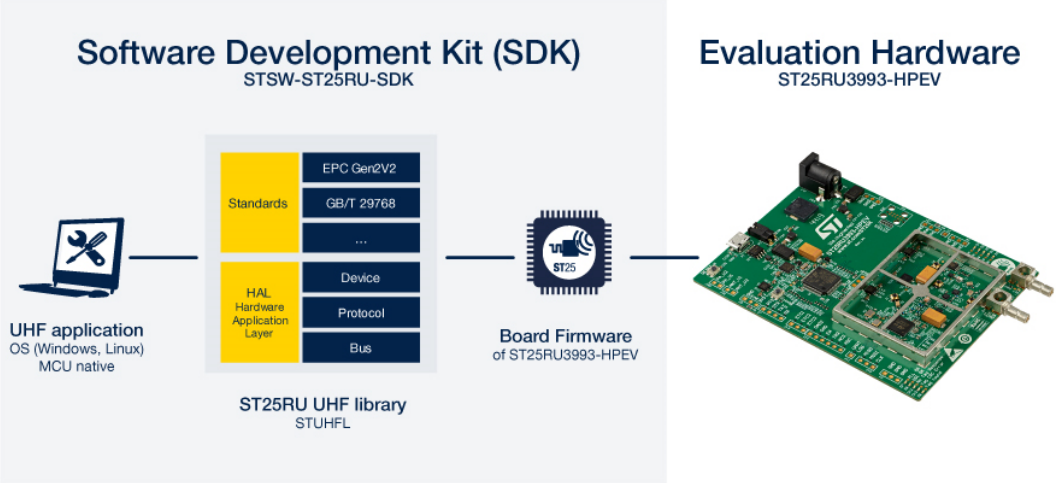
Software development kit (SDK) for ST25RU3993 based Rain RFID readers

Introduction

The STSW-ST25RU-SDK is a comprehensive middleware software stack written in ANSI C, to build RAIN® RFID enabled applications for reader devices based on ST25RU3993. This document describes the structure and organization of the SDK and provides information about the usage of the ST UHF library API.

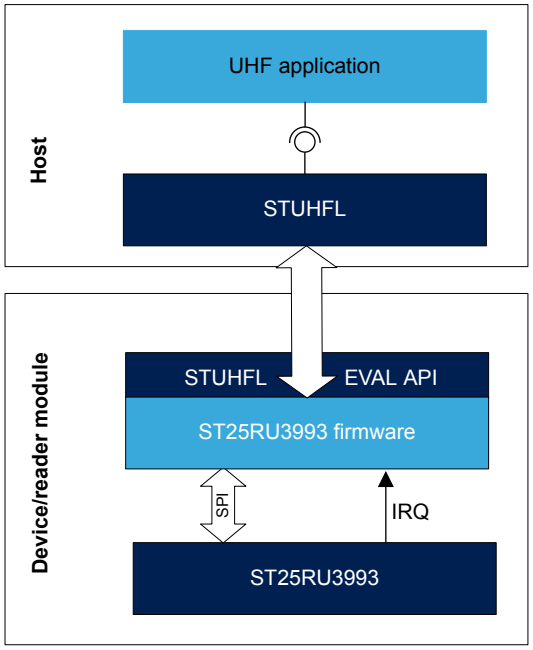
This SDK includes number of application source code examples that illustrate the usage of the ST UHF library (STUHFL) for Windows® and Linux® platforms. The complete software stack is ANSI C and POSIX complaint and enables fast and straightforward porting to other operating systems and/or toolchains.



1 System overview

The SDK design is a classic middleware software stack running on a host system. It provides a simple software API to abstract low-level communication details. As shown in the simplified overview the system comprises of two hardware entities running the dedicated software components.

Figure 1. System overview



The hardware entity at the botttom (device/reader module) runs the firmware with low-level driver implementation to operate the ST25RU3993 reader IC and various UHF RFID protocols. The firmware also includes a software interface to the STUHFL EVAL API library.

The hardware entity on top (host side) runs the STUHFL and abstracts the low-level functionality. It also provides a dedicated software API for UHF applications.

The STUHF runs on host systems with an operating system as well as on embedded systems (with OS or native), with limited resources. In such a case the system would run on the reader module only.

1.1 General information

The software described in this document supports Arm®-based device.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.2 Features

- Comprehensive middleware stack to create RAIN® RFID enabled applications for reader devices based on ST25RU3993
- Identical (mirrored) host and device library API
- Written in pure ANSI C99
- POSIX complaint
- Straightforward portability across different platforms (MCU/RTOS/OS)
- Support of main UHF standards
  - EPC UHF Gen2v2 or ISO 18000-63
  - GB/T 29768
- Application source code examples available for:
  - Windows®(10 or newer)
  - Linux® (Raspberry Pi 3B or newer)
  - Embedded (STM32L4 device)
  - SW-Wrappers for state of the art programming languages: Java, C#, Python

1.3 Hardware requirements

The main purpose of the STUHFL is to simplify the ST25RU3993 low-level hardware driver, and to provide an easy-to-use software interface. Therefore, the STUHFL operates only with the ST25RU3993-based demonstration boards listed below and their corresponding firmware:

- ST25RU3993-EVAL
- ST25RU3993-HPEV

It is also possible to port the low-level hardware driver to other devices hosting a suitable MCU to drive the firmware.

1.4 Building environment

All host side software components and application source code examples are generated using Microsoft Visual Studio 2017 (VS2017). The SDK contains a VS2017 solution hosting projects for two different platforms. One project is for Win32 platforms suitable for generating code running on Windows® 10 platforms (and newer). The second project is for Arm platforms running on Linux® devices. The Arm project provides the possibility to do remote compilation with VS2017 given the Arm device ia accessible within the same network. To be tool-chain independent make files for all mentioned projects are available as well.

