FINAL DESIGN REPORT TO: Dr. Steven F. Barrett

Associate Dean for Academic Programs

Professor, Electrical & Computer Engineering

FROM: Sokwhan Huh

DATE: Nov 21st, 2016

SUBJECT: Smart Doorlock Final Design Report Draft

For your review, attached is the final design report of the senior design project. Contents include 1) Introduction, briefly summarizing the goals, requirements and specification of the project, 2) Operation, discussing the functionalities and possible use cases of the door lock, 3) Parts, describing the specifications and the rationale for choosing the parts in detail, 4) Development, which captures the approach I will take to complete this project, 5) Project cost, which describes a preliminary analysis of the anticipated cost of the project and 6) References.

Sincerely,

Sokwhan Huh

**Smart Doorlock Final Project Report**

Date: November 21st, 2016

**Primary Author: Sokwhan Huh**

|  |
| --- |
| ***Revision History*** |

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Reason for Changes** |
| 1.0 | May 12, 2016 | Created design report for Spring of 2016 |
| 1.1 | November 18, 2016 | Updated document to reflect actual progress of the project, added technical background section. |
| 1.2 | November 19, 2016 | Added source codes for the project |

|  |
| --- |
| ***Tables of Contents*** |

[**1. Introduction** 1](#_Toc467268288)

[**1.1 Executive Summary** 1](#_Toc467268289)

[**1.2 Terminologies** 2](#_Toc467268290)

[**1.3 Target Market** 3](#_Toc467268291)

[**1.4 Project Requirements** 3](#_Toc467268292)

[**1.5 Technical Specifications** 4](#_Toc467268293)

[**2. Technology Background** 5](#_Toc467268294)

[**2.1 NFC Introduction** 5](#_Toc467268295)

[**2.2 NFC Standards** 5](#_Toc467268296)

[**2.3 ISO 7816-4 Standard** 6](#_Toc467268297)

[**2.4 ISO 14443 Standard** 8](#_Toc467268298)

[**2.5 NDEF Structure** 12](#_Toc467268299)

[**2.6 Android Host-Based Card Emulation** 13](#_Toc467268300)

[**2.7 IoT Introduction** 14](#_Toc467268301)

[**2.8 MQTT Protocol** 15](#_Toc467268302)

[**3. Hardware** 16](#_Toc467268303)

[**3.1 Microcontroller** 16](#_Toc467268304)

[**3.2 NFC Transceiver IC** 17](#_Toc467268305)

[**3.3 LCD** 18](#_Toc467268306)

[**3.4 Keypad** 19](#_Toc467268307)

[**2. Operation** 20](#_Toc467268308)

[**2.1 General** 20](#_Toc467268309)

[**2.2 Active Mode** 20](#_Toc467268310)

[**2.3 Registration Mode** 20](#_Toc467268311)

[**2.4 Configuration Mode** 20](#_Toc467268312)

[**2.5 Web Service** 22](#_Toc467268313)

[**4. Design** 23](#_Toc467268314)

[**4.1 Platform and Dependencies** 23](#_Toc467268315)

[**4.1 Development Plan** 24](#_Toc467268316)

[**4.2 Block Diagram** 26](#_Toc467268317)

[**4.3 High-Level Design (Microcontroller)** 27](#_Toc467268318)

[**4.3 UML Flowchart(Microcontroller)** 27](#_Toc467268319)

[**4.4 High-Level Design (Web Server)** 27](#_Toc467268320)

[**4.3 UML Flowchart(Web Server)** 27](#_Toc467268321)

[**4.5 High-Level Design (Android)** 27](#_Toc467268322)

[**4.3 UML Flowchart(Android)** 27](#_Toc467268323)

[**4.3 Database Schema** 27](#_Toc467268324)

[**4.4 Schematic Diagram** 28](#_Toc467268325)

[**5. Testing** 29](#_Toc467268326)

[**6. Packaging** 30](#_Toc467268327)

[**5. Project Cost** 31](#_Toc467268328)

[**5.1 Preliminary Cost Analysis** 31](#_Toc467268329)

[**5.2 Funding Source** 31](#_Toc467268330)

[**7. Other Considerations** 32](#_Toc467268331)

[**7.1 Security** 32](#_Toc467268332)

[**8. Lessons Learned** 33](#_Toc467268333)

[**9. Source Code** 34](#_Toc467268334)

[**9.1 Microcontroller (C language)** 34](#_Toc467268335)

[**9.2 Web Server (C# language)** 231](#_Toc467268336)

[**9.3 Android (Java language)** 275](#_Toc467268337)

[**10. References** 306](#_Toc467268338)

# **1. Introduction**

## **1.1 Executive Summary**

Traditional mechanical locks have numerous issues pertaining to accessibility and security. Always having to carry keys can be burdensome and keys themselves can be misplaced, broken, stuck in the lock or may not even turn causing great deal of problems to home owners. These locks often are also not built to cater for disabled personnel who may lack the capability of operating such lock. But the bigger problem lies with its insufficient security. Mechanical locks are easily picked by people with minimal knowledge in lock picking and keys are vulnerable to replication by locksmiths. A new type of door lock needs to be designed to better secure your assets.

With the advent of NFC (Near Field Communication) and IoT (Internet of Things), virtually everything around us can now be controlled via nothing more than a smartphone. NFC is now a quintessential technology for transportation systems around the world, as a passenger simply needs to carry his or her smartphone to use the services as opposed to purchasing tickets and transportation cards. Commonly known as Smart Home Technology, IoT is being deployed in many households which allow household products to be controlled in all imaginable ways requiring nothing but an internet connection. The intent of this project is to provide a practical solution to aforementioned problems with traditional mechanical locks by applying both NFC and IoT technologies to an electronic door lock.

## **1.2 Terminologies**

The following acronyms and terminologies are used in this document:

|  |  |
| --- | --- |
| Acronym / Terminology | Description |
| NFC | Near Field Communication. |
| RFID | Radio-Frequency Identification. |
| IOT | Internet of Things. |
| ISO/IEC 14443 | Four-part international standard for contact-less smart cards operating at 13.56 MHz in close proximity. |
| ISO/IEC 7816-4 | International standard for logical structure of data in smart cards. |
| HCE | Host-based Card Emulation. Feature of Android operating system that allows smartphones to emulate a NFC tag in ISO 14443 standard. |
| NDEF | NFC Data Exchange Format. A standardized data format that is used to exchange information between any compatible NFC device. |
| TCP/IP | Transmission Control Protocol / Internet Protocol. Commonly known as Internet protocol suite. Defines set of communications protocols used on the Internet. |
| MQTT | Message Queuing Telemetry Transport. A publish-subscribe based "light weight" messaging protocol for use on top of the TCP/IP protocol. |
| API | Application Programming Interface. |
| OFDM | Orthogonal Frequency-Division Multiplexing |
| AP | Access Point. |
| SSID | Service Set Identifier. |
| ASK | Amplitude-Shift Keying . |
| APDU | Application Protocol Data Unit. Command and response packets for communication with smartcards. Defined in ISO 7816-4 standard. |
| AID | Application ID. Defined in ISO 7816-4 standard to distinguish NFC devices. |
| PICC | Proximity Integrated Circuit Cards. Examples include NFC tags and contactless smart cards. |

## **1.3 Target Market**

This project is intended for the following audience:

* Lay person being able to control their door locks with ease, eliminating the need of carrying a key or memorizing passwords.
* Handicapped personnel who may have difficulties locking and unlocking a traditional door lock.
* People who are looking for additional security measures.

## **1.4 Project Requirements**

The core project requirements are as follows:

* Demonstrate NFC capability
  + Be able to open the door lock with NFC using an Android device. The device must be registered in advance to open the door lock.
* Demonstrate IoT capability
  + Be able to open or permanently lock (lockdown mode) the door lock over the internet using an Android device or by accessing Smart Doorlock’s web server with a browser. The phone must be registered in advance to open the door lock.
* Demonstrate enhanced security system
  + Be able to register phones for access.
  + Be able to clear existing registered phones.

Additional requirements are as follows:

* Implement full logging system for entries.
* Have the door lock function with a battery backup in case of power outage.

## **1.5 Technical Specifications**

* Interfacing via NFC between Smart Doorlock and an Android device using Host-based Card Emulation technology
* Interfacing via IoT to a cloud server over MQTT Protocol
* NFC communication range: 5 cm maximum (from smartphone to Smart Doorlock)
* Wi-Fi communication range: 50 meters maximum (TX 14.5dBM, RX -74.0dBM at 54 OFDM)
* Microcontroller: Texas Instruments CC3200
* NFC Transceiver IC: Texas Instruments TRF7970A
* AC power with DC battery backup
* Minimum backup battery life of 24 hours
* Keypad: Grayhill 4x1 87FC3-201 transparent keypad
* LCD: Newhaven’s NHD-0420D3Z-NSW-BBW-V3-ND 4x20 characters
* Programming Language: C (Microcontroller), Java (Android Application), C# (Web Server)
* Supported Android versions: 4.4 (Kitkat) and above
* Web Server Framework: NancyFX
* Web Server Database: MySQL

# **2. Technology Background**

As Smart Doorlock utilizes some of the most cutting edge technologies available today, this section will briefly provide a background information on the two major technologies (NFC, IoT) driving the project.

## **2.1 NFC Introduction**

NFC is a collection of communication protocols that enable two electronic devices to establish communication in a close proximity, typically within 5 cm of each other. Specifically, NFC is a branch of High-Frequency (HF) RFID (Radio-Frequency Identification) that operates at the 13.56 MHz frequency. NFC is designed to be a secure form of data exchange and unlike RFID, an NFC device is flexible in the sense that it is capable of acting as a reader (active mode) or as a tag (passive mode). This unique feature allows the two devices to communicate in a bi-directional manner (peer-to-peer communication). The following chart displays the speed and two codings NFC employs:

|  |  |  |
| --- | --- | --- |
| Speed | Active device | Passive device |
| 424 kbits/s | Manchester coding, 10% ASK | Manchester coding, 10% ASK |
| 212 kbits/s | Manchester coding, 10% ASK | Manchester coding, 10% ASK |
| 106 kbits/s | Modified Miller, 100% ASK | Manchester coding, 10% ASK |

Table 2.1: Speed and codings used by NFC technology

## **2.2 NFC Standards**

As NFC is a subset of RFID, much of its standards are derived from existing RFID standards including ISO 7816-4, ISO 14443, ISO 15693, ISO 18092 and FeliCa. These standards specify the modulation schemes, coding, transfer speeds, frame format, initialization schemes, collision-control, and transport protocol. The choice of standard entirely dictates the method and capability of communication between the two NFC devices. This document will provide background on ISO 7816-4 and ISO 14443 standards as Smart Doorlock utilizes these two standards to communicate between the door lock and the smart phone.

