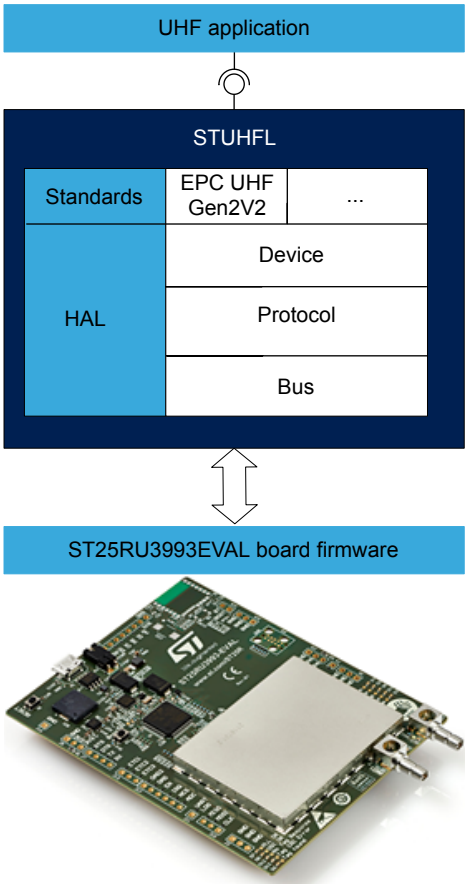
ST25RU3993 ST UHFL library

Introduction

The ST UHF library (short STUHFL) is a middleware software stack written in ANSI C, to build RAIN® RFID enabled applications for reader devices based on ST25RU3993. This document describes the usage of the ST UHF library software API.

Along with this package, a comprehensive demonstration source code is available to illustrate the usage of the ST UHF library for Windows® and Linux® platforms. The complete software stack is ANSI C and POSIX compatible and allows fast and straightforward porting to other operating systems and/or toolchains.



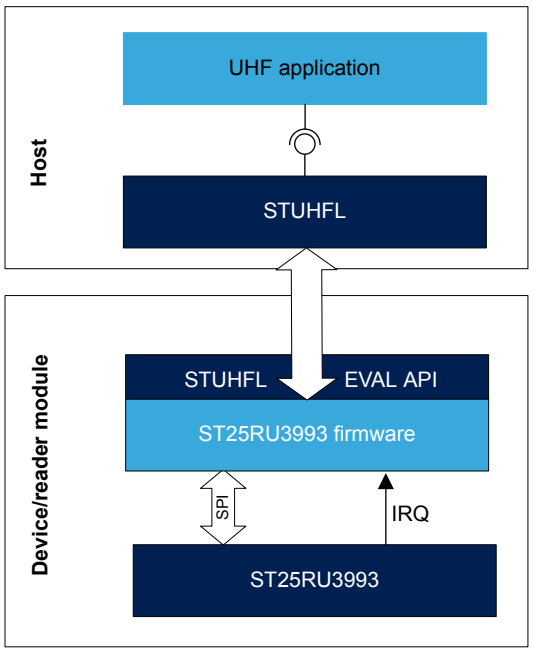
1 System overview

The ST UHF library design is a classic middleware software stack running on a host system. It provides a simple software API to abstract low-level communication details. The following image gives a simplified system overview. The system consist of two hardware components running the involved software components.

The first component, described as device/reader module, runs the firmware with low-level driver implementation for operating the ST25RU3993 reader IC and various protocols. The firmware implements a software interface to the ST UHF library.

The second component, called host side, runs the ST UHF library and abstracts the low-level functionality. It provides a clean software API for UHF applications.