Note: The STUHFL project for Arm and the corresponding example project requires a Raspberry Pi 3B(or newer) hardware for development. Refer to the corresponding project solution files to have more details about project configuration.

2 Software architecture

The STUHFL follows a classic stacked middleware software design and consists of sofware components that run on a reader device/module and on a host system as the default use case.

The complete STUHFL is ANSI-C and POSIX compliant, with the benefit of easy porting to other operating systems and platforms.

The STUHFL in its current implementation supports the common UHF RFID standards.

Application source code examples are available for Windows® and Linux®, to be used with corresponding library modules.

2.1 STUHFL API

The accessible STUHFL API allows the user to build UHF RFID (e.g. RAIN RFID) applications on ST25RU3993 reader devices. The three stacked software layers listed below represent the complete accessible STUHFL API.

- Activity
- Session
- Device

In this arrangement abstraction and complexity increases starting device layer moving towards the activity layer. The purpose of the 'Device' layer is to abstract the hardware specific functionality, such as ST25RU3993 register access. The 'Session' layer uses the 'Device' layer to implement UHF RFID protocols such as the EPC Gen2V2 or GB/T 29768. The 'Activity' layer uses the 'Device' and 'Session' layers to abstract functionality as for example automated tag inventory running in the background.

Table 1. Main layers

Layer	Description
Activity	Background functionality (e.g. tag inventory) via the build-in "Inventory Runner"
Session	Implementation of various UHF RFID standards or protocols like EPC Gen2V2 or GB/T 2968
Device	Abstraction of hardware specific functionalities like ST25RU3993 register access

2.2 STUHFL EVAL API

On the device/reader module side the STUHFL EVAL API provides a basic software interface to ST25RU3993 low-level drivers. The STUHFL builds on top of this low-level interface and derives its ST25RU3993 specific functionality. Applications where no host system is involved can directly call the low-level STUHFL EVAL API on reader/device side.

Note: Applications not using the STUHFL middleware stack can attach to this interface.

2.3 STUHFL wrappers

On top of the STUHFL host interface, several software wrappers are available for fast prototyping.

- Java/C#/Python
- ST EVAL API

Java, C# and Python are state of the art programming languages and enable software developers to implement fast and efficiently a multitude of UHF RFID application. Additionally, it is also possible to implement custom interface wrappers similar to the ST EVAL API wrapper.

Note: Software wrappers for other languages can easily be implemented when these programming languages support calling native C code.

The STUHFL EVAL API wrapper (host side) is a complete replica of the STUHFL EVAL API on the device side. This allows users to start developing firmware applications on host systems and finish their task by porting the firmware to the device/reader side. The advantage of this approach is that the host side offers debuggers, advanced code tracers and many other host system development features. These host tools speed-up and simplify code development. When the firmware code development on the host side is completed, simply "copy and paste" the firmware application code to the device/reader side. No additional source code modifications are required to run the firmware on the MCU.

Note: The STUHFL EVAL API wrapper fully emulates the STUHFL EVAL API, exactly as it is available on the device side.

2.4 Not exposed layers

Underneath the accessible STUHFL API, several other layers collaborate on the host side with their counterparts on the device side. This layer stack is taking care of the data transfer between the host and the reader device.

- Dispatcher
- Protocol
- Bus

These stacked layers depend only on their direct counterparts. The main purpose of the 'Dispatcher' layer is to bundle the STUHFL API function calls into one single module and to forward it to the protocol layer.

The 'Protocol' layer encodes the abstracted function calls into transferable TLV formatd data chunks forwarding them to the 'Bus' layer.

Finally, the 'Bus' layer sends the data chunks over the physical interface and receives them on the device/reader module side. For the data transfer in the opposite direction from device/reader side to the host side, the protocol and the dispatcher layers do their task in the reverse order.

Table 2. Not exposed layers

Layer	Description
Dispatcher	Concentrates or deconcentrates API function calls
Protocol	Encodes and decodes dispatched data into TLV formatted data chunks ready for transfer between host and device
Bus	Platform depended transfer layer handling data exchange

