\_nexgo/uFCoder.aar')** - which supports Nexgo devices & Android internal NFC;
3. Click on the 'sync now' button to sync projects with Gradle files.

## Android example walkthrough

First step, in almost every example in our SDK, would be utilizing ReaderOpen (or ReaderOpenEx which requires providing advanced parameters) to open communication with the device of your choice. For the purpose of this demo, we will use our [uFR Online Series reader](#).

### Reader open

- Select 'μFR uređaj' and, depending on device, select 'Dodatne opcije' for providing advanced parameters and using the *ReaderOpenEx()*, instead of the *ReaderOpen()* function by default.
- Fill in the parameters according to the device type. For the purpose of this demo, we'll start by using the uFR Online series reader. Parameters for this use case would be as following:



**E-fiskalizacija test uređaja**

**Odaberi uređaj:**

☐ Telefon ☒ μFR uređaj ☐ Sunmi ☐ Nexgo

**READER OPEN** ☒ Dodatne opcije

Reader type: 1 Port name: 192.168.1.8

Interface: U Arg: 0

**TOKEN**

URL: api.sandbox.suf.purs.gov.rs

PATH: /api/v3/sdc/token

PIN:

**PREUZMI TOKEN**

**PODACI O TOKENU:**

Reader connected successfully

### APDU komande

Image 1. Successful 'Reader Open' with advanced parameters

Code equivalent of calling 'Reader Open' with these parameters would be the following line in Android:  
`status = uFCoder.ReaderOpenEx(1, '192.168.1.8', 85, "");`

At the bottom, a message with the results of our action will appear. Successful 'Reader Open' would display the message as it is on the previous image, otherwise a message with error code will be shown..

## Get token

- After a successful 'Reader Open' we can use the rest of the software properly.
- Next option would be to get a token via the 'Preuzmi token' button. This requires valid input of parameters 'URL', 'PATH', and finally 'PIN' which varies depending on your card. Successful result of this action will appear as following:

### TOKEN

URL:

PATH:

PIN:



### PREUZMI TOKEN

#### PODACI O TOKENU:

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Fri, 24 Dec 2021 14:06:51 GMT
Connection: close
Content-Length: 83

{"token":"81247557-da38-46b4-a621-a8c9cae63356","expiresAt":"2021-12-24 15:44:46Z"}
```

### APDU komande

Image 2. Token results.

Code necessary for execution of 'Preuzmi Token' is equivalent to the following snippet in Android:

```
byte null_cert[] = new byte[10];
status[0] = uFCoder.DL_TLS_SetClientCertificate(2, null_cert);
if (status[0] == 0) {
    int[] ret_status = new int[1];
    String resStr = uFCoder.DL_TLS_Request(ret_status, URL, PATH, 443, PIN);
}
```

Function *DL\_TLS\_SetClientCertificate()* - is used for user certificate authentication with our TLS 1.2 HTTPS client. For more details on the *DL\_TLS\_SetClientCertificate()* method, refer to 'uFR Series NFC reader API':

<https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git>

Function *DL\_TLS\_Request* is used to transceive HTTPS GET request over TLS 1.2 secure connection, implementing TLS/SSL user certificate authentication on server request. Request doesn't contain HTTP body and use minimal of the HTTP headers.

**Important: After the token was received (*DL\_TLS\_Request()* method finished), the library does not use HTTPS any further. The result, token, if successfully received - can be used at the user's discretion. Any other operation with the token requiring HTTP/HTTPS needs to be implemented separately in the user's software, uFCoder library does not provide any other methods in our API for this purpose.**

For more details on the *DL\_TLS\_Request()* method and our TLS support, refer to 'uFR Series NFC reader API':

<https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git>

## Custom APDU commands

- Next option is the sending of custom APDU commands, based on sending initially the separate 'Select APDU' via 'Pošalji Select APDU' button, and then APDU commands based on your input via 'Pošalji APDU komandu' button. Card/tag must be in the reader's RF field before sending this command.
- By default, the 'Select' APDU command used in this software has the following value: '00A4040010A000000748464A492D546178436F726500'. Successful execution yields the following result:

## APDU komande

## "SELECT" APDU

00A4040010A000000748464A492D546178436F726500

POŠALJI SELECT APDU

## APDU KOMANDA:

88040400000000

POŠALJI APDU KOMANDU

## APDU ODGOVOR:

Card detected... Sending "SELECT" command:  
[C]: 00A4040010A000000748464A492D546178436F726500  
status: [0x00 (0)] UFR\_OK  
[R]: 9000

O APLIKACIJI

LICENCA

Image 3. 'Select APDU' result

Custom APDU command - Extended APDU

- **Important: If the 'Select' command was executed successfully, you should not remove card/tag from the devices RF field. If you do, communication will be lost and the card/tag will need to be 'Selected' again before sending APDU commands.**
- Finally, we have a demo of an extended APDU command, with the default value of '88040400000000' in the input field. This command, if successful, will yield a large amount of data in one response. The successful execution should look like this:

**APDU KOMANDA:**

88040400000000

**POŠALJI APDU KOMANDU**

**APDU ODGOVOR:**

Card detected... Sending "SELECT" command:  
[C]: 00A4040010A000000748464A492D546178436F726500  
status: [0x00 (0)] UFR\_OK  
[R]: 9000  
  
status: [0x00 (0)] UFR\_OK  
[R]:  
308205CE308204B6A003020102020E6220971F98E6F28900000  
00004A4300D06092A864886F70D01010D05003049310B30090  
6035504061302525331173015060355040A130E506F7265736B  
61205570726176613121301F0603550403131853616E64626F7  
8205355462049737375696E672043412031301E170D32313039  
32313038323030395A170D3234303932313038333030395A30  
81C3310B3009060355040613025253311B301906035504080C  
12D09FD0BED0B6D0B0D180D0B5D0B2D0B0D18631123010060  
35504071309506F7A617265766163311730150603550409130E

Image 4. Extended APDU command output