Figure 1. System overview



The ST UHF library runs on host systems with an operating system as well as on embedded systems with OS or native with limited resources. Even a design without a dedicated host module is possible, were the complete 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 to build RAIN® RFID enabled applications for reader devices based on ST25RU3993.
- Mirrored "host and device" library API
- Written in pure ANSI C99
- POSIX compatible

- Straightforward portability across different platforms (MCUs/RTOSs/OSs)
- Compatible with main UHF standards
  - EPC UHF Gen2v2
  - ISO18000-6B
  - Further protocol implementations underway
- Source code example implementation available
  - Windows®
  - Linux® (Raspberry Pi 3B)
  - Further sample platform implementations underway

1.3 Hardware requirements

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

- ST25RU3993 EVAL board
- ST25RU3993 ELANCE module

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

The following table shows the current memory usage requirements for the pre-build ST UHF library running on Windows® and Linux® operating systems.

Table 1. Hardware requirement

-	Windows	Linux Raspberry Pi 3B
Flash memory [kBytes]	92	78
RAM [kBytes]	4	4

When running the ST UHF library on different platforms and/or operating systems the memory requirements might deviate significantly from the table shown above.

1.4 Build environment

Currently all host side software components and demonstration codes are generated by using Microsoft Visual Studio 2017 (VS2017) or newer. The package contains one VS2017 solution hosting projects for two different platforms. One project for Win32 platforms suitable for generating code that runs on Windows 7 platforms and newer. The second project is for generating Arm® platforms suitable for running on Linux devices. The Arm project uses the possibility to do remote compilation with VS2017. This requires a properly configured, Arm device accessible in the same network.

Note: The available ST UHF library project for Arm, and the corresponding example project, uses Raspberry Pi 3B hardware for development.

For more details about project configuration, refer to the appropriate project solution files.

2 Software architecture

The ST UHF library follows a classic stacked middleware software design and consists of components that run on a reader module and on a host system as the default use case.

The complete ST UHF library is ANSI-C and POSIX compliant. This allows easy porting to several other operating systems and platforms.

The ST UHF library currently supports the most common UHF standards. Extensions and more protocols released in upcoming versions.

Working sample source code is available for Windows and Linux, to use different library modules.

2.1 ST UHF library API

The exposed ST UHF library API allows building RAIN applications on ST25RU3993 reader devices. The following three stacked layers represent the complete exposed ST UHF library API.

- Activity
- Session
- Device

In the above layer arrangement starting from device moving up to activity abstraction and complexity, increase as well. The purpose of the device layer is to abstract the hardware specific functionalities, such as direct register access. The session layer uses the device layer to implement entire protocols such as the EPC Gen2V2 standard or other UHF standards and protocols. The activity layer on top uses the device and session layers to abstract functionalities for automated inventory of tags running in the background.

Table 2. Main layers

Layer	Description
Activity	Functionality like background inventory lookup via a build-in "Inventory Runner"
Session	Implementation of different standards or protocols like EPC Gen2V2
Device	Abstraction of hardware specific functionalities like 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 ST UHF library builds on top of this low-level interface and derives its ST25RU3993 dependent functionalities. Applications were no host system is involved directly call the low-level STUHFL EVAL API on reader/device side.

Note: Applications not using the ST UHF library stack may directly attach to this interface.

2.3 ST UHF library wrappers

On top of the ST UHF library host interface, there are several wrappers available for fast prototyping.

- Java/C#/Python (under development, planned release fall 2019)
- ST EVAL API
- Rain RCI(implementation pending, planned release spring 2020)

Java, C# and Python are state of the art programming languages and enable software developers to implement fast and efficiently different kinds of UHF application.

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

The STUHFL EVAL API wrapper (host side) fully emulates the STUHFL EVAL API on the device side. This allows developers to start developing firmware applications on host systems and finish 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 & paste" the firmware application code to the device/reader side. No additional source code modifications are required to run the firmware then 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 exposed ST UHF library layer, several other layers work on the host 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 three layers follows a stack design and depend only on their direct neighbours. The main purpose of the dispatcher layer is to bundle the ST UHF library API function calls in one single module and to forward it to the protocol layer.

The protocol layer transfers the abstracted function calls into transferable data chunks and forwards these data packets to the bus layer.

Finally, the bus layer sends the data chunks over a 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 inversely.