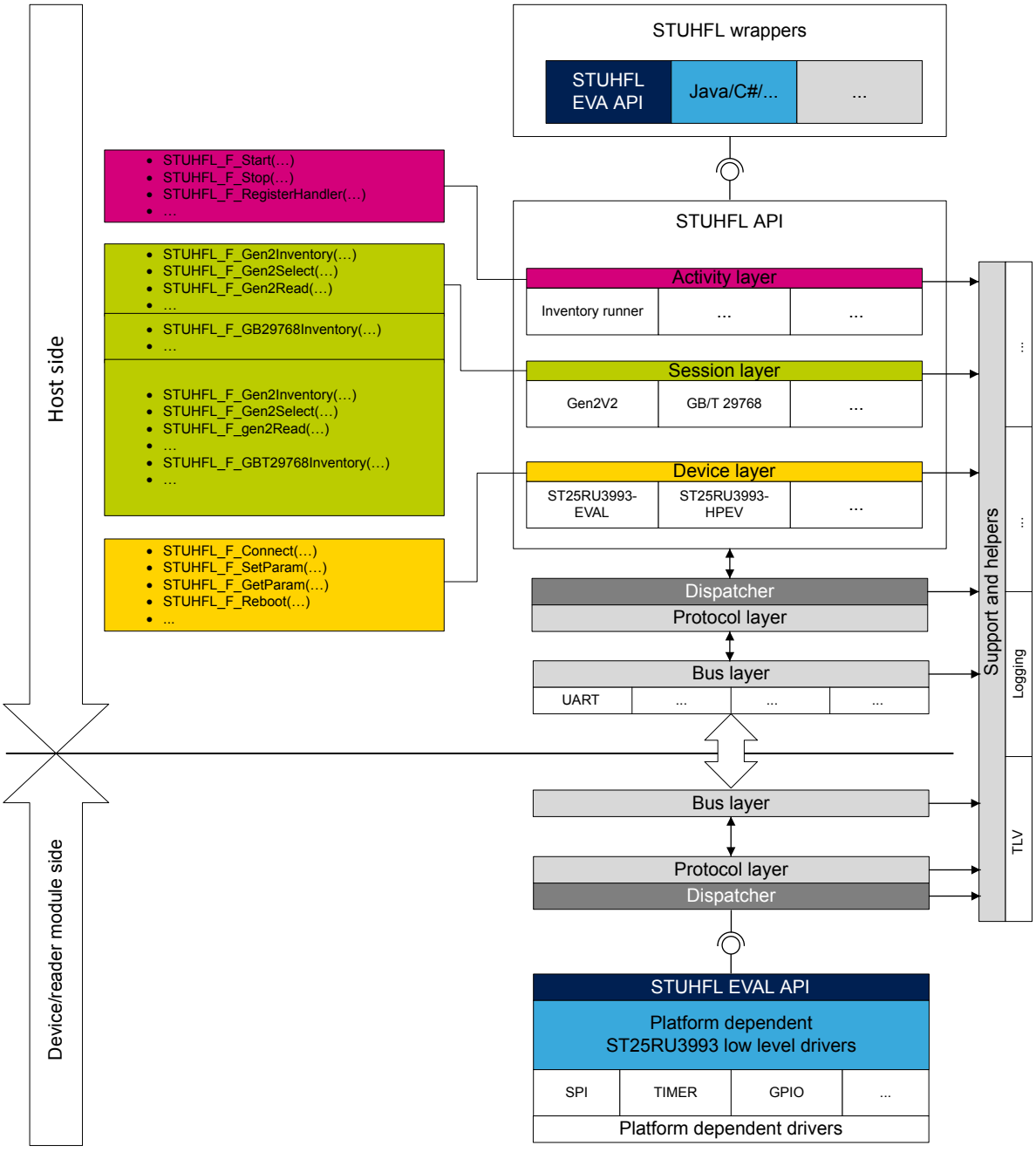
2.4.1 Support and helper

Support and helper layers provide relevant functionality and make their interface available to almost all other modules.

Table 3. Helper layer

Module	Description
TLV	Tag-Length-Value encoding and decoding support
Logging	Logging helper

Figure 2. **ST UHFL library system architecture**

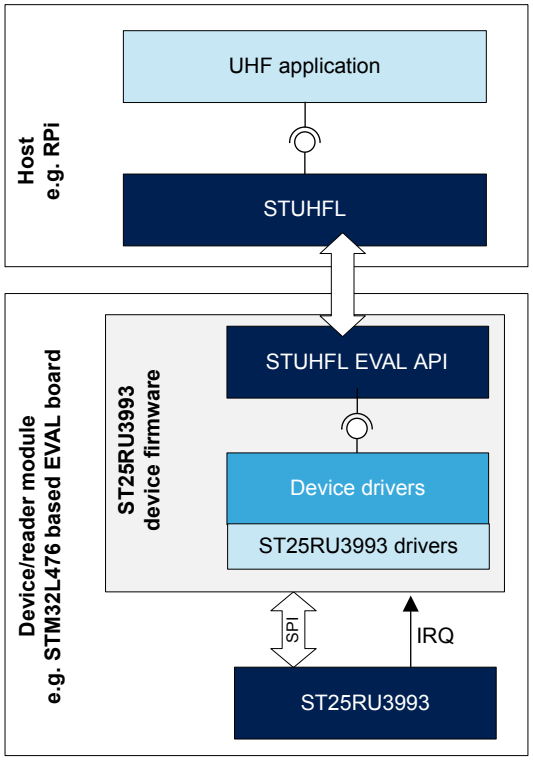


2.5 Host side usage

The functionality that is avalaible by the STUHFL API enables the host-side development of UHF RFID reader applications based on the ST25RU3993 reader IC. Host systems running Windows® or Linux® operating systems are supported by default. As the whole library is ANSI-C and POSIX compliant, it also easily builds for other POSIX compliant platforms.

The following figure shows an example of host side usage.

Figure 3. STUHFL usage overview

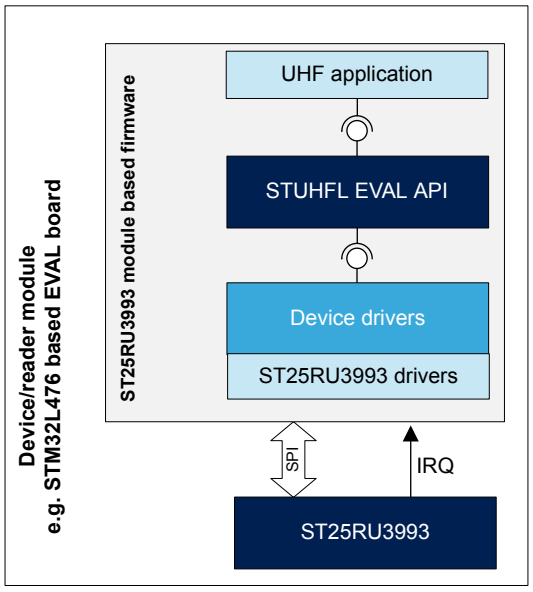


2.6 Device side usage

The available STUHFL EVAL API provides an abstract interface to the complete UHF RFID reader functionality provided by the ST25RU3993 reader IC avoiding the burden of manually handling low-level register access. Therefore, this interface is host device-independent and enables the implementation of de facto any abstract UHF RFID reader application.