```
F713992B4988557F6137602BE9B846DBA70ACEBA1D2248358|
DA318AA522FAAD39E1CC4AEACF09638EB193B956A3D90555C
8FC38B8BA7A816673C9ED379E893B7D0AA4F60D164915F590
7AE201C2078C9D6039F28754EAC7C84FEC187A176619EBA57
0B855F86989D3EDFA85A9915DE2C29C6944CAD17068156E04
CEBDF0B07B2BB8A849680850F1E50C8FFA683C177B29B16CD
741AF0CC37FBBFE53AB048E405390203010001A38202373082
0233300F0603551D130101FF04053003020100300E0603551D0
F0101FF04040302049030170603551D250410300E060C2B0601
040183862005080303081C50603551D200481BD3081BA3081
B7060C2B06010401838620050804023081A6303206082B0601
05050702011626687474703A2F2F706B692E73616E64626F782
E7375662E707572732E676F762E72732F706B69307006082B06
01050507020230641E6200730061006E00640062006F0078005
50073006500200070004B00490053007200620069006A006100
530061006E00640062006F00780020006900730073007500610
06E006300650050006F006C00690063007900200063006C0061
0073007300323032060B2B06010401838620050805042368747
470733A2F2F6170692E73616E64626F782E7375662E70757273
2E676F762E72733018060B2B060104018386200508060409313
131333835343434301D0603551D0E04160414D3499EE620209
0D32EED5E64EF4D393DFC8C0577301F0603551D23041830168
014332482ABBB7BF155698118B8DE971B0BDFE4A1E1304A060
3551D1F04433041303FA03DA03B8639687474703A2F2F706B6
92E73616E64626F782E7375662E707572732E676F762E72732F
706B692F5355464943413153616E64626F782E63726C3055060
82B0601050507010104493047304506082B0601050507300286
39687474703A2F2F706B692E73616E64626F782E7375662E707
572732E676F762E72732F706B692F5355464943413153616E64
626F782E636572300D06092A864886F70D01010D0500038201
01008B12C9CCF9A58066FA2C07419800949D9CDE269B5F0CF
CD23EAD93DF796F96BDAC673EA9DD7C9523A4DC3E556BCF6
BCE9CD4021D3368340A5BB9EE8C5072FE30679F7A66052EAC
CA05071FD622A35B62F6D654F4BC9CCAC32F611E905911B0E
61B50BDF9E4E0EE23CA4B71E3B6CC74279F5C11BE7FF073DB
BB1E7DD9B3A84E8DF5976E7006B6A26B8B46905CA3BA0068A
B4F46EEF5A7130AE4B7AADAD56687FB3BB6F1FB168FB03867
B2F61AE5D3264D48704AC748B4E0FC551DFC94B1B3B2318A5
5889D7876CC6137D5380B50CACDA093557A8D2CFDBCDBAB9
8EA2F341108EB359B44A9B40D376F93F5035A4B2F829DC78F3
C0C1B5367083FEB35E73484D03F9000
Received: 1492 bytes
```

O APLIKACIJI

LICENCA

Image 5. Completed extended APDU command output



Transmission of both 'Select' command and the 'Extended APDU' commands demonstrated can be done in the following manner in Android:

```
ret_status[0] = uFCoder.APDUHexStrTransceive(send_cmd_string, resp_string);
```

By using the *APDUHexStrTransceive* function, you can send C-APDU in the *c\_string* (zero terminated) format, containing pairs of the hexadecimal digits.

This software is available on the Play store:

[https://play.google.com/store/apps/details?id=com.dlogic.e\\_fiskaltest](https://play.google.com/store/apps/details?id=com.dlogic.e_fiskaltest)

## uFCoder library API example - get token minimal snippet