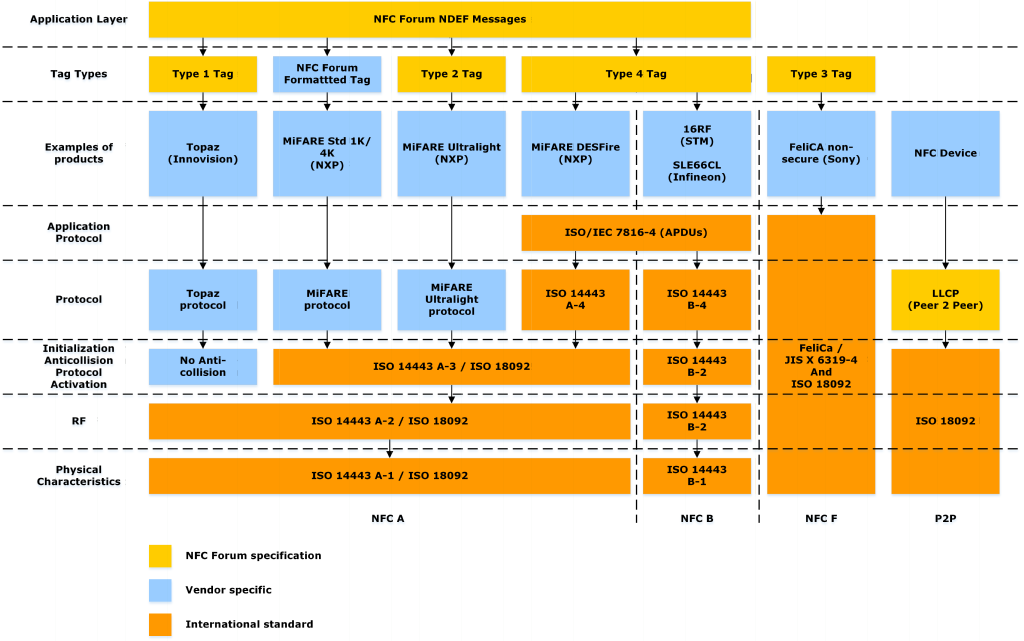


Figure 2.2: NFC protocol stack overview

## **2.3 ISO 7816-4 Standard**

ISO 7816 is an international standard covering electronic identification cards with contacts, such as smart cards. ISO 7816-4 specifically denotes the organization, security and commands for interchange. The standard specifies the following:

* Contents of command-response pairs exchanged at the interface,
* Means of retrieval of data elements and data objects in the card,
* Structures and contents of historical bytes to describe operating characteristics of the card,
* Structures for applications and data in the card, as seen at the interface when processing commands,
* Access methods to files and data in the card,
* A security architecture defining access rights to files and data in the card,
* Means and mechanisms for identifying and addressing applications in the card,
* Methods for secure messaging,
* Access methods to the algorithms processed by the card.

Note that this standard does not cover the internal implementation within the card or the outside world. This is covered in a different standard such as ISO 14443.

APDU (Application Protocol Data Unit) are the command and response packets used in ISO 7816-4 standard. A command APDU consists of an instruction code and associated parameter data. In response to a preceding command APDU, a response APDU is sent back which consists of response data and a response status code. Table 2.31 shows the structure of command-response APDU.

|  |  |  |
| --- | --- | --- |
| Field | Description | Number of bytes |
| Command Header | Class byte denoted CLA | 1 |
| Instruction byte denoted INS | 1 |
| Parameter bytes denoted P1-P2 | 2 |
| Lc field | Absent for encoding Nc = 0, present for encoding Nc > 0 | 0, 1 or 3 |
| Command data field | Absent if Nc = 0, present as a string of Nc bytes if Nc > 0 | Nc |
| Le field | Absent for encoding Ne = 0, present for encoding Ne > 0 | 0, 1, 2 or 3 |
| Response data field | Absent if Nr = 0, present as a string of Nr bytes if Nr > 0 | Nr (at most Ne) |
| Response trailer | Status bytes denoted SW1-SW2 | 2 |

Table 2.31: Command-response APDU pair

ISO 7816-4 also defines a unique identifier called AID (Application Identifier). This identifier is used to distinguish between different NFC devices so that it provides a guarantee method for an NFC device to recognize that it’s communicating to the correct target.

In relation to Smart Doorlock, a SELECT AID command is sent from NFC reader to Android device when the two devices are attempting to establish the NFC communication. For appliactions to be used with existing NFC readers, the AIDs that those readers are looking for are typically well-known and publicly registered (for example, the AIDs of payment networks such as Visa and MasterCard). If a new reader infrastructure is to be deployed for a proprietary application, registering a unique AID in accordance to ISO 7816-5 specification is recommended to avoid collisions with other applications.

Table 2.32 shows an actual example of APDU sent from Smart Doorlock to Android device in an attempt to check if the target has an AID of “D2760000850101”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Command Header | | | | Lc | Command Data Field | Le |
| CLA | INS | P1 | P2 | AID |
| 0x00 | 0xA4 | 0x04 | 0x00 | 0x07 | 0xD2760000850101 | 0x00 |

Table 2.32: Command-response APDU pair

## **2.4 ISO 14443 Standard**

ISO 14443 is a four-part international standard for contactless smart cards. The four parts, denoted from ISO 14443-1 to ISO 14443-4 are as follows:

* Part 1 [ISO 14443-1]: Defines the size and physical characteristics of the card. This standard lists several environmental stresses that the card must be capable of withstanding without permanent damage to the functionality.
* Part 2 [ISO 14443-2]: Defines the RF power and signal interface. Two signaling schemes known as Type A and Type B are defined in this part. Both communication schemes are half duplex with a 106 kbis per second data rate in each direction. Data transmitted by the card is load modulated with a 847.5 kHz subcarrier.
* Part 3 [ISO 14443-3]: Defines the initialization and anticollision protocols for Type A and Type B. The anticollision commands, responses, data frame, and timing are defined in this part. The initialization and anticollision scheme is designed to permit the construction of multi-protocol readers capable of communication with both Type A and Type B cards. Both card types wait silently in the field for a polling command. A multi-protocol reader would poll one type of card, complete any transactions with cards responding and then poll for the other type of card and transact witht hem.
* Part 4 [ISO 14443-4]: Defines the high-level data transmission protocols for Type A and Type B. The protocol defined in this part is also capable of transferring APDU as defined in ISO 7816-4 and of AID selection.

Difference between Type A and Type B is that Type A uses Miller encoding (delay encoding) with an amplitude modulation at 100% while Type B uses Manchester encoding with amplitude modulation at 10%. Note that in case of Smart Doorlock, Android devices emulate Type A cards with HCE (Host-Based Card Emulation) technology. Figure 2.4 provides an overview of ISO 14443 protocol stack.

ISO/IEC 14443

ISO 14443-1  
Physical Properties

ISO 14443-2 Type A

Power, Bit Transmission

ISO 14443-3 Type A

Activation, Anticollision

ISO 14443-4

Transport Protocol

ISO 7816-4  
APDU, Application Protocol

Figure 2.41: ISO 14443 Protocol Stack

As ISO 14443-1 and ISO 14443-2 focuses on the physical and RF characteristics of the standard, the actual implementation of these two protocols are dictated by the choice of hardware (in case of Smart Doorlock, this refers to NFC reader and Android device). This section will primarily focus on ISO 14443-3 and ISO 14443-4 for the actual implementation of the protocol.

ISO 14443-3 defines the activation and anticollision sequence. In order for an NFC reader to detect PICC (Proximity Integrated Circuit Cards), the NFC reader goes through a sequence of polling to repeatedly send request commands known as REQA (Request Command, Type A). When a PICC of Type A is exposed to an unmodulated operating field, the request to command is accepted and responded with ATQA (Answer to Request, Type A). Next, the NFC reader undergoes a looping process known as anticollision detection to determine if at least two PICCs are simultaneously transmitting bit patterns. As bit patterns are continuously sent from the PICC, the NFC reader needs to determine if the received UID (Unique Identifier, Type A) is complete during its anticollision loop.

Each time the NFC reader receives a UID, it transmits a SEL (Select) command appended with either partial or fully received UID to the PICC, which is responded with SAK (Select Acknowledge). The SAK response indicates that UID transmission is complete or not. ISO 14443A standard permits UIDs of different lengths – 4 bytes, 7 bytes and 10 bytes. Therefore, in case the PICC has a UID of 7 or 10 bytes, it will notify the NFC reader by setting the cascade bit in SAK command. This causes the reader to increase the cascade level in anticollision loop and repeat the process starting from selecting the UID. If the UID is 10 bytes long, the anticollision loop is ran a third time. Table 2.42 and 2.43 shows the SEL command and the SAK response structure.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | Meaning |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | ‘93’ Select cascade level 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | ‘95’ Select cascade level 2 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | ‘97’ Select cascade level 3 |
| 1 | 0 | 0 | 1 | Other values except those here above | | | | RFU |

Table 2.42 SEL command structure

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | Meaning |
| x | x | x | x | x | 1 | x | x | Cascade bit set: UID not complete |
| x | x | 1 | x | x | 0 | x | x | UID complete, PICC compliant with ISO 14443-4 |
| x | x | 0 | x | x | 0 | x | x | UID complete, PICC not compliant with ISO 14443-4 |

Table 2.43 SAK response structure

Note that UID serves a completely different purpose than AID as mentioned in ISO 7816-4 standard. Contrary to its name, there is no specification in the standard that mandates the UID to be truly unique. As it will be apparent in Android HCE section of this document, the UID is randomly generated on an Android device each time the device attempts to perform NFC communication. Hence, the UID should not be relied on for actual identification of the NFC device or PICC in the actual implementation of an application. Figure 2.44 is a flowchart for initialization and anticollision procedure of ISO 14443-3.