The following figure shows an example of device side usage.

Figure 4. ST UHF library EVAL API device side usage



3 Source code

The complete source code (including application source code examples for Windows® and Linux®) is available for download from www.st.com.

3.1 Main SDK folders and file structure

The package comes as zip file, with the following directory structure.

Figure 5. Directory structure

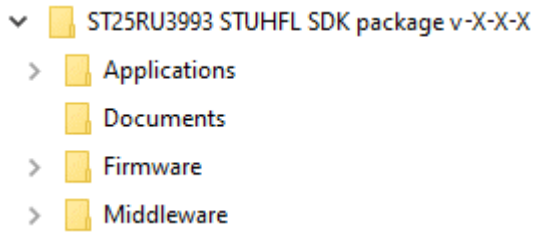


Table 4. Directory description

Directory	Description
Application	Demonstration application(s) for Windows® and Linux® showcasing the usage of the SDK
Documents	Release notes, help files and “Software Package License Agreement”
Firmware	ST25RU3993 based firmware supporting GS1 EPC Gen2v2 air interface and GB/T 29768-2013 protocols
Middleware	Native C ST-UHF-Library for host and device specific application development with ST25RU3993 Reader IC based demonstration board

3.1.1 "Applications" folder

THe "Applications" folder contains all demonstration applications for Windows® and Linux® showcasing the usage of SDK.

Figure 6. "Application" folder structure

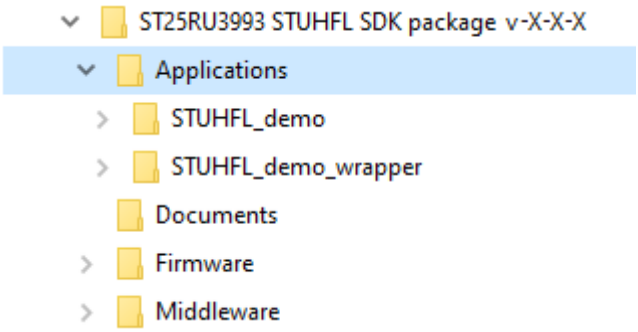


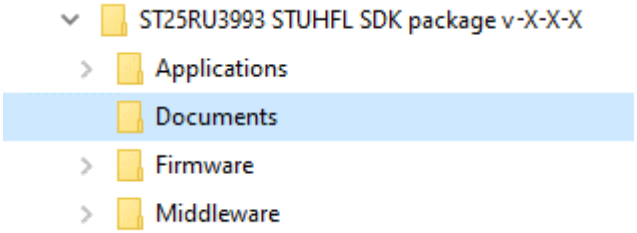
Table 5. **"Application" folder description**

Name	Description
STUHFL_demo	Demonstration application(s) for Windows® and Linux® showcasing the usage of the SDK
STUHFL_demo_wrapper	Demonstartion application demonstrating the usage of the SDK with the C#, Python or Java wrappers.

3.1.2 "Documents" folder

The "Documents" folder contains the “Release notes”, help files and “Software Package License Agreement” for all components.

Figure 7. **"Documents" folder structure**



For easier navigation the folder also contains the file README.htm with an overview of all documents and links to access them.

3.1.3 "Firmware" folder

The "Firmware" folder contains the ST25RU3993 based FW supporting GS1 EPC Gen2v2 Air Interface & GB/T 29768-2013 protocols.

Figure 8. **"Firmware" folder strucuture**

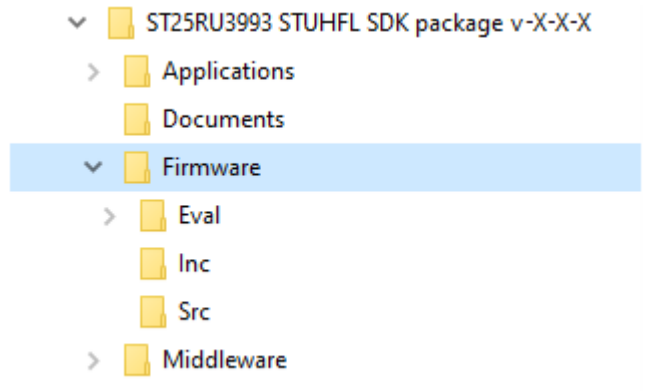


Table 6. **"Firmware" folder description**

Name	Description
EVAL	ST25RU3993-EVAL and ST25RU3993-HPEV board dependent source code files
Inc	Implementation of header files for low level drivers and protocol functionality independent of onboard MCU.
Src	Implementation of source files for low level drivers and protocol functionality independent of onboard MCU.

3.1.4 "Middleware" folder

The "Middleware" folder contains the Native C ST-UHF-Library for host and device specific application development with ST25RU3993 Reader IC based demonstration board. Also, part of the middleware folder are the wrapper implementations for C#, Python and Java based on the native C library.