Table 3. Not exposed layers

Layer	Description
Dispatcher	Merge and divide from API layers
Protocol	Encodes and decodes data into TLV formatted data chunks ready for transfer between host and device
Bus	Platform depended transfer layer handling data exchange

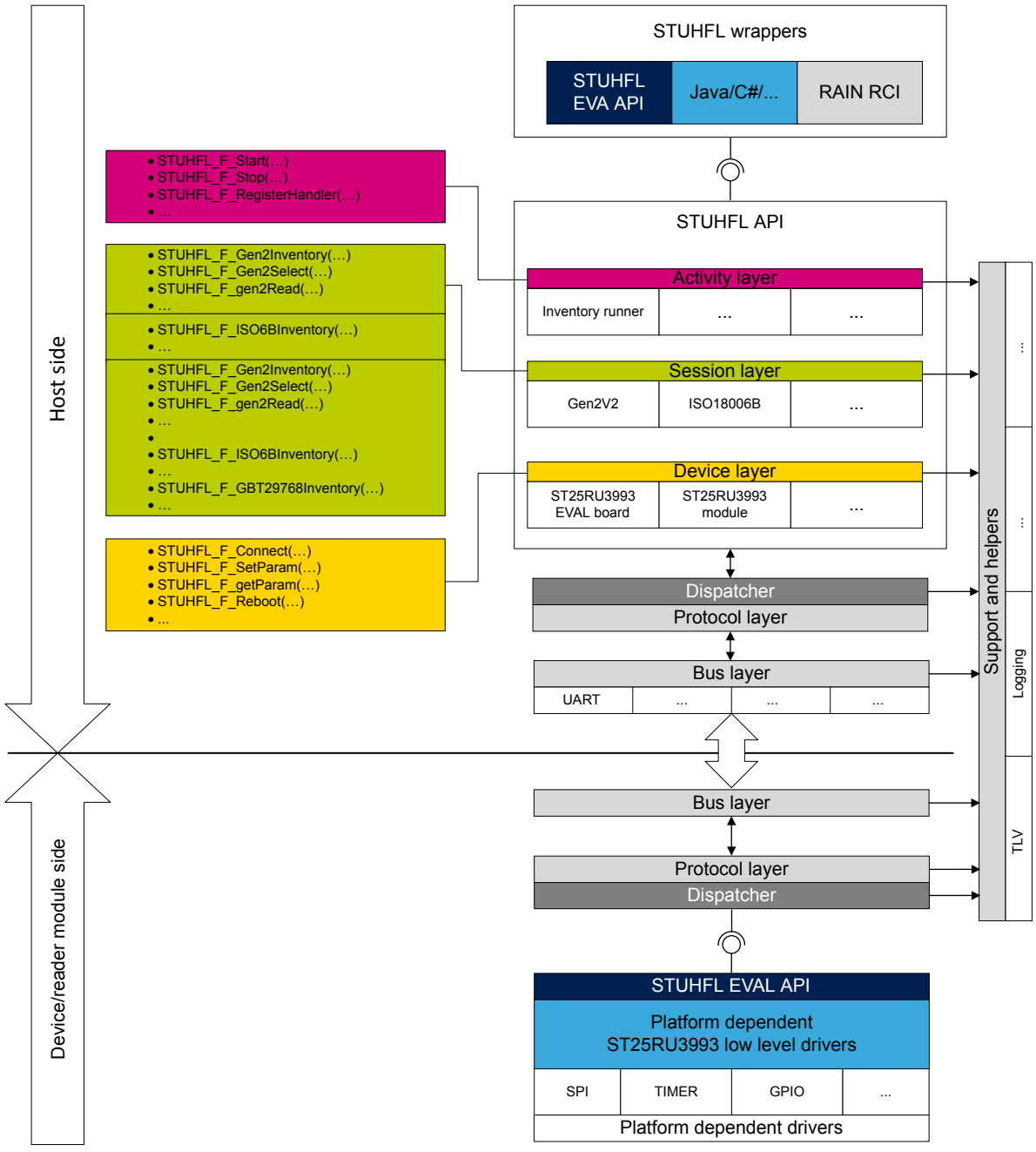
2.4.1 Support and helper

In addition to the above-mentioned not exposed modules, the library implements the following support and helper layers. These support and helper layers provide relevant functionalities and expose their interface to almost all other modules.