```
package com.dlogic.example;

import com.dlogic.uFCoder;

public class MainActivity extends AppCompatActivity {

    static {
        System.loadLibrary("uFCoder"); //Load uFCoder library
    }

    int status;
    uFCoder uFCoder; //Create uFCoder class instance

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        uFCoder = new uFCoder(getApplicationContext(), this);

        status = uFCoder.ReaderOpen();
        // if you wish to use the internal Android NFC reader and/or use other supported devices, implement
        // ReaderOpenEx with parameters you need. e.g Android internal NFC reader parameters
        // status = uFCoder.ReaderOpenEx(5, "", 0, "");
        if(status == 0x00) // if the device open was successful
        {
            byte null_cert[] = new byte[10];
            status = uFCoder.DL_TLS_SetClientCertificate(2, null_cert);
        }
    }
}
```

```

        if (status == 0) {
            String tokenResult = uFCoder.DL_TLS_Request(status, URL, PATH, 443, PIN);
            if (status == 0) {
                // if the status was a valid one, handle result here
            } else {
                // otherwise handle errors in this segment
            }
        }
    }
}

```

Full code of this snippet, as an Android example, can be found in our SDK git repository:

[https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-efiscalisation\\_test.git](https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-efiscalisation_test.git)

## uFCoder library API example - send APDU minimal snippet

```

package com.dlogic.example;

import com.dlogic.uFCoder;

public class MainActivity extends AppCompatActivity {

    static {
        System.loadLibrary("uFCoder"); //Load uFCoder library
    }

    int status;
    uFCoder uFCoder; //Create uFCoder class instance

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        uFCoder = new uFCoder(getApplicationContext(), this);

        int status = 0x01;
        status = uFCoder.ReaderOpen();
        // if you wish to use the internal Android NFC reader and/or use other supported devices, implement
        // ReaderOpenEx with parameters you need. e.g Android internal NFC reader parameters
    }
}

```

```

// status = uFCoder.ReaderOpenEx(5, "", 0, "");
if(status == 0x00) // if the device open was successful
{
    status = uFCoder.SetISO14443_4_Mode(); // this function is mandatory for uFR devices, omit if
                                           // you're not using uFR Series reader

    if (status == 0) {
        resp = uFCoder.APDUHexStrTransceive(ret_status, ApduCmd_Str);
        //alternatively you can use uFCoder.APDUPlainTransceive() method for sending an APDU
        // command as a byte array instead of a hex string
        if (status == 0) {
            // if the status was a valid one, handle result 'resp' here
        } else {
            // otherwise handle errors in this segment
        }
        uFCoder.s_block_deselect((byte) 100); // mandatory so that uFR reader can resume polling
                                              // use this method when you're finished with
                                              // sending/receiving of APDU commands.
    }
}
}
}
}

```

For more details on our API and function calls, refer to 'uFR Series NFC reader API':

<https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git>

## uFCoder library HTTP service example

This feature currently only supports Nexgo and Sunmi devices and is still in development.

This feature does **not** support Android Internal NFC usage.

Source code can be found here: [https://www.d-logic.com/code/nfc-rfid-reader-sdk/ufr-lib\\_http\\_service](https://www.d-logic.com/code/nfc-rfid-reader-sdk/ufr-lib_http_service)

For more details contact us at: <mailto:support@d-logic.com>

## Disclaimer

Using uFCoder HTTP service apk for commercial purposes requires of user to compile their own, **unique, package name** of the service apk (by default set as "com.dlogic.ufrwebstarter"). Simply change the package name & application ID provided in the example source code, and recompile under a different package name. Refer to "uFCoder HTTP Service" document [here](#) for more details.

## Usage

Install *uFRHttpService.apk* and start one time. Reboot device. Service will be started automatically on device boot.

HTTP service endpoint: <http://ip-address:1234> (by default).

Available commands (send plain body text):

**ReaderOpenEx** 5 0 1 0 - NexGo contact card side slot

**ReaderOpenEx** 5 0 4 0 - NexGo PSAM1 slot

**ReaderOpenEx** 5 0 5 0 - NexGo PSAM2 slot

**ReaderOpenEx** 5 0 8 0 - Sunmi PSAM slot

**DL\_TLS-Token** host path port pin

(e.g "DL\_TLS-Token api.sandbox.suf.purs.gov.rs /api/v3/sdc/token 443 8440")

**APDU** "hexstring"

(e.g) APDU 00A4040009A0000003974254465900

**Restart** - Restarts service

## HTTP example - Get token

For this demonstration [Talend API Tester](#) was used.

1. Simply send "ReaderOpenEx" with necessary parameters via HTTP as plain text:

The screenshot shows the Talend API Tester interface. The 'METHOD' is set to 'POST'. The 'SCHEME // HOST [ ":" PORT ] [ PATH [ "?" QUERY ] ]' is set to 'http://192.168.1.210:1234'. Below this, there is a section for 'QUERY PARAMETERS' with a right-pointing arrow. On the left, there is a 'HEADERS' section with a right-pointing arrow and a 'Form' dropdown. Below the headers, there are buttons for '+ Add header' and 'Add authorization'. On the right, there is a 'BODY' section with a right-pointing arrow. Below the body, there is a text input field containing '1 ReaderOpenEx 5 0 1 0'.

Response should be in JSON format:

### Response

200 OK

#### HEADERS

Content-length: 44 bytes  
Content-type: application/json

▶ COMPLETE REQUEST HEADERS

pretty

#### BODY