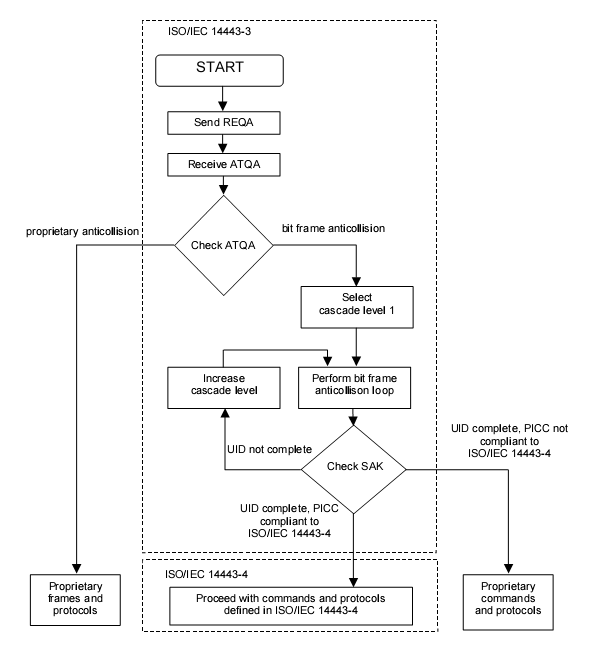


Figure 2.44 – Initialization and anticollision flowchart for NFC reader

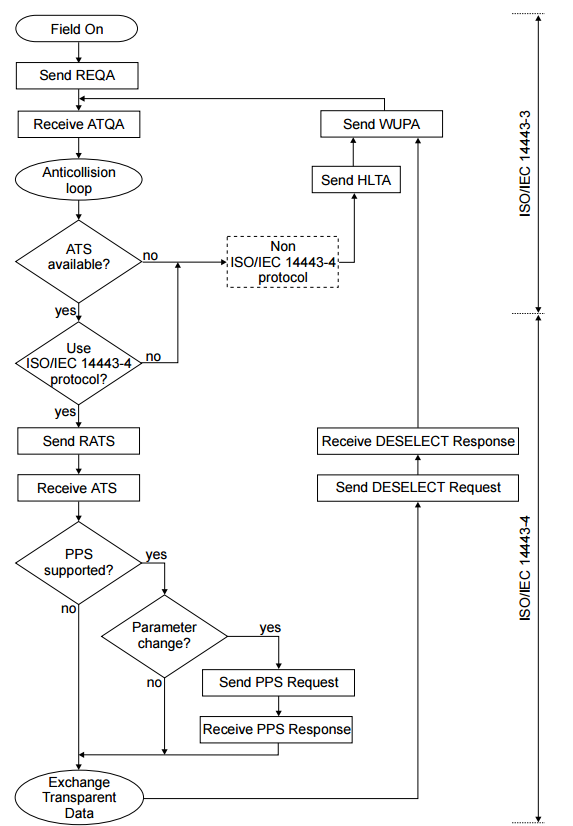
ISO 14443-4 specifies a half-duplex block transmission protocol featuring the special needs of contactless environment and defines the activation and deactivation sequence of the protocol. When PICC activation sequence as defind in ISO 14443-3 is complete, the SAK byte is checked for availability of an ATS (Answer To Select). If ATS is not available, then the target of NFC reader is assumed to be non ISO14443-4 compliant. If it is available, the NFC reader sends RATS (Request for Answer To Select) to the PICC. The PICC must respond with ATS as an answer to the RATS. NFC reader can check the ATS to determine if PICC supports any changeable parameters, such as transmission rate. The NFC reader may optionally send a PPS command if the PICC has any modifiable parameters. The protocol activation sequence for ISO 14443-4 is shown in Figure 2.45.

Figure 2.45 – Protocol activation sequence for ISO 14443-4

## **2.5 NDEF Structure**

NDEF (NFC Data Exchange Format) is a standardized data format that is used to exchange information between any compatible NFC devices. It is a binary format structured in messages, intended to be used on the application layer. Each record is made up of a header, which contains metadata about the record, such as the record type, length etc. The header is followed by a payload, which is the actual content of the message being transmitted.

Each record’s payload can be one of several different data types. The header for each record contains metadata describing the record and its place in the mssage, followed by its type and ID. Figure 2.51 shows the structure of the bits and bytes of an NDEF record.

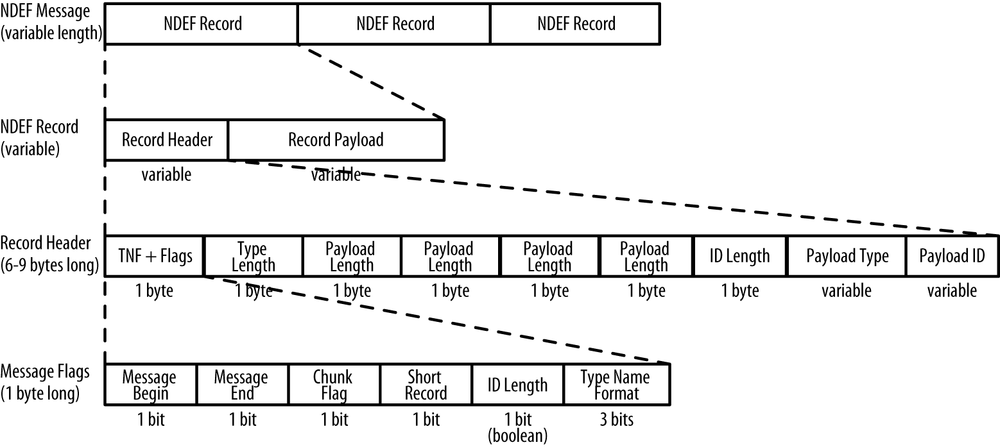


Figure 2.51 NDEF Message Structure

TNF (Type Name Format) indicates how to interpret the type field. The seven possible TNF values are shown in Table 2.52

|  |  |  |
| --- | --- | --- |
| TNF | Meaning | Description |
| 0 | Empty | Empty record with no payload |
| 1 | Well-Known | One of several pre-defined types laid out in the NFC Forum RTD specification |
| 2 | MIME media-type | An Internet media type as defined in RFC 2046 |
| 3 | Absolute URI | A URI as defined in RFC 3986 |
| 4 | External | A user-defined value |
| 5 | Unknown | Type is unknown |
| 6 | Unchanged | Only for middle and terminating records of chunked payloads |
| 7 | Reserved | Reserved by the NFC Forum for future use |

Table 2.52 Type Name Format values

For use in proprietary applications, the choice of TNF is often 04 (external). Smart Doorlock uses this TNF to transmit its set of messages defined by the application protocol.

Use of NDEF structure is not mandated for NFC communication by ISO 7816-4 and ISO 14443. However, large data may have to be transmitted in chunks due to the nature of NFC transactions being short. Thus, NDEF structure is generally considered to be lingua franca for NFC devices and certain protocols in Android devices, such as ISO 18092 (Peer-to-Peer) is strictly configured to handle NDEF type messages only.

## **2.6 Android Host-Based Card Emulation**

Android-powered devices with NFC support three main modes of operation:

* + Reader / writer mode, allowing the NFC device to read and/or write passive NFC tags and stickers. Supported from version Gingerbread (2.3.3).
  + Peer-to-Peer mode, allowing the NFC device to exchange data with other NFC peers (Android Beam). Supported from version Ice Cream Sandwitch (4.0).
  + Card emulation mode, allowing the NFC device itself to act as an NFC card. The emulated NFC card can then be accessed by an external NFC reader. Supported from version KitKat (4.4)

For bi-directional NFC communication, Peer-to-Peer mode or Card emulation mode may be chosen as the mode of operation. While Peer-to-Peer mode has the advantage that the API (Application Programming Interface) is more widely supported than Card emulation mode, Android OS requires the user to manually “tap” the screen to transmit the payload to the target with Peer-to-Peer mode. As this manual intervention by the user can cause usability issues, Smart Doorlock’s Android application takes advantage of HCE (Host-Based Card Emulation) mode to perform NFC communication.

HCE emulates NFC cards that are based on ISO 14443 and ISO 7816-4 specification. By default, Type A technology is supported by the Android but support for Type B technology is optional.

When an NFC card is emulated, the data is routed to the host CPU on which Android applications are running directly. Figure 2.6 illustrates how HCE works.

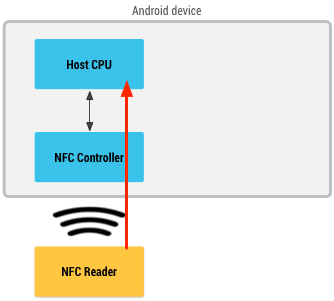


Figure 2.6 NFC card emulation on an Android device

Majority of details pertaining to ISO 14443 and ISO 7816-4, such as the anti-collision sequence and protocol activation are all handled by Android automatically are abstracted away in Android APIs hence, this allows Android developers to primarily focus on the highest NFC layer available to them, but any implementation change in the underneath protocol is practically disallowed. HCE based applications must define an AID for NFC readers to be able to select for identification (See ISO 7816-4 standard).

## **2.7 IoT Introduction**

IoT (Internet of Things) is the internetworking of physical devices embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to exchange data. As the name “Things” indicate, it can refer to a variety of devices such as heart monitoring implants, biochip transponders, automobiles with built-in sensors, or field operation devices that assist firefighters in search and rescue operations. IoT is expected to offer advanced connectivity of devices, systems, and services that goes beyond machine-to-machine communications and covers a variety of protocols, domains, and applications.

Many connectivity options exist today for IoT including Bluetooth, Zigbee, Wi-Fi, and 2G/3G/4G cellular. As Smart Doorlock is expected to be installed in buildings with access point available nearby, Wi-Fi was chosen as main option for IoT connection. Thus, this document will illustrate the technology and protocol behind Smart Doorlock’s Wi-Fi connectivity.

## **2.7 MQTT Protocol**

MQTT (Message Queing Telemetry Transport) is a connectivity protocol specifically designed for use in IoT. It is a publish/subscribe, simple and lightweight messaging protocol for use on top of the TCP/IP protocol. Its design makes it ideal for constrained devices and low-bandwidth, high-latency or unreliable networks.

MQTT defines methods (also referred to as verbs) to indicate the desired action to be performed on the identified resource. The representation of this resource, wherther pre-existing data or data that is generated dynamically depends entirely on the implementation of the server, known as MQTT broker. Table 2.71 shows the available MQTT methods.

|  |  |
| --- | --- |
| Method | Description |
| Connect | Waits for a connection to be established with the server. |
| Disconnect | Waits for the MQTT client to finish any work it must do, and for the TCP/IP session to disconnect. |
| Subscribe | Subscribes to a given topic to receive messages in the futre via Publish. |
| Unsubscribe | Unsubscribes from a given topic to no longer receive published message on the topic. |
| Publish | Publishes a message to a topic. The message is broadcasted to all clients subscribed to the topic. |

Table 2.71 MQTT Methods

After a MQTT client is connected to a broker, it can publish messages. MQTT has a topic-based filtering of the messages on the broker; hence each message must contain a topic, which will be used by the broker to forward the message to interested clients. Each message typically has a payload which contains the actual data to transmit in byte format. As MQTT is data-agnostic, the format of the data is up to the implementation of application. An example MQTT packet is shown in figure 2.72.