Figure 9. "Middleware" folder structure

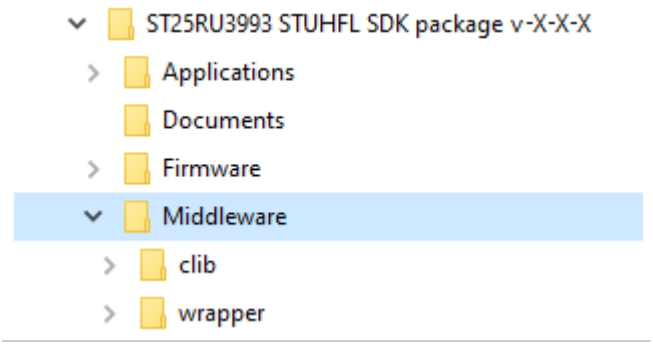


Table 7. "Middleware" folder description

Name	Description
clib	Source code of native C ST-UHF-Library for host and device specific application development with ST25RU3993 Reader IC based demonstration board ST25RU3993-EVAL and ST25RU3993-HPEV.
wrapper	Software wrappers source code for STUHFL API usage with C#, Python and Java.

3.2 Build instruction STUHFL middleware only

The VS2017 solution to build a Win32 dynamic link library or Arm® based shared object library is located at ./Middleware/clib/STUHFL.sln. For usage without Visual Studio a makefile can be found at the same location.

Note: For software development based on STUHFL a solution contains the STUHFL project itself and an example application source code to demostrate the usage of library is also available. This can be found at ./Applications/STUHFL_demo/STUHFL_demo.sln. A variant of the solution available for RPi can be found at the same folder location. These VS2017 solutions allows user to build and debug the application source code and the STUHFL middleware at once. For usage without Visual Studio a makefile can be found at the same location.

3.2.1 VS2017 project used in Windows® for STUHFL

The user must execute the following steps:

1. Open the VS2017 solution .\Middleware\clib\STUHFL.sln
2. Right-click on the STUHFL project and click rebuild

Note: This generates a Windows® DLL that can be used to communicate to ST25RU3993 Reader IC based demonstration boards on a RPi.

3.2.2 VS2017 project used in Linux® for STUHFL

The user must execute the following steps:

1. Open the VS2017 solution .\Middleware\clib\STUHFL.sln
2. Add the IP address of the RPi in the Tools/Options/Cross platform options
3. In the STUHFL (Linux®) project properties set this as the remote machine
4. Right-click on the STUHFL(Linux®) project and click rebuild

Note: This generates a Linux® complete shared object that can be used to communicate to ST25RU3993 Reader IC based demonstrantion boards on a RPi.

3.2.3 "makefile" usage for STUHFL

The user can run : `make -C ./STUHFL/Middleware/clib/makefile`

3.3 Build instruction STUHFL middleware wrapper

3.3.1 C# for STUHFL

The user must execute the following steps:

- Open .../Middleware/wrapper/cs/STUHFL_cs.sln
  - Select "Solution Configuration" (Debug/Release)
  - Select "Solution Platform" (x86/x64)
- Rebuild full solution (Build -> "Rebuild Solution")
  - STUHFL library is generated at .../Middleware/wrapper/cs/<Configuration>/<Platform>/STUHFL.dll
  - C# wrapper library is generated at .../Middleware/wrapper/cs/STUHFL_cs/bin/<Configuration>/STUHFL_cs.dll

3.3.2 Python for STUHFL

The user must execute the following steps:

- Open .../Middleware/wrapper/python/STUHFL_py.sln
  - Select "Solution Configuration" (Debug/Release)
  - Select "Solution Platform" (x86/x64)
- Rebuild full solution (Build -> "Rebuild Solution")
  - STUHFL library is generated at .../Middleware/wrapper/python/<Configuration>/<Platform>/STUHFL.dll

Note: NOTE: Select appropriate libPath variable in .../Middleware/wrapper/python/STUHFL_native.py (lines 32 to 35) in relation with selected <Configuration> & <Platform>

3.3.3 Java (Windows®) for STUHFL

The user must execute the following steps:

- Build STUHFL.dll and JNISTUHFL.dll (optional, only needed when modifications are applied)
  - Open .../Middleware/wrapper/java/jni/JNISTUHFL/JNISTUHFL.sln with Microsoft® Visual Studio Professional 2017 or newer.
  - Check "Include Directories" to match local JDK installation for the JNISTUHFL solution
    1. Open "Properties" of the JNISTUHFL solution and check "Configuration Properties - VC++ Directories - Include Directories"
    2. Edit, if needed, the path to JDK includes to match local installation and prefered Java version
- Select "Solution Configuration" (Debug/Release)
- Select "Solution Platform" (x86/x64)
- Rebuild all

Note: NOTE: The generated DLL's can be found at .../Middleware/wrapper/java/jni/JNISTUHFL/<Configuration>/<platform>/

Command line:

```
> cd .../Middleware/wrapper/java/jni/JNISTUHFL
> MSBuild.exe JNISTUHFL.sln /verbosity:m /property:Configuration=Debug /property:Platform=x64
```

- Build STUHFL.jar (optional, only needed when modifications are applied)
  - Compile the java sources from .../Middleware/wrapper/java/src and package it to a jar archive

Command line:

```
> cd .../Middleware/wrapper/java/src/
> javac -verbose stuhfl/*.java
> javac -verbose stuhflBridge/*.java
 > jar cvf stuhfl.jar stuhfl stuhflBridge
```

3.3.4 Java (Linux®) for STUHFL

The user must execute the following steps:

- Build libSTUHFL.so and libJNISTUHFL.so
  - Run the makefile in the JNISTUHFL directory to build both shared libraries. By default the libraries are build as release.

Note: The generated libJNISTUHFL shared object depends on the libSTUHFL and is generated first.

libSTUHFL.so can be found in .../Middleware/clib/STUHFL/bin/ARM/<target>

libJNISTUHFL.so can be found in .../Middleware/wrapper/java/jni/JNISTUHFL/JNISTUHFL/bin/ARM/<target>

Note: The make file depends on the system environment variable JAVA_HOME with the location to the JDK installation.

