



About

Usage

- nfcpy
- pyscard

Operation

- End state
- Points of failure

Setup

- ACS ACR122
- Identiv uTrust 3700F
- Sony RC-S380
- Windows
- macOS and Linux

Examples

- Identiv uTrust 3700F & app_demo_dp_tlogger
- Sony RC-S380 & app_demo_dp_blinky

1. About

The *NHS31xx one-time NFC program downloader* is the default firmware image flashed by NXP in all NHS31xx ICs during production. This python3 script is one example how the host side can be implemented to communicate with that firmware.

Using this script, you can download over the NFC interface your final firmware image. This can only be done once: upon successful completion, the default firmware image will be at least partially overwritten with your firmware image, and your firmware image will be executed after a reset.

2. Usage

The tag to program (containing the NHS31xx IC) must be battery powered and never been flashed before (i.e. the default firmware image *NHS31xx one-time NFC program downloader* is still active).

```
1  nfcloader.py [-h] [-d {nfcpy,pyscard}] [-t TAGREADER] file
2
3  positional arguments:
4      file                The binary file to be loaded into the target
5
6  optional arguments:
7      -h, --help          show this help message and exit
8      -d {nfcpy,pyscard}, --driver {nfcpy,pyscard}
9                          Module to be used for tag handling (default: pyscard)
10     -t TAGREADER, --tagreader TAGREADER
11                          tag reader to connect to (default: None)
```

When no module is explicitly selected, the *nfcpy* module is used.

2.1 nfcpy

The *nfcpy* module will try to use the first USB NFC tag reader it finds. This is equal to providing `-t usb` at the command line.

It is possible to discriminate between different models by supplying vendor ID (vid) and a product ID (pid). The format for a correct port name then becomes `usb:<vid><pid>`.

Examples:

- to target the Sony RC-S380 reader, provide `-t usb:054c:06c1` at the command line
- to target the ACS ACR122 reader, provide `-t usb:072F:2200` at the command line

(Aside: the vendor ID for NXP products is `1fc9`.)

2.2 pycard

The *pycard* module will try to use the first tag it can find. This is usually the NFC reader built-in in your PC.

A list of readers accessible through *pycard* can be obtained by running

```
1 python3 -c "from smartcard.System import readers; print(readers())"
```

With as example output:

```
1 ['Broadcom Corp Contacted SmartCard 0', 'Broadcom Corp Contactless SmartCard 0',  
  'Identiv CLOUD 3700 F Contactless Reader 0']
```

To target the Identiv reader, you can provide a string that matches with that reader alone. For example, by providing `-t cloud` at the command line.

3. Operation

Once the python3 script is launched, access to a tag reader is tried. When that succeeds, the script checks if an NFC tag is present. When that fails, it waits for an NFC tag to be touched.

When a tag is within reach of the NFC field of the tag reader, the NDEF data is read out. When an NDEF message is found, containing a MIME record with version information as payload, *and* the version numbers match, the download process starts.

The provided binary file is split up in smaller portions ‘chunked’ and fed one by one to the NHS31xx IC via the NFC shared memory. Each chunk is wrapped in an NDEF message and sent to the tag. A new chunk is sent when the tag has acknowledged the data by writing a response in the NFC shared memory.

The last chunk is sent with a CRC calculated over the entire file. This is used to check if the full file is received without errors. The last action of the *NHS31xx one-time NFC program downloader* running on the NHS31xx IC is to shut down the IC. After any reset the newly download firmware application gets control.

3.1 End state

The last response contains the overall error code, indicating whether the flash is successfully re-programmed. The IC remains awake for a few 100 milliseconds after providing the last response, to ensure the host has enough time to read the NDEF message. After that, the IC enters the power-off mode.

3.2 Points of failure

During the download process, it is possible that

- a chunk of data is not correctly written in the NFC shared memory.
- a response is not written by the NHS31xx.
- a response contains an error code indicating a handling problem by the NHS31xx IC.
- a response is not correctly read out by the tag reader.

This can occur when

- The tag’s position is not optimal.
- The tag’s position is not kept stable.
- The file size is too big.

If anything goes wrong during download, the process will be aborted. The script will write a corresponding error message and exit. The host is to check the error and make adjustments to the setup. After waiting 5 seconds, the download can be retried.

One critical point of failure is the moment when the *NHS31xx one-time NFC program downloader* gets swapped out for the new firmware. This is done immediately after receiving the last chunk of data (which also includes the file CRC). The flash sector is erased, and written again. Sector 0 contains among others the interrupt vector table.

- After erasing sector 0, the downloader firmware can no longer run after a reset.
- Before sector 0 is written again, the new firmware is not yet able to run after a reset.

A too severe voltage drop or a reset pulse will reset the IC. There is no valid firmware image, so the target will hang and a retry is not possible. The only way to overcome this issue is to program the IC wired via the HW pins (SWD pins).

Erasing a sector is more taxing to the supply than programming a sector. Programming or erasing a flash sector necessitates the presence of a battery or an external supply, to guarantee the load can be provided during the erase operation. When running passively, using the NFC harvested energy, this guarantee is not possible. The tag must be battery or externally powered.

Last: the write operation in flash is not read back to verify the correct contents have been written. Testing from our part of the on-chip flash driver has not raised a concern.

4. Setup

4.1 ACS ACR122

- for *nfcpy*: The PlugAndPlay driver automatically selected and installed by Win10 is sufficient. If not, use [Zadig](#) (Win10 only) to ensure the correct USB driver - *WinUSB* - is used with the tag reader.
- for *pyscard*: download the *ACS Unified PC/SC driver* for all platforms from the [ACR support site](#).