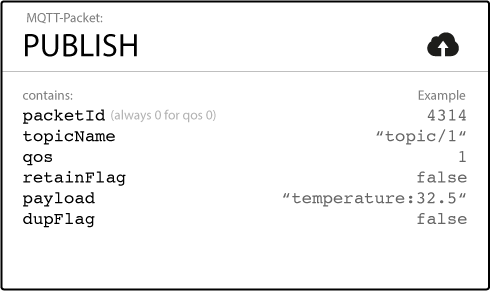


Figure 2.72 Example of MQTT Packet

Aside from the performance bottleneck of the broker, there is no hard limitation on the number of MQTT clients that can connect. As mentioned earlier, a published message from a client is only broadcasted to other clients that subscribed to the topic of interest. This enables a simple yet efficient management of multiple IoT devices being connected to a single MQTT broker as illustrated in Figure 2.73.



Figure 2.73 Example of MQTT Broker Handling Message Publish

A QoS (Quality of Service) level can be defined for each message being published. The level (0,1,2) determines the guarantee of message reaching the other end (client or broker). The minimal level zero, often referred to as “fire and forget”, guarantees a best effort delivery. The message sent won’t be acknowledged by the receiver or stored and redelivered by the sender. Using QoS level one guarantees that a message will be delivered at least once to the receiver. However, there is a chance that a duplicate message is delivered to the receiver. The highest level of QoS two guarantees that each message is received only once by the counter part. This is by far the safest and also the slowest quality of service level. The guarantee is provided by two flows there and back between sender and receiver. If a receiver gets a QoS two publish, it will process the publish message accordingly and acknowledge it to the sender with a PUBREC message. Security is of a highest concern in Smart Doorlock thus, QoS level of two was used.

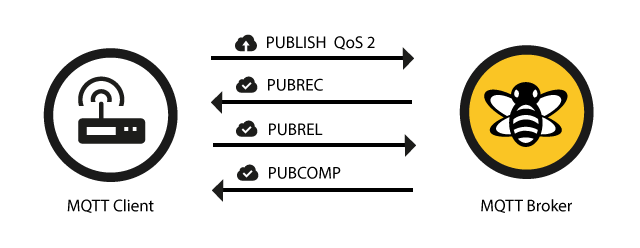


Figure 2.74 Example of QoS Two Message Publish and Response

# **3. Hardware**

## **3.1 Microcontroller**

Texas Instruments CC3200 SimpleLink™ was chosen for the microcontroller for this project. Relevant specifications of the chip are as follows:

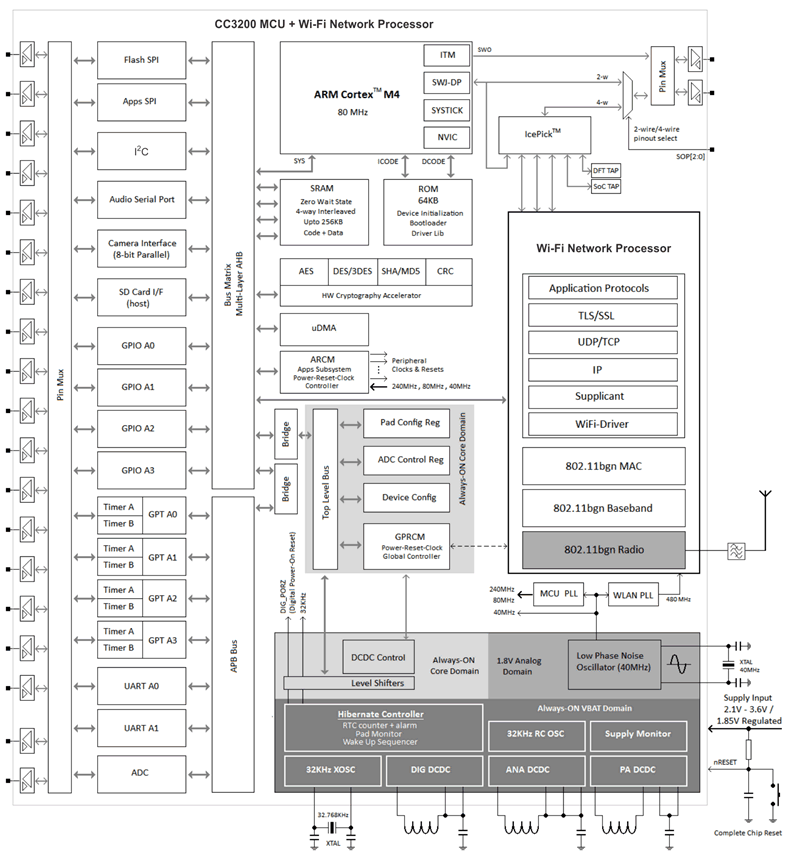
* Single-Chip Wireless MCU
* ARM®Cortex®-M4 Core at 80 MHz
* Up to 27 Individually Programmable, Multiplexed GPIO Pins
* 1 Serial Peripheral Interface (SPI)
* Embedded Memory RAM (Up to 256KB)

Figure 3.1: Functional block diagram of Texas Instruments CC3200

## **3.2 NFC Transceiver IC**

Texas Instruments TRF7970A was chosen for the NFC transceiver IC. A core requirement of this project is to be able to perform P2P communication via NFC and this chip satisfies this requirement. Relevant specifications of the chip are as follows:

* Supports Near Field Communication (NFC) Standards NFCIP-1 (ISO/IEC 18092) and NFCIP-2 (ISO/IEC 21481)
* Parallel or SPI Interface (With 127-Byte FIFO)
* Integrated Encoders, Decoders, and Data Framing for NFC Initiator, Active and Passive Target Operation for All Three Bit Rates (106 kbps, 212 kbps, 424 kbps) and Card Emulation

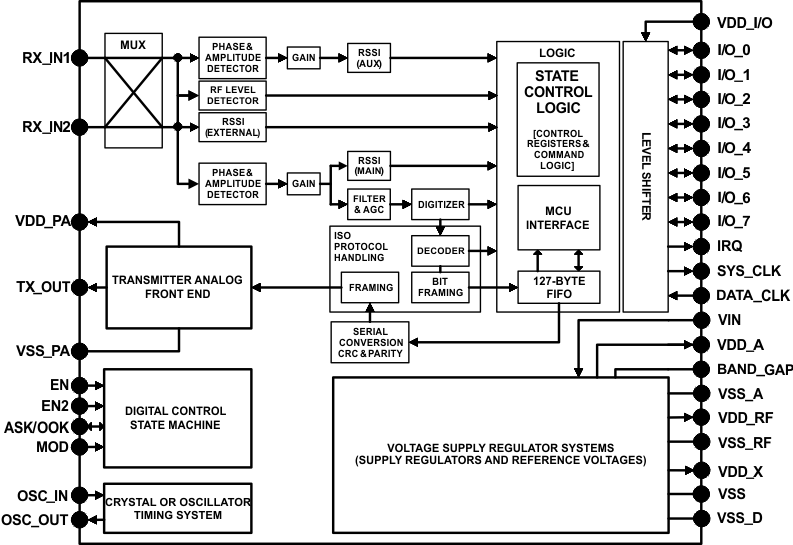


Figure 3.2: Functional block diagram of Texas Instruments TRF7970A

## **3.3 LCD**

Due to the number of initial configuration required, an LCD display that can display a relatively large number of characters is required. An LCD capable of interfacing in UART was chosen to limit the number of connections lines to the MCU. This project will use Newhaven’s NHD-0420D3Z-NSW-BBW-V3-ND LCD as it is capable of displaying 80 characters total and supports UART interfacing. Following is the specification of the LCD:

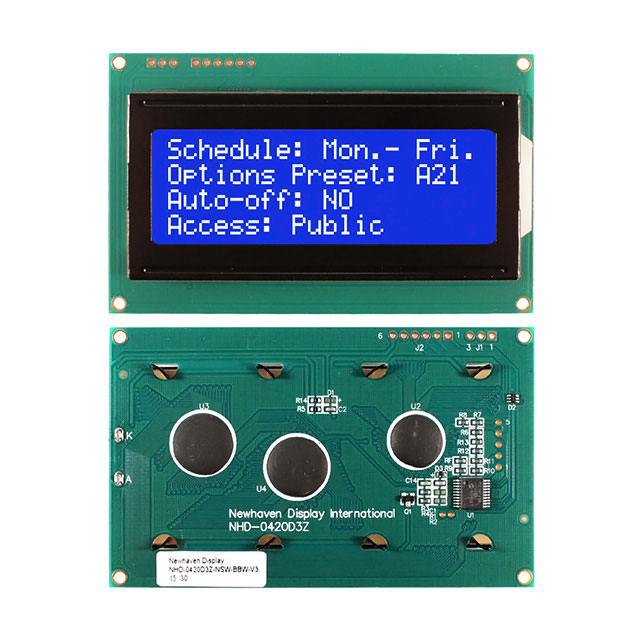
* Display Format: 20 x 4
* Character Format: 5 x 8 Dots
* Character Size: 4.75mm H x 2.95mm W
* Outline: 98.00mm x 60.00mm x 13.50mm
* Viewing Area: 76.00mm L x 25.20mm W
* Interface: UART

Figure 3.3: NHD-0420D3Z-NSW-BBW-V3-ND LCD unit

## **3.4 Keypad**

A keypad is required for the navigation of the menus described in section 2: Operation of this document. This project will use Storm Interface’s Storm 720GFXI Series (7207-121W20) keypad. Relevant specifications of the keypad are as follows:

* 3 x 4 Matrix keypad
* Switch Type: Snap Dome
* Pins required: 10
* Mounting Type: Panel Mount, Front



Figure 3.4: Storm Interface’s Storm 720GFXI Series

# **4. Operation**

## **4.1 General**

The Smart Doorlock will have three modes of operation: 1) Active, 2) Registration and 3) Configuration. Current mode of operation is displayed on LCD and the user is able to shift through the modes using the keypad. It should be noted that any IoT related functionalities are purely optional as the door lock may not have access to internet on the premises.

## **4.2 Active Mode**

This is the default mode of operation for the door lock. In order to operate the door lock by smartphone contact (NFC) or by accessing the Smart Doorlock’s web service (IoT), the door lock must currently be set in this mode. Only registered phones will be able to open the door using NFC (see registration mode). A login ID with a password is required to operate the door lock over the web service. Records of access are written inside the microcontroller’s log each time the door lock is operated which can be queried by accessing the web service.