Command line:

```
> cd .../Middleware/wrapper/java/jni/JNISTUHFL
> make
```

- Build STUHFL.jar (optional, only needed when modifications are applied)
  - Compile the java sources from .../Middleware/wrapper/java/src and package it to a jar archive

Command line:

```
> cd .../Middleware/wrapper/java/src/
> javac -verbose stuhfl/*.java
> javac -verbose stuhflBridge/*.java
 > jar cvf stuhfl.jar stuhfl stuhflBridge
```

3.4 Build instruction STUHFL_demo applications

3.4.1 VS2017 project used in Windows® for STUHFL_demo

The user must execute the following steps:

- Upload firmware on board
- Open the VS2017 solution .\Applications\STUHFL_demo\STUHFL_demo.sln
- Rebuild solution
- Start debugging

Note: The STUHFL_demo tests at startup that the version information of the board and the STUHFL middleware is exactly the same. In case of a version mismatch the demonstartion application terminates.

3.4.2 VS2017 project used in Linux® for STUHFL_demo

The user must execute the following steps:

- Upload FW on board
- Open the VS2017 solution .\Applications\STUHFL_demo\STUHFL_demo_rpi.sln
- Add the IP address of the RPi in the Tools-Options-CrossPlatform options
- In the STUHFL and STUHFL demonstartion project properties set this IP as remote machine
- Update the communication port location (default: /dev/ttyUSB0) to where the ST25RU3993-HPEV board is connected to on the RPi.
  - Check STUHFL_demo.c COM_PORT define
- Rebuild solution
- Start debugging

Note: The STUHFL_demo application tests at startup the version information of the board and the STUHFL middleware. Both must have the same version. In case of a version mismatch the demonstration application terminates.

3.4.3 "makefile" usage for STUHFL_demo

The user must execute the following steps:

- Upload firmware to the reader board
- Ensure <linuxmachine>:./STUHFL/bin/ARM/Release/libSTUHFL.so has been generated (Middleware Linux® shared object generation)
- Copy .\Applications\STUHFL_demo\STUHFL_demo* to <linuxmachine>:./STUHFL_demo/Applications/STUHFL_demo/STUHFL_demo
- Copy .\Applications\STUHFL_demo\makefile to <linuxmachine>:.
- On a Linux® machine, make -C ./makefile

Note: The STUHFL_demo application tests at startup the version information of the reader board and the STUHFL middleware. Both must have the same version. In case of a version mismatch the demonstration application terminates.

Note: This generates the executable .../STUHFL_demo/bin/ARM/Release/STUHFL_demo.out

3.5 Build instruction STUHFL_demo wrapper applications

3.5.1 C# for STUHFL_demo

The user must execute the following steps:

- Upload firmware to the reader board
- Open the VS2017 solution .../Applications/STUHFL_demo_wrapper/STUHFL_demo_cs/STUHFL_demo_cs.sln
- Rebuild solution
- Run or debug the demonstration

3.5.2 Python for STUHFL_demo

The user must execute the following steps:

- Upload firmware on board
- Open .../Applications/STUHFL_demo_wrapper/STUHFL_demo_py/STUHFL_demo_py/STUHFL_demo.py in the Python IDE (i.e. PyCharm)
- Run or debug the demonstration

3.5.3 Java (Windows®) for STUHFL_demo

The user must execute the following steps:

- Upload firmware on board
- Compile .../Applications/STUHFL_demo_wrapper/STUHFL_demo_j/src/demo_j/Program.java
- Run or debug the demonstration

Note: The wrapper libraries STUHFL.dll, JNISTUHFL.dll and STUHFL.jar must be accessible with java library path. In the command line example below the binaries are located in ./lib/Debug/x64/

Note: The STUHFL.jar file must be added to class path. In the command line example below the file is located in ./lib/Debug/x64/

Note: The demonstration program expects to have only one transponder present at the antenna.

Command line:

```
> cd .../Applications/STUHFL_demo_wrapper/STUHFL_demo_j/
> javac -classpath lib/Debug/x64/stuhfl.jar src/demo_j/Program.java
> java -classpath lib/Debug/x64/stuhfl.jar;src -Djava.library.path=lib/Debug/x64/ demo_j/
Program
```

3.5.4 Java (Linux®) for STUHFL_demo

The user must execute the following steps:

- Upload firmware to the reader board
- Compile .../Applications/STUHFL_demo_wrapper/STUHFL_demo_j/src/demo_j/Program.java
- Run or debug the demonstration

Note: The shared libraries libSTUHFL.so, libJNISTUHFL.so and STUHFL.jar must be accessible with the java library path.

In the command line example below the binaries are located in /usr/local/lib/stuhfl and the the LD_LIBRARY_PATH is extended by this directory -> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/stuhfl

Note: The STUHFL.jar file must be added to the classpath. In the command line example below the file is located in ./usr/local/lib/stuhfl

Note: The demonstration program expects to have only one transponder present at the antenna.

Command line:

```
> cd .../Applications/STUHFL_demo_wrapper/STUHFL_demo_j/
> javac -classpath /usr/local/lib/stuhfl/stuhfl.jar src/demo_j/Program.java
> java -classpath /usr/local/lib/stuhfl/stuhfl.jar:src -Djava.library.path=/usr/local/lib/
stuhfl/ demo_j/Program
```

4 Software interface description

For a detailed API description check the available documentation, which is part of the SDK and is available in compiled HTML file format (chm). A brief overview of the API functionality for the various modules is given in the following sections.