Table 4. Helper modules

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

Figure 2. ST UHFL library system architecture

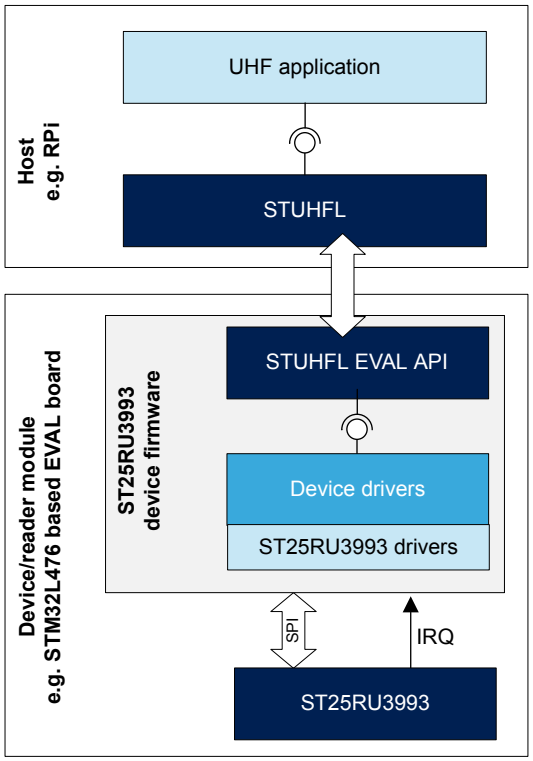


2.5 Host side usage

The functionality exposed by the ST UHF library API enables developer to develop any RAIN application based on ST25RU3993 reader IC devices or modules on host devices. Host systems running the OS Windows or Linux by default supported. As the whole library is ANSI-C and POSIX compatible it also easily builds for other POSIX compatible platform.

The following figure shows a usage example for host side usage.

Figure 3. ST UHF library usage overview

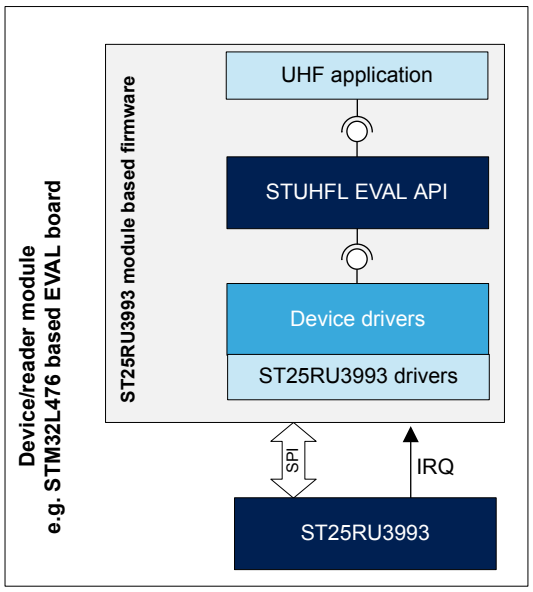


2.6 Device side usage

The functionalities exposed by STUHFL EVAL API provides an abstract interface to the full UHF functionalities provided by ST25RU3993 Reader IC itself without the burden of manually handling low-level register access. Therefore, this interface is device independent and allows implementing any abstract UHF application.

The following figure shows an example for device side usage.