## **4.3 Registration Mode**

This is the mode to register or unregister a given smartphone. Upon entering this mode, a list of already registered smartphones will be shown on the LCD. Using the arrow keys on the keypad, the user can navigate through registered phones and press a designated “clear” button to delete it from the list. Upon pressing a designated “new” button, the LCD will display “contact smartphone on the door lock to register”, indicating that the door lock is ready to register a new phone. When the phone is in contact with the reader in this instance, NFC communication is initiated and the phone is successfully registered into the door lock.

## **4.4 Configuration Mode**

Settings for the Smart Doorlock can be adjusted in this mode. There are several configurations to adjust from:

* Door lock operation: NFC / Web Service / Both   
  Default setting is NFC. To enable web service access, Smart Doorlock must be configured in advance to ensure proper connectivity with the access point (see IoT settings below).
* Enable / Disable Lockdown Mode  
  When lockdown mode is initiated, operating the door lock via NFC or web service is entirely forbidden. Intended for emergent cases only.
* IoT settings  
  In order for the IoT functionality to work, Smart Doorlock must be connected to an active AP. Due to the complexity of configuring the AP connection using the keypad, all settings must be entered in the application of smartphone, then the settings will be transferred via NFC. A confirmation message will be displayed on the LCD upon the transmission of settings. The following IoT settings are available:
  + Set / Reset admin password
  + Configure AP settings (SSID, Password, DHCP/Static IP)

## **4.5 Web Service**

Upon successfully configuring AP, the door lock will automatically be added to a list of devices in the web server in order to maintain the connection. At any point the connection is lost (based on the keep alive interval set in the protocol), the door lock is automatically removed from the web server’s list. As long as the connection is maintained, users can access the web service using a browser to control the door lock. The following is a list of functionalities offered by the web service:

* Open / Lock door lock
* Lockdown Mode
* Create / Modify / Delete web service accounts
* View / Clear entry logs

# **5. Design**

## **5.1 Platform and Dependencies**

## **5.2 Block Diagram**

The following is a functional block diagram of Smart Doorlock. The diagram shows the relationship between each electronic components and the method of interface between hardware and software.

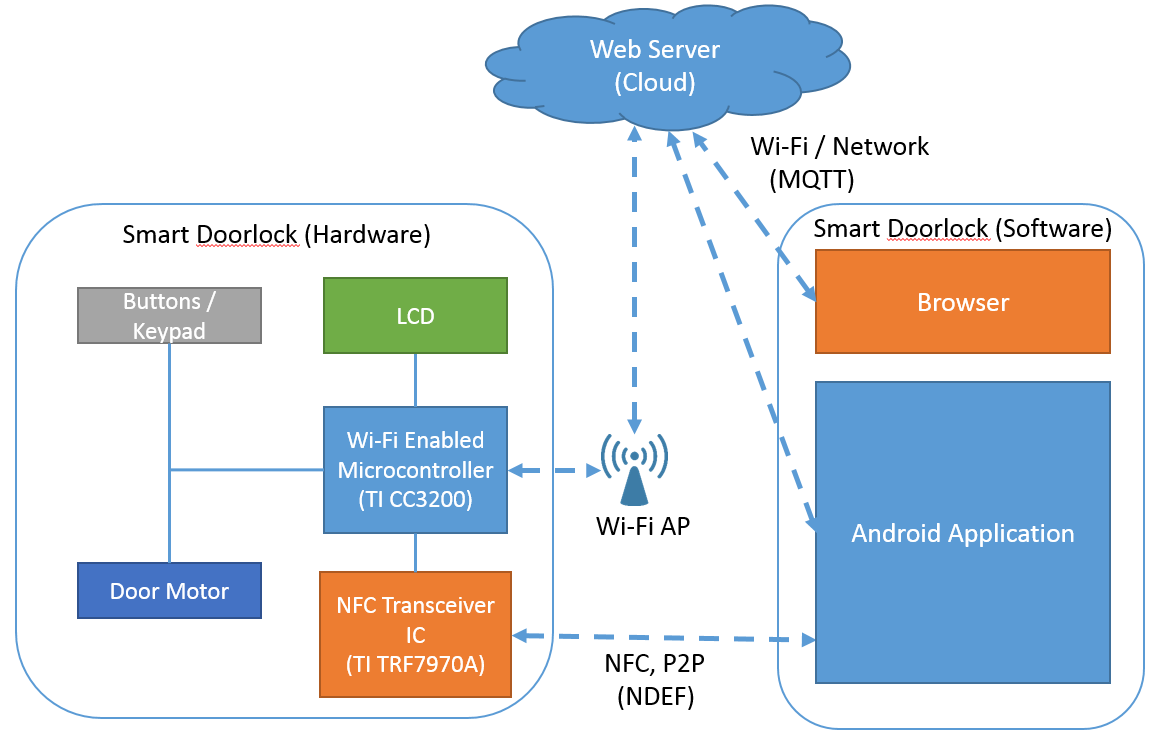


Figure 4.2: Functional block diagram of Smart Doorlock

## **5.3 High-Level Design (Microcontroller)**

## **5.4 UML Flowchart(Microcontroller)**

## **5.5 High-Level Design (Web Server)**

## **5.6 UML Flowchart(Web Server)**

## **5.7 High-Level Design (Android)**

## **5.8 UML Flowchart(Android)**

## **5.9 Database Schema**

## **5.10 Schematic Diagram**

The following is a schematic diagram of the hardware part of Smart Doorlock.



## **5.11 Development Plan**

To ensure the project can be successfully completed, the project will take the following phases of development:

|  |  |  |
| --- | --- | --- |
| Phase | Content | Estimated date of completion |
| 1 | Design the case and any related mechanic parts using Solidworks | August 15, 2016 |
| 2 | Build a prototype of the hardware with only microcontroller and NFC Transceiver IC connected. Ensure the microcontroller is able to utilize NFC communication by reading and writing to a simple tag. | August 22, 2016 |
| 3 | Connect LCD, Keypad and door servo. Test their basic functionalities. Begin coding logic for active / registration mode of the phone. | September 5, 2016 |
| 4 | Begin developing android application, switch NFC Transceiver IC from tag mode to P2P mode. Ensure P2P communication via NDEF protocol works as intended. Test that phone registration and door lock activation via NFC works without any issue. | September 19, 2016 |
| 5 | Begin writing logic for MQTT server on microcontroller’s side. Ensure MQTT protocol works as intended by directly performing test calls from a browser connected to the same AP. | October 3, 2016 |
| 6 | Begin developing web server for the cloud. Ensure the communication works when the calls to control APIs are invoked outside the network microcontroller is connected in. | October 17, 2016 |
| 7 | Design UI for android application, browser and build a case for the door lock. Assemble the final product. | November 7, 2016 |
| 8 | Stress test the final product | December 5, 2016 |

# **6. Testing**

# **7. Packaging**

# **8. Project Cost**

## **8.1 Preliminary Cost Analysis**

The following table is an estimation of the total project cost. All figures are in US dollars.

|  |  |  |
| --- | --- | --- |
| Type | Part name | Cost ($, USD) |
| Microcontroller | Texas Instruments CC3200 | 16.78 |
| Microcontroller development board | Texas Instruments CC3200 LaunchPad | 29.99 |
| NFC transceiver IC | Texas Instruments TRF7970A | 6.98 |
| NFC transceiver IC development kit | Texas Instruments TRF7970A booster pack | 24.99 |
| Keypad | Storm 720GFXI Series (7207-121W20) | 39.14 |
| LCD | Electronic Assembly EADIP203G-4NLED LCD | 30.32 |
| Total | | **145.20** |

The figure includes the parts needed for development as well (CC3200 LaunchPad and TRF7970A booster pack). Excluding parts necessary for development, the actual total of the final product is estimated to be $90.22 USD. The figure also excludes any labor fee that may occur.

## **8.2 Funding Source**

A total amount of $250 from Volpi Cupal fund will be used to cover for the expenses of the project.