Note: For detailed information, refer to the file "Documents/Middleware-STUHFL.chm"

4.1 Device layer

4.1.1 Connections

The table below gives an overview of the connectivity functions.

Table 8. STUHFL connection functionality

Function	Description
STUHFL_F_Connect	Connect to a device via STUHFL
STUHFL_F_Disconnect	Disconnect from current connected device via STUHFL
STUHFL_F_GetCtx	Get device context of current attached device
STUHFL_F_Reset	Reset current attached device

4.1.2 Parameter access

The table below gives an overview of the parameter access functions.

Table 9. STUHFL parameter access functionality

Function	Description
STUHFL_F_SetParam	Generic set parameter function to set value of configuration
STUHFL_F_GetParam	Generic get parameter function to get value of configuration
STUHFL_F_GetParamInfo	Get parameter information
STUHFL_F_SetMultipleParams	Set list of multiple parameters
STUHFL_F_GetMultipleParams	Get list of multiple parameters

4.1.3 Data exchange

The table below gives an overview of the data exchange functions.

Table 10. STUHFL data exchange functionality

Function	Description
STUHFL_F_SendCmd	Send command to device
STUHFL_F_ReceiveCmdData	Receive command to device
STUHFL_F_ExecuteCmd	Send and receive combination in one call
STUHFL_F_GetRawData	Receive raw data from device

4.1.4 Generic maintenance

The table below gives an overview of the generic maintenance functions.

Table 11. STUHFL maintenance functionality

Function	Description
STUHFL_F_GetVersion	Get device version information
STUHFL_F_GetInfo	Get device information
STUHFL_F_Upgrade	Firmware upgrade
STUHFL_F_EnterBootloader	Reboot and enter bootloader
STUHFL_F_Reboot	Reboot FW

4.2 Session layer

4.2.1 Gen2V2 of STUHFL

The table below gives an overview of the Gen2V2 functions.

Table 12. STUHFL Gen2V2 functionality

Function	Description
STUHFL_F_Gen2_Inventory	Gen2 Inventory depending on the current inventory and gen2 configuration
STUHFL_F_Gen2_Select	Gen2 Select command to select or filter Gen2 transponders
STUHFL_F_Gen2_Read	Gen2 Read command to read from the Gen2 transponders
STUHFL_F_Gen2_Write	Gen2 Write command to write data to Gen2 transponders
STUHFL_F_Gen2_Lock	Gen2 Lock command to lock Gen2 transponders
STUHFL_F_Gen2_Kill	Gen2 Kill command to kill Gen2 transponders.
STUHFL_F_Gen2_GenericCmd	Generic Gen2 bit exchange
STUHFL_F_Gen2_QueryMeasureRssi	RSSI measurement during Gen2 Query command

4.2.2 GB/T 29768 of STUHFL

The table below gives an overview of the GB/T 29768 functions.

Table 13. STUHFL GB/T 29768 functionality

Function	Description
STUHFL_F_Gb29768_Inventory	GB/T 29768 Inventory depending on the current inventory and GB/T 29768 configuration
STUHFL_F_Gb29768_Sort	GB/T 29768 Sort command to select or filter GB/T 29768 transponders
STUHFL_F_Gb29768_Read	GB/T 29768 Read command to read from the GB/T 29768 transponders
STUHFL_F_Gb29768_Write	GB/T 29768 Write command to write data to GB/T 29768 transponders
STUHFL_F_Gb29768_Lock	GB/T 29768 Lock command to lock GB/T 29768 transponders
STUHFL_F_Gb29768_Kill	GB/T 29768 Kill command to kill GB/T 29768 transponders.
STUHFL_F_Gb29768_Erase	GB/T 29768 Erase command to erase GB/T 29768 transponders.

4.3 Activity layer

4.3.1 Actions

The table below gives an overview of the action functions.

Table 14. STUHFL action functionality

Function	Description
STUHFL_F_Start	Start actions
STUHFL_F_Stop	Stop action

4.4 STUHFL EVAL API wrapper

4.4.1 Generic

The table below gives an overview of the generic functions to access the evaluation board and retrieve basic information

Table 15. STUHFL EVAL API generic functionality

Function	Description
Connect	Connect to the ST25RU3993 based evaluation board
Disconnect	Disconnect from the current board
Get_BoardVersion	Read the board software and hardware information
Get_BoardInfo	Get human readable software and hardware device information
Reboot	Reboot evaluation board
EnterBootloader	Shutdown FW and enter STM32 ROM bootloader of evaluation board

4.4.2 Configuration

The table below gives an overview of the configuration functions.