Figure 4. ST UHF library EVAL API device side usage



3 Source code

The complete source code including sample sources for Windows and Linux is available to download in www.st.com

3.1 Directory and file structure

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

Figure 5. Directory structure

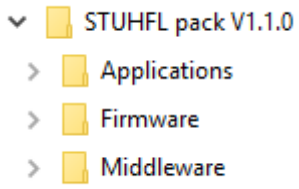


Table 5. Directory description

Directory	Description
Application	Several demonstration project source code
Firmware	Several firmware projects
Middleware	The ST UHF library projects for Windows and Linux as source code

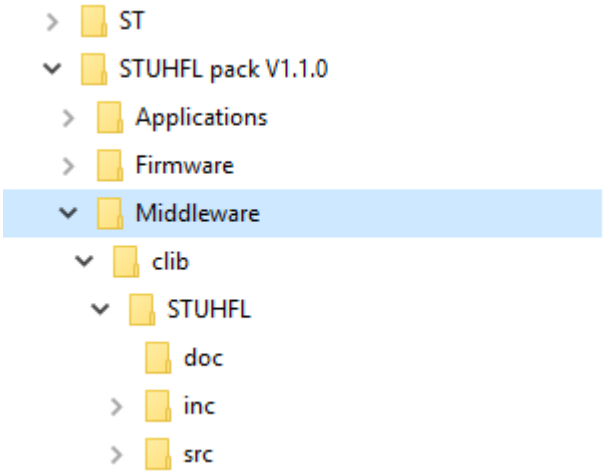
Note: For correct operation, the user must choose one of appropriate firmware packages, delivered along with ST UHF library.

The “STUHFL” folder is located inside the “middleware/clib” folder.

It contains following folders:

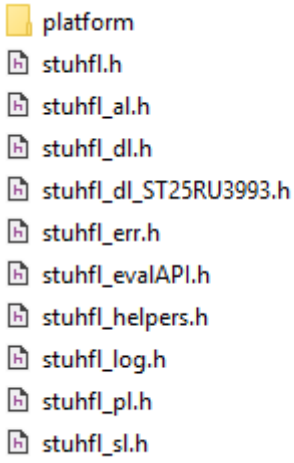
- “doc”: containing a detailed API description as Windows chm file.
- “inc”: containing all relevant interface files
- “src”: containing all implementation files

Figure 6. **STUHFL directory structure**



The “inc” directory contains beside all relevant interface files also a subfolder containing platform specific files.

Figure 7. **"inc" structure**



The table below gives an overview to these files and their content.

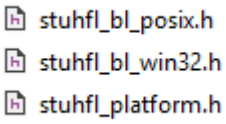
Table 6. **API headers**

Filename	Description	Note
stuhfl.h	Basis header file, containing main typedefs and defines	STUHFL API
stuhfl_al.h	Application module	STUHFL API
stuhfl_dl.h	Generic part of device module	STUHFL API
stuhfl_dl_ST25RU3993.h	ST25RU3993 IC depending device module layer	Not exposed as STUHFL API, internal usage
stuhfl_err.h	Definition of error codes	STUHFL API
stuhfl_evalAPI.h	STUHFL EVAL API definition. Used by STUHFL enabled firmware implementation and as STUHFL wrapper implementation	STUHFL wrapper - STUHFL EVAL API

Filename	Description	Note
stuhfl_helpers.h	Helper module	Not exposed as STUHFL API, internal usage
stuhfl_log.h	Logging module definition	Not exposed as STUHFL API, internal usage
stuhfl_pl.h	Protocol layer definition	Not exposed as STUHFL API, internal usage
stuhfl_sl.h	Session layer	STUHFL API

The “platform” directory contains all relevant files that are important when porting to other platforms.

Figure 8. "Platform" directory



The table below gives an overview to these files and their contents.

Table 7. Platform dependent headers

Filename	Description	
Stuhfl_bl_posix.h	POSIX compatible bus layer for UART communication	
stuhfl_bl_win32.h	UART communication implementation for Windows	
stuhfl_platform.h	Implementation of generic functionality for platform independency	
stuhfl_dl_ST25RU3993.h	ST25RU3993 IC depending device module layer	Not exposed as STUHFL API, internal usage

3.2 ST UHF library solution location

The solution to build a Win32 dynamic link library or Arm based shared object library is located @ ./Middleware/clib/STUHFL.sln.