# **7. Other Considerations**

## **7.1 Security**

# **8. Lessons Learned**

# **9. Source Code**

## **9.1 Microcontroller (C language)**

|  |
| --- |
| **main.c** |
| /\*  \* Smart Doorlock  \* main.c  \*  \* Created on: 2016. 8. 27.  \*  \* Author: Sokwhan Huh  \*/  // C-Library includes  **#include** <string.h>  // Driverlib includes  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "hw\_memmap.h"  **#include** "hw\_common\_reg.h"  **#include** "hw\_types.h"  **#include** "hw\_ints.h"  **#include** "uart.h"  **#include** "interrupt.h"  **#include** "pinmux.h"  **#include** "utils.h"  **#include** "prcm.h"  **#include** "simplelink.h"  **#include** "gpio.h"  // Common interface include  **#include** "common.h"  **#include** "uart\_if.h"  **#include** "gpio\_if.h"  // Project includes  **#include** "sd\_globals.h"  **#include** "network.h"  **#include** "keypad.h"  **#include** "lcd.h"  **#include** "nfc.h"  **#include** "mqtt\_client.h"  **#include** "spi\_l.h"  **#include** "s\_flash.h"  **#include** "menu.h"  **#include** "mcu.h"  **#define** APP\_NAME "Smart Doorlock"  //RTOS Related Defines  **#define** OSI\_STACK\_SIZE 4096 /\* 2048 \*/  **#define** SPAWN\_TASK\_PRIORITY 9  //SD Related Defines  **#define** DOORLOCK\_OPEN\_DELAY 4000  **#define** PHONE\_REGISTER\_DELAY 6000  **#define** MENU\_NAVIGATE\_DELAY 110  **#define** BUZZER\_DELAY 55  //Globals  **unsigned** **int** g\_firstTimeSetup;  **unsigned** **int** g\_appMode;  **unsigned** **int** g\_currMenuOption;  **static** **unsigned** **int** g\_nfcFirstTimeSetup = 0;  //Function Prototypes  **static** **void** **SmartDoorlockNFCTask**(**void** \*pvParameters);  **static** **void** **SmartDoorlockMenuTask**(**void** \*pvParameters);  **static** **void** **SmartDoorlockInitTask**(**void** \*pvParameters);  **static** **void** **DisplayBanner**(**char** \* AppName)  {  Report("\n\n\n\r");  Report("\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");  Report("\t\t %s \n\r", AppName);  Report("\t\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");  Report("\n\n\n\r");  }  **static** **void** **BoardInit**(**void**)  {  // Enable Processor  MAP\_IntMasterEnable();  MAP\_IntEnable(FAULT\_SYSTICK);  **PRCMCC3200MCUInit**();  }  **static** **void** **SoundBuzzer**(**unsigned** **int** count) {  **unsigned** **int** i;  **for** (i = 0; i < count; i++) {  GPIO\_IF\_Set(PIN\_BUZZER,1);  **osi\_Sleep**(BUZZER\_DELAY);  GPIO\_IF\_Set(PIN\_BUZZER,0);  **osi\_Sleep**(BUZZER\_DELAY);  }  GPIO\_IF\_Set(PIN\_BUZZER,0);  }  **static** **void** **OpenDoor**(**unsigned** **char** \*logMsg) {  Report("Opening Doorlock\n\r");  SmartDoorlockLCDDisplay(*LCD\_DISP\_OPENING\_DOOR*);  **if** (g\_ConfigData.operationMode != *OPER\_NFC\_ONLY*) {  MqttPublishLockAccess(logMsg);  }  g\_appMode = *MODE\_OPENING\_DOOR*;  GPIO\_IF\_Set(13,1);  SoundBuzzer(1);  **osi\_Sleep**(DOORLOCK\_OPEN\_DELAY);  GPIO\_IF\_Set(13,0);  g\_appMode = *MODE\_ACTIVE*;  SmartDoorlockLCDDisplay(*LCD\_DISP\_ACTIVE*);  Report("Closing Doorlock\n\r");  }  **static** **void** **RebootSmartDoorlock**() {  g\_appMode = *MODE\_EXIT*;  Report("Disconnecting from MQTT/AP\n\r");  /\* Mqtt\_ClientExit();  Network\_IF\_DisconnectFromAP();  Network\_IF\_DeInitDriver();\*/  SmartDoorlockLCDDisplay(*LCD\_DISP\_REBOOTING*);  **osi\_Sleep**(2000);  RebootMCU();  Report("Rebooting");  }  **static** **void** **RegisterNewPhone**() {  g\_appMode = *MODE\_REGISTERING\_PHONE*;  SmartDoorlockLCDDisplay(*LCD\_DISP\_REGISTERING\_PHONE*);  Report("Register Phone\n\r");  **strcpy**(g\_ConfigData.doorlockPhoneId[g\_ConfigData.regDoorlockCount],nfcCmdPayload);  **strcpy**(g\_ConfigData.doorlockRegDate[g\_ConfigData.regDoorlockCount],nfcCmdPayload2);  Report("Writing Phone ID: %s\n\r",g\_ConfigData.doorlockPhoneId[g\_ConfigData.regDoorlockCount]);  Report("Date: %s\n\r",g\_ConfigData.doorlockRegDate[g\_ConfigData.regDoorlockCount]);  g\_ConfigData.regDoorlockCount++;  ManageConfigData(SF\_WRITE\_DATA\_RECORD);  **osi\_Sleep**(PHONE\_REGISTER\_DELAY);  **if** (g\_firstTimeSetup) {  **if** (g\_ConfigData.operationMode == *OPER\_NFC\_ONLY*) {  SmartDoorlockLCDDisplay(*LCD\_DISP\_REBOOTING*);  **osi\_Sleep**(2000);  RebootMCU();  }  **else** {  SmartDoorlockLCDDisplay(*LCD\_DISP\_WIFI\_SETUP\_NFC*);  g\_appMode = *MODE\_WIFI\_CONFIG\_NFC*;  }  }  **else** {  SmartDoorlockLCDDisplay(*LCD\_DISP\_REGISTER\_ACTIVE*);  g\_appMode = *MODE\_REGISTER\_ACTIVE*;  }  }  **static** **long** **IsPhoneIdRegistered**(**char** \*phoneId) {  **int** i;  Report("Received ID: %s\n\r",phoneId);  **for** (i = 0; i < g\_ConfigData.regDoorlockCount; i++) {  Report("Comparing: %s\n\r",g\_ConfigData.doorlockPhoneId[i]);  **if** (**strcmp**(phoneId,g\_ConfigData.doorlockPhoneId[i]) == 0) {  **return** 1;  }  }  **return** 0;  }  **static** **void** **NFCWifiConfig**() {  g\_appMode = *MODE\_REGISTERING\_PHONE*;  SmartDoorlockLCDDisplay(*LCD\_DISP\_WIFI\_SETUP\_NFC\_CONFIGURING*);  Report("Wifi Config (NFC)\n\r");  **strcpy**(g\_ConfigData.SSID,nfcCmdPayload);  **strcpy**(g\_ConfigData.Password,nfcCmdPayload2);  Report("SSID: %s\n\r",g\_ConfigData.SSID);  Report("Password: %s\n\r",g\_ConfigData.Password);  ManageConfigData(SF\_WRITE\_DATA\_RECORD);  **osi\_Sleep**(3000);  SmartDoorlockLCDDisplay(*LCD\_DISP\_REBOOTING*);  **osi\_Sleep**(2000);  RebootMCU();  }  **static** **void** **SmartDoorlockMenuTask**(**void** \*pvParameters) {  **if** (g\_firstTimeSetup) {  g\_currMenuOption = *MENU\_OPERATION\_SETUP*;  MenuProcessConfig(*ENTER*);  }  **while** (g\_firstTimeSetup) {  buttonEnum pressedBtn = getPressedButton();  **if** (g\_appMode == *MODE\_REGISTER\_ACTIVE* && !g\_nfcFirstTimeSetup) {  g\_nfcFirstTimeSetup = 1;  // Start the SmartDoorlock NFC task  **osi\_TaskCreate**( SmartDoorlockNFCTask,  (**const** **signed** **char**\*)"Smart Doorlock NFCTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  }  MenuProcessConfigInner(pressedBtn);  **osi\_Sleep**(MENU\_NAVIGATE\_DELAY);  }  **while** (g\_appMode != *MODE\_INITIALIZE\_COMPLETE*) {  **osi\_Sleep**(1);  }  g\_currMenuOption = 0;  Report("Initializing Menu\n\r");  g\_appMode = *MODE\_ACTIVE*;  SmartDoorlockLCDDisplay(*LCD\_DISP\_ACTIVE*);  /\*g\_appMode = MODE\_MENU;  MoveMenu(g\_currMenuOption);\*/  **for** (;;) {  **if** (g\_appMode == *MODE\_EXIT*) {  RebootSmartDoorlock();  **return**;  }  buttonEnum pressedBtn = getPressedButton();  **if** (g\_appMode == *MODE\_MENU*) {  MenuProcessMain(pressedBtn);  }  **else** **if** (g\_appMode == *MODE\_ACTIVE*) {  **if** (pressedBtn == *CANCEL*) {  g\_appMode = *MODE\_MENU*;  MoveMenu(g\_currMenuOption);  }  }  **else** **if** (g\_appMode == *MODE\_CONFIG*) {  MenuProcessConfig(pressedBtn);  }  **else** **if** (g\_appMode == *MODE\_REGISTER\_ACTIVE* ||  g\_appMode == *MODE\_UNREGISTER\_PHONE* ||  g\_appMode == *MODE\_OPERATION\_SETUP* ||  g\_appMode == *MODE\_WIFI\_CONFIG\_NFC*) {  MenuProcessConfigInner(pressedBtn);  }  **osi\_Sleep**(MENU\_NAVIGATE\_DELAY);  }  }  **static** **void** **SmartDoorlockNFCTask**(**void** \*pvParameters) {  // Init NFC hardware  Report("Initializing NFC\n\r");  NFCInit();  **unsigned** **char** logPayload[100];  **if** (!g\_firstTimeSetup) {  g\_appMode = *MODE\_INITIALIZE\_COMPLETE*;  }  **for** (;;) {  **if** (g\_appMode == *MODE\_EXIT*)  **return**;  **if** (g\_appMode != *MODE\_ACTIVE* && g\_appMode != *MODE\_REGISTER\_ACTIVE* && g\_appMode != *MODE\_WIFI\_CONFIG\_NFC*) {  **osi\_Sleep**(1);  **continue**;  }  nfcCmdEnum cmd = readNFCTag();  **switch** (cmd) {  **case** *NFC\_OPEN\_DOORLOCK*:  **if** (g\_appMode == *MODE\_ACTIVE*) {  **if** (IsPhoneIdRegistered(nfcCmdPayload)) {  **memset**(logPayload, 0, **sizeof**(logPayload));  **strncpy**((**char**\*)logPayload, "LOG|NFC|", 8);  **strncat**((**char**\*)logPayload, nfcCmdPayload, **sizeof**(nfcCmdPayload));  OpenDoor(logPayload);  }  **else** {  g\_appMode = *MODE\_UNREGISTERED\_PHONE\_TAPPED*;  SmartDoorlockLCDDisplay(*LCD\_DISP\_UNREGISTERED\_PHONE\_TAPPED*);  **osi\_Sleep**(3000);  SmartDoorlockLCDDisplay(*LCD\_DISP\_ACTIVE*);  g\_appMode = *MODE\_ACTIVE*;  }  }  **break**;  **case** *NFC\_REG\_PHONE*:  **if** (g\_appMode == *MODE\_REGISTER\_ACTIVE*) {  SoundBuzzer(1);  RegisterNewPhone();  }  **break**;  **case** *NFC\_WIFI\_CONFIG*:  **if** (g\_appMode == *MODE\_WIFI\_CONFIG\_NFC*) {  SoundBuzzer(1);  NFCWifiConfig();  }  **break**;  **default**:  **break**;  }  }  }  **static** **void** **SmartDoorlockIoTTask**(**void** \*pvParameters) {  **unsigned** **char** logPayload[100];  **while** (g\_appMode == *MODE\_INITIALIZING*) {  **osi\_Sleep**(1);  }  SmartDoorlockLCDDisplay(*LCD\_DISP\_CONNECT\_AP*);  **int** retVal = ConnectAP(g\_ConfigData.SSID, g\_ConfigData.Password);  **if** (retVal != 0) {  SmartDoorlockLCDDisplay(*LCD\_DISP\_AP\_CONN\_FAILURE*);  g\_ConfigData.operationMode = *OPER\_NFC\_ONLY*;  **osi\_Sleep**(3000);  // Start the SmartDoorlock NFC task  **osi\_TaskCreate**( SmartDoorlockNFCTask,  (**const** **signed** **char**\*)"Smart Doorlock NFCTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  **return**;  }  Report("Connection Successful!\n\r");  retVal = initMqtt();  **if** (retVal != 0)  **return**;  **osi\_Sleep**(100);  SmartDoorlockLCDDisplay(*LCD\_DISP\_CONNECT\_MQTT*);  retVal = mqttConnect();  **osi\_Sleep**(500);  **if** (retVal != 0) {  SmartDoorlockLCDDisplay(*LCD\_DISP\_MQTT\_CONN\_FAILURE*);  g\_ConfigData.operationMode = *OPER\_NFC\_ONLY*;  **osi\_Sleep**(3000);  // Start the SmartDoorlock NFC task  **osi\_TaskCreate**( SmartDoorlockNFCTask,  (**const** **signed** **char**\*)"Smart Doorlock NFCTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  **return**;  }  **if** (g\_ConfigData.operationMode == *OPER\_NFC\_IOT*) {  // Start the SmartDoorlock NFC task  **osi\_TaskCreate**( SmartDoorlockNFCTask,  (**const** **signed** **char**\*)"Smart Doorlock NFCTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  }  **else** {  g\_appMode = *MODE\_INITIALIZE\_COMPLETE*;  }  event\_msg RecvQue;  **for**(;;)  {  **if** (g\_appMode == *MODE\_EXIT*)  **return**;  **osi\_MsgQRead**( &g\_PBQueue, &RecvQue, OSI\_WAIT\_FOREVER);  **if** (g\_appMode != *MODE\_ACTIVE*) {  Report("IoT Task: Msg received but not in active mode\n\r");  **osi\_Sleep**(1);  **continue**;  }  **if**(*BROKER\_DISCONNECTION* == RecvQue.event)  {  attemptReconnect();  }  **if**(*DOORLOCK\_OPEN* == RecvQue.event)  {  **if** (g\_appMode == *MODE\_OPENING\_DOOR*) {  Report("IoT Task: Doorlock is already being opened\n\r");  **continue**;  }  **memset**(logPayload, 0, **sizeof**(logPayload));  **strncpy**((**char**\*)logPayload, "LOG|IOT|SmartDoorlock", 21);  OpenDoor(logPayload);  }  }  }  **static** **void** **SmartDoorlockInitTask**(**void** \*pvParameters) {  // Init LCD  lcdInit();  lcdClearScreen();  SmartDoorlockLCDDisplay(*LCD\_DISP\_INIT*);  SoundBuzzer(3);  //Initialize simplelink  **long** lMode = **sl\_Start**(0, 0, 0);  ASSERT\_ON\_ERROR(lMode);  //ManageConfigData(SF\_DELETE\_DATA\_RECORD);  **if** (ManageConfigData(SF\_TEST\_DATA\_RECORD) < 0) {  ManageConfigData(SF\_CREATE\_DATA\_RECORD);  g\_firstTimeSetup = 1;  g\_appMode = *MODE\_OPERATION\_SETUP*;  **osi\_Sleep**(1000);  SmartDoorlockLCDDisplay(*LCD\_DISP\_FIRST\_TIME\_SETUP*);  **osi\_Sleep**(4000);  // Start the Menu task  **osi\_TaskCreate**( SmartDoorlockMenuTask,  (**const** **signed** **char**\*)"MenuTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  **return**;  }  **else** {  ManageConfigData(SF\_READ\_DATA\_RECORD);  g\_firstTimeSetup = 0;  // Start the Menu task  **osi\_TaskCreate**( SmartDoorlockMenuTask,  (**const** **signed** **char**\*)"MenuTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  }  **if** (g\_ConfigData.operationMode == *OPER\_NFC\_IOT* || g\_ConfigData.operationMode == *OPER\_IOT\_ONLY*) {  g\_appMode = *MODE\_INITIALIZING\_IOT*;  // Start the SmartDoorlock IoT task  **osi\_MsgQCreate**(&g\_PBQueue,"PBQueue",**sizeof**(event\_msg),10);  **osi\_TaskCreate**( SmartDoorlockIoTTask,  (**const** **signed** **char**\*)"Smart Doorlock IoTTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  }  **else** {  g\_appMode = *MODE\_INITIALIZING\_NFC*;  // Start the SmartDoorlock NFC task  **osi\_TaskCreate**( SmartDoorlockNFCTask,  (**const** **signed** **char**\*)"Smart Doorlock NFCTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  }  }  **int** **main**(**void**) {  // Initailizing the board  BoardInit();  // Muxing for Enabling GPIO, UART\_TX and UART\_RX.  PinMuxConfig();  // Init SPI  SPIInit();  // Init Terminal  InitTerm();  ClearTerm();  DisplayBanner(APP\_NAME);  GPIO\_IF\_Set(PIN\_BUZZER,0);  //Start the simplelink host  **VStartSimpleLinkSpawnTask**(SPAWN\_TASK\_PRIORITY);  //Set app mode to initializing  g\_appMode = *MODE\_INITIALIZING*;  // Start the SmartDoorlock Initialization Task  **osi\_TaskCreate**( SmartDoorlockInitTask,  (**const** **signed** **char**\*)"Smart Doorlock InitTask",  OSI\_STACK\_SIZE, NULL, 1, NULL );  **osi\_start**();  **return** 0;  } |