Only one option - *nfcpy* or *pyscard* - can be used at any given time. Use Zadig to change the associated driver to *WinUSB*, use the ACR installer to change it back to *ACR122U*.

4.2 Identiv uTrust 3700F

For use with the *pyscard* module only.

The PlugAndPlay driver automatically selected and installed by Win10 is sufficient. Alternatively, you can download driver installers for all platforms from the [Identiv support site](#).

4.3 Sony RC-S380

- for *nfcpy*: use [Zadig](#) (Win10 only) to ensure the correct USB driver - *WinUSB* - is used with the tag reader.
- for *pyscard*: install the [NFC Port Software](#) from Sony (Win10 only).

Only one option - *nfcpy* or *pyscard* - can be used at any given time. Use Zadig to change the associated driver to *WinUSB*, use the Sony installer to change it back to *sonynfcport100c*.

4.4 Windows

Install Python 3.5 or later, make sure `python.exe` and `pip.exe` are in your PATH environment variable.

- When using *nfcpy*, follow the installation instructions on [nfcpy.readthedocs.io](#).

In short:

- Install the [libusb](#) tool
- Install the *nfcpy* module: `pip3 install nfcpy`

When experiencing problems with the installation, you can try `pip3 install --upgrade --force-reinstall nfcpy`

- *pyscard* is hosted on sourceforge.net. It too requires additional tools to be installed.
In short:
 - Install the [SWIG \(Simplified Wrapper and Interface Generator\)](#) tool and add it to your PATH environment variable.
 - Install [Microsoft Visual C++ Build Tools 2015](#)
Note: this is already installed together with Visual Studio.
 - Install the *pyscard* module: `pip3 install pyscard`
When experiencing problems with the installation, you can try `pip3 install --upgrade --force-reinstall pyscard`

This script is successfully tested on a Win10 PC running Python 3.7.4 32 bit, using these combinations ¹:

Running Python 3.7.4 32 bit on Win10 64 bit	ACS ACR122	Identiv uTrust 3700F	Sony RC-S380
<i>nfcpy</i> i.c.w. WinUSB driver ²	yes		yes
<i>pyscard</i> i.c.w. product-specific driver	yes	yes	yes

¹ This is not an exhaustive list.

² A list of supported devices for use with *nfcpy* is available on nfcpy.readthedocs.io

4.5 macOS and Linux

Full support for running this script on other platforms than Win10 is still pending.

This script has been reported to work on macOS and Linux (Ubuntu).

- On macOS 10.15 Catalina
using *nfcpy*, after installing *nfcpy* and libusb.
- On Ubuntu 19.10 Eoan Ermine
using *nfcpy*, after installing *nfcpy*, *ndeflib* and libusb.

Add `import ndef.message` to `site-packages/nfc/tag/__init__.py`
Follow the instructions given after executing `nfc/__main__.py`

5. Examples

5.1 Identiv uTrust 3700F & app_demo_dp_tlogger

Using the Identiv uTrust 3700F tag reader to flash the NHS3100 temperature logger demo from SDK 12.3.

Launching the script

```
1 C:\sdk\tools\ncfcloder\python>python nfcloader.py -d pyscard -t identiv
C:\sdk\sw\nss\app_demo_dp_tlogger\Release\app_demo_dp_tlogger.bin
```

produces the following output:

```
1 NHS31xx one-time NFC program downloader
2 Searching for a tag
3 TAG connected
4 Firmware version 1.2
5 Tag present, application version OK
6 [ 0.00] Sending 486 bytes of data ...
7     ... data sent. Waiting for response ...
8     ... response received.
9 [ 0.79] Sending 486 bytes of data ...
10    ... data sent. Waiting for response ...
```

```

11     ... response received.
12 [ 1.58] Sending 486 bytes of data ...
13     ... data sent. Waiting for response ...
14     ... response received.
15 .
16 .
17 .
18 [28.56] Sending 486 bytes of data ...
19     ... data sent. Waiting for response ...
20     ... response received.
21 [29.35] Sending 486 bytes of data ...
22     ... data sent. Waiting for response ...
23     ... response received.
24 [30.15] Sending the last 484 bytes of data ...
25     ... data sent. Waiting for response ...
26     ... response received.
27 Download successful
28 Transmitted (bytes): 18952
29 Speed (bytes/sec): 609.20

```

5.2 Sony RC-S380 & app_demo_dp_blinky

Using the Sony RC-S380 tag reader to flash the ‘Hello World’ blinky demo.

Launching the script

```

1 C:\sdk\tools\nfcloader\python>python nfcloader.py -d nfcpy
C:\sdk\sw\nss\app_demo_dp_blinky\Release\app_demo_dp_blinky.bin

```

produces the following output:

```

1 NHS31xx one-time NFC program downloader
2 Now accessing tag reader SONY RC-S380/S NFC Port-100 v1.11 at usb:001:011.
3 Searching for a tag
4 TAG connected: Type2Tag ID=04CA0E00131221
5 Firmware version 1.2
6 Tag present, application version OK
7 [ 0.00] Sending 486 bytes of data ...
8     ... data sent. Waiting for response ...
9     ... response received.
10 [ 1.26] Sending 486 bytes of data ...
11     ... data sent. Waiting for response ...
12     ... response received.
13 [ 2.47] Sending 486 bytes of data ...
14     ... data sent. Waiting for response ...
15     ... response received.
16 [ 3.69] Sending 486 bytes of data ...
17     ... data sent. Waiting for response ...
18     ... response received.
19 [ 4.93] Sending the last 336 bytes of data ...
20     ... data sent. Waiting for response ...
21 Download successful
22 Transmitted (bytes): 2280
23 Speed (bytes/sec): 378.01

```