Table 16. STUHFL EVAL API configuration functionality

Function	Description
SetRegister	Writes the ST25RU3993 register content
Set_RegisterMultiple	Writes multiple ST25RU3993 registers content at once
GetRegister	Reads one ST25RU3993 register content
Get_RegisterMultiple	Reads multiple ST25RU3993 registers
Set_RwdCfg	Set reader configuration
Get_RwdCfg	Get reader configuration
Set_Gen2_RxFilter	Set reader Rx Filter configuration for Gen2 config
Get_Gen2_RxFilter	Get reader Rx Filter configuration for Gen2 config
Set_Gb29768_RxFilter	Set reader Rx Filter configuration for Gb29768 config
Get_Gb29768_RxFilter	Get reader Rx Filter configuration for Gb29768 config
Set_Gen2_FilterCalibration	Set reader Rx Filter calibration for Gen2
Get_Gen2_FilterCalibration	Get reader Rx Filter calibration for Gen2
Set_Gb29768_FilterCalibration	Set reader Rx Filter calibration for Gb29768
Get_Gb29768_FilterCalibration	Get reader Rx Filter calibration for Gb29768
Set_AntennaPower	Set antenna power
Get_AntennaPower	Get antenna power
Set_Gen2_Timings	Set Gen2 protocols timings
Get_Gen2_Timings	Get Gen2 protocols timings
Set_Gen2_ProtocolCfg	Set Gen2 protocol configuration
Get_Gen2_ProtocolCfg	Get Gen2 protocol configuration
Set_Gb29768_ProtocolCfg	Set Gb29768 protocol configuration
Get_Gb29768_ProtocolCfg	Get Gb29768 protocol configuration
Set_TxRxCfg	Set TxRx configuration
Get_TxRxCfg	Get TxRx configuration
Set_PowerAmplifierCfg	Set power amplifier configuration
Get_PowerAmplifierCfg	Get power amplifier configuration
Set_Gen2_InventoryCfg	Set general Gen2 inventory configuration
Get_Gen2_InventoryCfg	Get general Gen2 inventory configuration
Set_Gb29768_InventoryCfg	Set general Gb29768 inventory configuration
Get_Gb29768_InventoryCfg	Get general Gb29768 inventory configuration

4.4.3 Frequency

The table below gives an overview of the frequency configuration functions.

Table 17. STUHFL EVAL API frequency settings functionality

Function	Description
Set_ChannelList	Set frequency channel list
Get_ChannelList	Get frequency channel list
Set_FreqHop	Set frequency hopping time
Get_FreqHop	Get frequency hopping time
Set_FreqLBT	Set listen before talk configuration
Get_FreqLBT	Get listen before talk configuration
Set_FreqContinuousModulation	Set continuous modulation configuration
Get_FreqRSSI	Get RSSI
Get_FreqReflectedPower	Get reflected power

4.4.4 Tuning

The table below gives an overview of the tuning functions.

Table 18. STUHFL EVAL API tuning functionality

Function	Description
Set_TuningCaps	Set Cin, Clen and Cout of current selected channel list item.
Get_TuningCaps	Get Cin, Clen and Cout of current selected channel list item.

4.4.5 Gen2V2

The tables below gives an overview of the Gen2V2 functions.

Table 19. STUHFL EVAL API Gen2V2 functionality

Function	Description
Gen2_Inventory	Gen2 Inventory depending on the current inventory and gen2 configuration
Gen2_Select	Gen2 Select command to select or filter Gen2 transponders
Gen2_Read	Gen2 Read command to read from the Gen2 transponders
Gen2_Write	Gen2 Write command to write data to Gen2 transponders
Gen2_BlockWrite	Gen2 Block-Write command to write data to Gen2 transponders
Gen2_Lock	Gen2 Lock command to lock Gen2 transponders
Gen2_Kill	Gen2 Kill command to kill Gen2 transponders.
Gen2_GenericCmd	Generic Gen2 bit exchange
Gen2_QueryMeasureRssi	RSSI measurement during Gen2 Query command

4.4.6 GB/T 29768

The table below gives an overview of the GB/T 29768 functions.

Table 20. STUHFL EVAL API GB/T 29768 functionality

Function	Description
Gb29768_Inventory	GB/T 29768 Inventory depending on the current inventory and GB/T 29768 configuration
Gb29768_Sort	GB/T 29768 Sort command to select or filter GB/T 29768 transponders
Gb29768_Read	GB/T 29768 Read command to read from the GB/T 29768 transponders
Gb29768_Write	GB/T 29768 Write command to write data to GB/T 29768 transponders
Gb29768_Lock	GB/T 29768 Lock command to lock GB/T 29768 transponders
Gb29768_Kill	GB/T 29768 Kill command to kill GB/T 29768 transponders.
Gb29768_Erase	GB/T 29768 Erase command to erase GB/T 29768 transponders.

4.4.7 Inventory runner

The table below gives an overview of the inventory runner functions.

Table 21. STUHFL EVAL API inventory runner functionality

Function	Description
Inventory_RunnerStart	Start inventory runner
Inventory_RunnerStop	Stop current inventory runner

Revision history

Table 22. **Document revision history**

Date	Version	Changes
17-Sep-2019	1	Initial release.
26-Nov-2019	2	Updated document title and Section Introduction. Minor text edits across the whole document.
03-Jun-2020	3	Updated: • Section Introduction, Section 1.2 Features, Section 1.3 Hardware requirements, Section 2.3 STUHFL wrappers • Figure 2. ST UHFL library system architecture • title of Section 4.2.2 GB/T 29768 of STUHFL and of Section 4.4.6 GB/T 29768 • Table 13. STUHFL GB/T 29768 functionality, Table 20. STUHFL EVAL API GB/T 29768 functionality Removed Table 1. Hardware requirement
23-Apr-2021	4	Added Section 4.4.1 Generic Updated Section 2.3 STUHFL wrappers, Section 4.4.2 Configuration, Section 4.4.3 Frequency, Section 4.4.4 Tuning, Section 4.2.1 Gen2V2 of STUHFL, Section 4.4.6 GB/T 29768, Section 4.4.7 Inventory runner
05-Oct-2021	5	Updated: • Document title • Section Introduction, Section 1 System overview, Section 1.2 Features, Section 1.3 Hardware requirements, Section 1.4 Building environment, Section 2 Software architecture, Section 2.1 STUHFL API, Section 2.2 STUHFL EVAL API, Section 4.4 STUHFL EVAL API wrapper, Section 2.4 Not exposed layers, Section 2.5 Host side usage, Section 2.6 Device side usage, Section 4 Software interface description • Reworked all Section 3 Source code

Contents

List of tables

List of figures