|  |
| --- |
| **sd\_globals.h** |
| /\*  \* sd\_globals.h  \*  \* Created on: Oct 31, 2016  \* Author: shuh  \*/  **#ifndef** SD\_GLOBALS\_H\_  **#define** SD\_GLOBALS\_H\_  **typedef** **enum**  {  *MODE\_INITIALIZING*,  *MODE\_INITIALIZING\_IOT*,  *MODE\_INITIALIZING\_NFC*,  *MODE\_INITIALIZE\_COMPLETE*,  *MODE\_MENU*,  *MODE\_ACTIVE*,  *MODE\_CONFIG*,  *MODE\_OPENING\_DOOR*,  *MODE\_OPERATION\_SETUP\_FIRST*,  *MODE\_OPERATION\_SETUP*,  *MODE\_REGISTER\_ACTIVE*,  *MODE\_REGISTERING\_PHONE*,  *MODE\_UNREGISTERED\_PHONE\_TAPPED*,  *MODE\_UNREGISTER\_PHONE*,  *MODE\_WIFI\_TEST*,  *MODE\_WIFI\_CONFIG\_NFC*,  *MODE\_EXIT*  } appModeEnum;  **typedef** **enum**  {  *MENU\_ACTIVE*,  *MENU\_CONFIG*,  *MENU\_REBOOT*  } appMenuEnum;  **typedef** **enum**  {  *MENU\_OPERATION\_SETUP*,  *MENU\_REGISTER\_PHONE*,  *MENU\_UNREGISTER\_PHONE*,  *MENU\_WIFI\_CONFIG*,  *MENU\_WIFI\_TEST*,  *MENU\_FACTORY\_RESET*  } configMenuEnum;  **typedef** **enum**  {  *OPER\_NFC\_ONLY*,  *OPER\_IOT\_ONLY*,  *OPER\_NFC\_IOT*,  *OPER\_NOT\_SET*  } operEnum;  //Globals  **extern** **unsigned** **int** g\_appMode;  **extern** **unsigned** **int** g\_currMenuOption;  **extern** **unsigned** **int** g\_firstTimeSetup;  **#endif** /\* SD\_GLOBALS\_H\_ \*/ |

|  |
| --- |
| **mcu.h** |
| /\*  \* mcu.h  \*  \* Created on: Oct 31, 2016  \* Author: shuh  \*/  **#ifndef** MCU\_H\_  **#define** MCU\_H\_  **extern** **void** **RebootMCU**();  **#endif** /\* MCU\_H\_ \*/ |

|  |
| --- |
| **mcu.c** |
| /\*  \* mcu.c  \*  \* Created on: Oct 31, 2016  \* Author: shuh  \*/  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "hw\_memmap.h"  **#include** "hw\_common\_reg.h"  **#include** "hw\_types.h"  **#include** "hw\_ints.h"  **#include** "prcm.h"  **void** **RebootMCU**() {  **sl\_Stop**(30);  MAP\_PRCMHibernateIntervalSet(330);  MAP\_PRCMHibernateWakeupSourceEnable(PRCM\_HIB\_SLOW\_CLK\_CTR);  MAP\_PRCMHibernateEnter();  } |

|  |
| --- |
| **gpio\_if.h** |
| /\*  \* gpio\_if.h  \*  \* Created on: 2016. 9. 7.  \* Author: Sokwhan  \*/  **#ifndef** \_\_GPIOIF\_H\_\_  **#define** \_\_GPIOIF\_H\_\_  //GPIO # Definitions  **typedef** **enum**  {  PIN\_LCD\_RS = 0, //4 RS (CS) H / L H=Data, L=Command  PIN\_LCD\_RW = 3, //5 R/W (SID) H / L H=Read, L=Write  PIN\_LCD\_E = 4, //6 E (SCLK) H Enable (falling edge)  PIN\_LCD\_D0 = 5, //7 D0 (SOD) H / L Display Data, LSB  PIN\_LCD\_D1 = 6, //8 D1 H / L Display Data  PIN\_LCD\_D2 = 7, //9 D2 H / L Display Data  PIN\_LCD\_D3 = 8, //10 D3 H / L Display Data  PIN\_LCD\_D4 = 9, //11 D4 (D0) H / L Display Data  PIN\_LCD\_D5 = 10, //12 D5 (D1) H / L Display Data  PIN\_LCD\_D6 = 11, //13 D6 (D2) H / L Display Data  PIN\_LCD\_D7 = 12 //14 D7 (D3) H / L Display Data, MSB  } lcdPinEnum;  **typedef** **enum**  {  PIN\_KEYPAD\_B1 = 3,  PIN\_KEYPAD\_B2 = 4,  PIN\_KEYPAD\_B3 = 5,  PIN\_KEYPAD\_B4 = 6,  } keypadPinEnum;  **#define** PIN\_BUZZER 8  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  // API Function prototypes  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **extern** **void** GPIO\_IF\_GetPortNPin(**unsigned** **char** ucPin,  **unsigned** **int** \*puiGPIOPort,  **unsigned** **char** \*pucGPIOPin);  **extern** **void** GPIO\_IF\_ConfigureNIntEnable(**unsigned** **int** uiGPIOPort,  **unsigned** **char** ucGPIOPin,  **unsigned** **int** uiIntType,  **void** (\*pfnIntHandler)(**void**));  **extern** **void** GPIO\_IF\_Set(**unsigned** **int** gpioNum, **unsigned** **int** state);  **extern** **void** GPIO\_IF\_Toggle(**unsigned** **int** gpioNum);  **extern** **unsigned** **char** GPIO\_IF\_GetVal(**unsigned** **char** ucPin,  **unsigned** **int** uiGPIOPort,  **unsigned** **char** ucGPIOPin);  **static** **void** GPIO\_IF\_SetVal(**unsigned** **char** ucPin,  **unsigned** **int** uiGPIOPort,  **unsigned** **char** ucGPIOPin,  **unsigned** **char** ucGPIOValue);  **extern** **void** GPIO\_Set(**unsigned** **char** ucGPIONum);  **extern** **void** GPIO\_Clear(**unsigned** **char** ucGPIONum);  **extern** **void** GPIO\_Toggle(**unsigned** **char** ucGPIONum);  **extern** **unsigned** **char** GPIO\_IF\_Get(**unsigned** **int** gpioNum);  **void** GPIOIntInit( **unsigned** **long** ulPort,  **unsigned** **char** ucPin,  **unsigned** **long** ulInterrupt,  **void** (\*pfnHandler)(**void**),  **unsigned** **long** ulIntType,  **unsigned** **char** ucPriority);  **void** GPIOs3IntHandler(**void**);  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  // Mark the end of the C bindings section for C++ compilers.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#ifdef** \_\_cplusplus  }  **#endif**  **#endif** // \_\_GPIOIF\_H\_\_ |