```
{
  status: "[0x00 (0)] UFR_OK",
  response: ""
}
```

lines nums

2. Send "DL\_TLS\_TOKEN" with necessary parameters:  
Parameters in order are: URL, PATH, PIN.

METHOD: POST SCHEME://HOST [":PORT"] [PATH ["?" QUERY]]  
http://192.168.1.210:1234

▶ QUERY PARAMETERS

HEADERS: Form

+ Add header Add authorization

BODY

```
1 DL_TLS_Token api.sandbox.suf.purs.gov.rs /api/v3/sdc/token 443 8440
```

## HTTP example - APDU commands

Sending APDU commands via uFCoder HTTP service is executed in the following manner:

1. Simply send "APDU" with the APDU command as a string following the keyword in plain text format.  
For example:

METHOD: POST SCHEME://HOST [":PORT"] [PATH ["?" QUERY]]  
http://192.168.1.210:1234

▶ QUERY PARAMETERS

HEADERS: Form

+ Add header Add authorization

BODY

```
1 APDU 00A4040010A000000748464A492D546178436F726500
```

2. Response is received as JSON:

### Response

200 OK

#### HEADERS <sup>?</sup>

Content-length: 48 bytes  
Content-type: application/json

► COMPLETE REQUEST HEADERS

pretty ▼

#### BODY <sup>?</sup>

```
{
  status: "[0x00 (0)] UFR_OK",
  response: "9000"
}
```

## Starting service from another application

To start uFR HTTP service from another application you need to use **com.dlogic.ufrwebstarter.uFRWebService** intent.

Example code:

```
Intent i = new Intent();
i.setComponent(new
ComponentName("com.dlogic.ufrwebstarter",com.dlogic.ufrwebstarter.uFRWebService"));
getApplicationContext().startService(i);
```

## References

### Google Play

uFR e-Fiscalisation app: [https://play.google.com/store/apps/details?id=com.dlogic.e\\_fiskaltest](https://play.google.com/store/apps/details?id=com.dlogic.e_fiskaltest)

uFR GIDS demo: <https://play.google.com/store/apps/details?id=com.dlogic.ufrgidsdemo>

### Youtube

D-Logic NFC SDK channel: <https://www.youtube.com/channel/UCNYYw0iUoua3bMTJbSuaWQA>

uFR e-Fiscalisation demo with Nexgo reader: <https://youtu.be/34rpoPHeHCA>

uFR e-Fiscalisation demo with Sunmi V2 reader: <https://youtu.be/Ms3h-eqWk5A>

uFR GIDS demo with uFR reader (WiFi connection): <https://youtu.be/L75l2ReA61c>

uFR GIDS demo with uFR reader (USB connection): <https://youtu.be/28af6oulv1c>

uFR GIDS demo with Android internal NFC: <https://youtu.be/b2QofrpRwHo>

## SDK examples

uFR e-Fiscalisation - Android source code:

[https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-efiscalisation\\_test.git](https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-efiscalisation_test.git)

uFR GIDS demo - Android source code:

<https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-android-gids.git>

uFR GIDS demo - iOS source code:

<https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-examples-ios-gids.git>

## Documentation

All documents related to our SDK can be found here:

<https://www.d-logic.net/code/nfc-rfid-reader-sdk/ufr-doc.git>

Important documents related to this manual:

- uFR Series NFC reader API
- uFR Online Quick Start Guide
- using uFR Android library
- using uFR library in XCode

## Revision history

Date	Version	Comment
2022-12-02	1.4	Updated <a href="#">Get token</a> section. Important explanation of DL_TLS_Request() added.
2022-02-24	1.3	Updated sections: <a href="#">Build Gradle configuration</a> ; <a href="#">HTTP example - Get token</a> ; <a href="#">About</a> ; Added section: <a href="#">Disclaimer</a> for uFCoder HTTP service.
2022-02-22	1.2	uFR HTTP service starting from another application example.
2022-02-01	1.1	Additional details for <a href="#">Custom APDU commands</a> section.
2019-10-31	1.0	Base document