Note: For software development based on STUHFL the STUHFL demonstration solution contains the STUHFL project and a complete demo application. This allows building and debugging in one go for Win32 and Raspberry Pi.

3.2.1 Build STUHFL for Windows (DLL only)

The user must execute 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 Win32 dynamic link library ready to use to communicate to ST25RU3993 Reader IC based demonstantion boards (Windows host systems only).

3.2.2 Build STUHFL for Linux with Raspberry Pi 3B (shared object only)

The user must execute following steps:

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

Note: This generates a Linux complete shared object to communicate to ST25RU3993 Reader IC based demonstrantion boards (Linux host systems only).

3.3 Implementation sample source code

3.3.1 Build STUHFL-demo for Windows

The user must execute following steps:

1. Open the VS2017 solution ./Applications/STUHFL_demo/STUHFL_demo.sln
2. Rebuild solution
3. Start debugging to debug the demo

Note: For further information about the available demonstration have a look into the documentation of the STUHFL_demo project.

3.3.2 Build STUHFL-demo for Linux

The user must execute following steps:

1. Open the VS2017 solution ./Applications/STUHFL_demo/STUHFL_demo_rpi.sln
2. Add the IP address of your Raspberry Pi in the Tools\Options\Cross platform options.
3. In the STUHFL and STUHFL demonstration project properties set this as your remote machine
4. Update the communication port location (default: /dev/ttyUSB0) where to ST25RU3993 EVAL board is connected to Raspberry Pi.
5. See STUHFL_demo.c and adjust the COM_PORT define
6. Rebuild solution
7. Start debugging to debug the demonstration

Note: For further information about the available demonstration have a look into the documentation of the STUHFL_demo project.

3.4 ST UHF library portability

The ST UHF library is ANSI-C and POSIX compatible. To port the library to other platforms or operating systems a simple cross-compilation or remote-compilation is sufficient. An exception are the low-level communication drivers for the host to device and the reverse communication. These drivers are always platform dependent. All platform dependent components that need to be reworked before successfully porting are located in ./Middleware/clib/STUHFL/src/platform

Note: To port the ST UHF library to other platforms and/or operating system please add the low-level communication drivers for your platform to the ./Middleware/clib/STUHFL/src/platform folder before compilation.

4 Software interface description

For detailed API description have a look into the available documentation. This documentation is part of the source code package and is available in compiled HTML file format (chm). Below only a short overview of the different API functionalities for the different modules is available.

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

4.1 Device layer

Device layer exposed functionality.

4.1.1 Connections

The table below gives an overview about 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 Parmeter access

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

Table 11. STUHFL maintenance functionality

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

4.2 Session layer

Session layer exposed functionality.

4.2.1 Gen2V2 of STUHFL

The table below gives an overview about 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 ISO18000-6B of STUHFL

The table below gives an overview about the ISO18000-6B functions.

Table 13. STUHFL ISO18000-6B functionality

Function	Description
STUHFL_F_ISO6B_Inventory	ISO18000-6B Inventory depending on the current inventory and ISO18000-6B configuration
STUHFL_F_ISO6B_Select	ISO18000-6B Select command to select or filter ISO18000-6B transponders
STUHFL_F_ISO6B_Read	ISO18000-6B Read command to read from the ISO18000-6B transponders
STUHFL_F_ISO6B_Write	ISO18000-6B Write command to write data to ISO18000-6B transponders
STUHFL_F_ISO6B_Lock	ISO18000-6B Lock command to lock ISO18000-6B transponders
STUHFL_F_ISO6B_Kill	ISO18000-6B Kill command to kill ISO18000-6B transponders.
STUHFL_F_ISO6B_Erase	ISO18000-6B Erase command to erase ISO18000-6B transponders.

4.3 Activity layer

Activity layer exposed functionality.

4.3.1 Actions

The table below gives an overview about the action functions.

Table 14. STUHFL action functionality

Function	Description
STUHFL_F_Start	Start foreground or background action
STUHFL_F_Stop	Start action

4.4 STUHFL EVAL API wrapper

EVAL API wrapper exposed functionality.

4.4.1 Configuration

The table below gives an overview about the configuration functions.

Table 15. STUHFL EVAL API configuration functionality

Function	Description
SetRegister	Writes the value to ST25RU3993 register at address
GetRegister	Reads one ST25RU3993 register at address and replies the value to the host
SetRwdCfg	Set reader configuration
GetRwdCfg	Get reader configuration
SetAntennaPower	Set antenna power
GetAntennaPower	Get antenna power
SetGen2Cfg	Set Gen2 configuration
SetTxRxCfg	Set TxRx configuration
SetPA_Cfg	Set power amplifier configuration
SetInventoryCfg	Set general inventory configuration
GetGen2Cfg	Get Gen2 configuration
GetGBT29768Cfg	Get GBT29768 configuration
GetTxRxCfg	Get TxRx configuration
GetPA_Cfg	Get power amplifier configuration
GetInventoryCfg	Get general inventory configuration

4.4.2 Frequency

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

Table 16. STUHFL EVAL API frequency settings functionality

Function	Description
SetFreqProfile	Set frequency profile
SetFreqProfileCustomAppend	Set append frequency to profile
SetFreqHop	Set frequency hopping time
SetFreqLBT	Set listen before talk configuration
SetFreqCondMod	Set continuous modulation configuration
GetFreqRSSI	Get frequency RSSI
GetFreqReflectedPower	Get append frequency to profile
GetFreqProfileInfo	Get frequency hopping time
GetFreqHop	Get listen before talk configuration
GetFreqLBT	Get continuous modulation configuration

4.4.3 Tuning

The table below gives an overview about the tuning functions.

Table 17. STUHFL EVAL API tuning functionality

Function	Description
SetTuning	Set Cin, Clen and Cout
SetTuningTableEntry	Set tuning table entry
SetTuningTableDefault	Revert to default tuning
SetTuningTableSave2Flash	Save current configured tuning table to flash
SetTuningTableEmpty	Delete tuning table entries
GetTuning	Get Cin, Clen and Cout configuration
GetTuningTableEntry	Get tuning table entry
GetTuningTableInfo	Set current tuning table info
Tune	Start tuning

4.4.4 Gen2V2

The tables below gives an overview about the gen2V2 functions.

Table 18. 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_Lock	Gen2 Lock command to lock Gen2 transponders
Gen2_Kill	Gen2 Kill command to kill Gen2 transponders.

Function	Description
Gen2_GenericCmd	Generic Gen2 bit exchange
Gen2_QueryMeasureRssi	RSSI measurement during Gen2 Query command

4.4.5 ISO18000-6B

The table below gives an overview about the ISO18000-6B functions.

Table 19. STUHFL EVAL API ISO18000-6B functionality

Function	Description
ISO6B_Inventory	ISO18000-6B Inventory depending on the current inventory and ISO18000-6B configuration
ISO6B_Select	ISO18000-6B Select command to select or filter ISO18000-6B transponders
ISO6B_Read	ISO18000-6B Read command to read from the ISO18000-6B transponders
ISO6B_Write	ISO18000-6B Write command to write data to ISO18000-6B transponders
ISO6B_Lock	ISO18000-6B Lock command to lock ISO18000-6B transponders
ISO6B_Kill	ISO18000-6B Kill command to kill ISO18000-6B transponders.
ISO6B_Erase	ISO18000-6B Erase command to erase ISO18000-6B transponders.

4.4.6 Inventory runner

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

Table 20. STUHFL EVAL API inventory runner functionality

Function	Description
InventoryRunnerStart	Start inventory runner
InventoryRunnerStop	Stop current inventory runner

Revision history

Table 21. **Document revision history**

Date	Version	Changes
17-Sep-2019	1	Initial release.

Contents

List of tables

List of figures