|  |
| --- |
| **gpio\_if.c** |
| /\*  \* gpio\_if.c  \*  \* Created on: 2016. 9. 7.  \* Author: Sokwhan  \*/  **#include** <stdio.h>  **#include** "hw\_types.h"  **#include** "hw\_ints.h"  **#include** "hw\_memmap.h"  **#include** "interrupt.h"  **#include** "pin.h"  **#include** "gpio.h"  **#include** "hw\_apps\_rcm.h"  **#include** "prcm.h"  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "gpio\_if.h"  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // GLOBAL VARIABLES  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **unsigned** **long** ulReg[]=  {  GPIOA0\_BASE,  GPIOA1\_BASE,  GPIOA2\_BASE,  GPIOA3\_BASE  };  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // LOCAL FUNCTION DEFINITIONS  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  **GPIO\_IF\_Set**(**unsigned** **int** gpioNum, **unsigned** **int** state) {  **unsigned** **int** portNum = 0;  **unsigned** **char** pinNum;  GPIO\_IF\_GetPortNPin(gpioNum,  &portNum,  &pinNum);  GPIO\_IF\_SetVal(gpioNum, portNum, pinNum, state);  }  **void**  **GPIO\_IF\_Toggle**(**unsigned** **int** gpioNum) {  **unsigned** **int** portNum = 0;  **unsigned** **char** pinNum;  **unsigned** **int** portStatus;  GPIO\_IF\_GetPortNPin(gpioNum,  &portNum,  &pinNum);  portStatus = !GPIO\_IF\_GetVal(gpioNum, portNum, pinNum);  GPIO\_IF\_SetVal(gpioNum, portNum, pinNum, portStatus);  }  **unsigned** **char**  **GPIO\_IF\_Get**(**unsigned** **int** gpioNum) {  **unsigned** **int** portNum = 0;  **unsigned** **char** pinNum;  GPIO\_IF\_GetPortNPin(gpioNum,  &portNum,  &pinNum);  **return** GPIO\_IF\_GetVal(gpioNum, portNum, pinNum);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Get the port and pin of a given GPIO  //!  //! \param ucPin is the pin to be set-up as a GPIO (0:39)  //! \param puiGPIOPort is the pointer to store GPIO port address return value  //! \param pucGPIOPin is the pointer to store GPIO pin return value  //!  //! This function  //! 1. Return the GPIO port address and pin for a given external pin number  //!  //! \return None.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  **GPIO\_IF\_GetPortNPin**(**unsigned** **char** ucPin,  **unsigned** **int** \*puiGPIOPort,  **unsigned** **char** \*pucGPIOPin)  {  //  // Get the GPIO pin from the external Pin number  //  \*pucGPIOPin = 1 << (ucPin % 8);  //  // Get the GPIO port from the external Pin number  //  \*puiGPIOPort = (ucPin / 8);  \*puiGPIOPort = ulReg[\*puiGPIOPort];  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Configures the GPIO selected as input to generate interrupt on activity  //!  //! \param uiGPIOPort is the GPIO port address  //! \param ucGPIOPin is the GPIO pin of the specified port  //! \param uiIntType is the type of the interrupt (refer gpio.h)  //! \param pfnIntHandler is the interrupt handler to register  //!  //! This function  //! 1. Sets GPIO interrupt type  //! 2. Registers Interrupt handler  //! 3. Enables Interrupt  //!  //! \return None  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  **GPIO\_IF\_ConfigureNIntEnable**(**unsigned** **int** uiGPIOPort,  **unsigned** **char** ucGPIOPin,  **unsigned** **int** uiIntType,  **void** (\*pfnIntHandler)(**void**))  {  //  // Set GPIO interrupt type  //  MAP\_GPIOIntTypeSet(uiGPIOPort,ucGPIOPin,uiIntType);  //  // Register Interrupt handler  //  MAP\_GPIOIntRegister(uiGPIOPort,pfnIntHandler);  //  // Enable Interrupt  //  MAP\_GPIOIntClear(uiGPIOPort,ucGPIOPin);  MAP\_GPIOIntEnable(uiGPIOPort,ucGPIOPin);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Set a value to the specified GPIO pin  //!  //! \param ucPin is the GPIO pin to be set (0:39)  //! \param uiGPIOPort is the GPIO port address  //! \param ucGPIOPin is the GPIO pin of the specified port  //! \param ucGPIOValue is the value to be set  //!  //! This function  //! 1. Sets a value to the specified GPIO pin  //!  //! \return None.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void**  **GPIO\_IF\_SetVal**(**unsigned** **char** ucPin,  **unsigned** **int** uiGPIOPort,  **unsigned** **char** ucGPIOPin,  **unsigned** **char** ucGPIOValue)  {  //  // Set the corresponding bit in the bitmask  //  ucGPIOValue = ucGPIOValue << (ucPin % 8);  //  // Invoke the API to set the value  //  MAP\_GPIOPinWrite(uiGPIOPort,ucGPIOPin,ucGPIOValue);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Set a value to the specified GPIO pin  //!  //! \param ucPin is the GPIO pin to be set (0:39)  //! \param uiGPIOPort is the GPIO port address  //! \param ucGPIOPin is the GPIO pin of the specified port  //!  //! This function  //! 1. Gets a value of the specified GPIO pin  //!  //! \return value of the GPIO pin  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **unsigned** **char**  **GPIO\_IF\_GetVal**(**unsigned** **char** ucPin,  **unsigned** **int** uiGPIOPort,  **unsigned** **char** ucGPIOPin)  {  **unsigned** **char** ucGPIOValue;  **long** lGPIOStatus;  //  // Invoke the API to Get the value  //  lGPIOStatus = MAP\_GPIOPinRead(uiGPIOPort,ucGPIOPin);  //  // Set the corresponding bit in the bitmask  //  ucGPIOValue = lGPIOStatus >> (ucPin % 8);  **return** ucGPIOValue;  }  **void** **GPIOIntInit**(**unsigned** **long** ulPort, **unsigned** **char** ucPin, **unsigned** **long** ulInterrupt, **void** (\*pfnHandler)(**void**), **unsigned** **long** ulIntType, **unsigned** **char** ucPriority) {  **IntRegister**(ulInterrupt, pfnHandler);  **IntPrioritySet**(ulInterrupt, INT\_PRIORITY\_LVL\_1);  **GPIOIntTypeSet**(ulPort,ucPin,ulIntType);  **GPIOIntClear**(ulPort,0xFF);  **GPIOIntEnable**(ulPort,ucPin);  **IntEnable**(ulInterrupt);  } |

|  |
| --- |
| **keypad.h** |
| /\*  \* keypad.h  \*  \* Created on: Sep 23, 2016  \* Author: shuh  \*/  **#ifndef** KEYPAD\_H\_  **#define** KEYPAD\_H\_  **#define** MAX\_KEYPAD\_ROWS 4  **typedef** **enum**  {  *NONE* = 0,  *UP\_ARROW* = 1,  *LEFT\_ARROW* = 2,  *DOWN\_ARROW* = 3,  *RIGHT\_ARROW* = 4,  *ENTER* = 5,  *CANCEL* = 6  } buttonEnum;  **extern** buttonEnum **getPressedButton**();  **#endif** /\* KEYPAD\_H\_ \*/ |

|  |
| --- |
| **keypad.c** |
| /\*  \* keypad.c  \*  \* Created on: Sep 23, 2016  \* Author: shuh  \*/  **#include** "keypad.h"  **#include** "gpio\_if.h"  buttonEnum **getPressedButton**() {  **if** (GPIO\_IF\_Get(*PIN\_KEYPAD\_B1*) == 0)  **return** *CANCEL*;  **if** (GPIO\_IF\_Get(*PIN\_KEYPAD\_B2*) == 0)  **return** *ENTER*;  **if** (GPIO\_IF\_Get(*PIN\_KEYPAD\_B3*) == 0)  **return** *DOWN\_ARROW*;  **if** (GPIO\_IF\_Get(*PIN\_KEYPAD\_B4*) == 0)  **return** *UP\_ARROW*;  **return** *NONE*;  } |

|  |
| --- |
| **lcd.h** |
| /\*  \* lcd.h  \*  \* Created on: Sep 23, 2016  \* Author: shuh  \*/  **#ifndef** LCD\_H\_  **#define** LCD\_H\_  **typedef** **enum**  {  LCD\_DISP\_INIT,  LCD\_DISP\_CONNECT\_AP,  LCD\_DISP\_CONNECT\_MQTT,  LCD\_DISP\_ACTIVE,  LCD\_DISP\_REGISTER\_ACTIVE,  LCD\_DISP\_OPENING\_DOOR,  LCD\_DISP\_UNREGISTERED\_PHONE\_TAPPED,  LCD\_DISP\_REGISTERING\_PHONE,  LCD\_DISP\_EXITING\_APP,  LCD\_DISP\_NFC\_DISABLED,  LCD\_DISP\_NO\_PHONE\_REGISTERED,  LCD\_DISP\_FACTORY\_RESET,  LCD\_DISP\_REBOOTING,  LCD\_DISP\_UNREGISTER\_PHONE\_SUCCESS,  LCD\_DISP\_FIRST\_TIME\_SETUP,  LCD\_DISP\_WIFI\_SETUP\_NFC,  LCD\_DISP\_WIFI\_SETUP\_NFC\_CONFIGURING,  LCD\_DISP\_IOT\_DISABLED,  LCD\_DISP\_AP\_CONN\_FAILURE,  LCD\_DISP\_MQTT\_CONN\_FAILURE,  LCD\_DISP\_WIFI\_TEST\_LAN,  LCD\_DISP\_WIFI\_TEST\_MQTT\_BROKER,  LCD\_DISP\_WIFI\_TEST\_PASS  } sdLcdEnum;  **extern** **void** lcdInit();  **extern** **void** lcdClearScreen(**void**);  **extern** **void** lcdPutString(**unsigned** **char**\* str);  **extern** **void** lcdPutChar(**unsigned** **char** lcdChar);  **extern** **void** lcdSetPosition(**unsigned** **int** position);  **extern** **void** SmartDoorlockLCDDisplay(sdLcdEnum lcdEnum);  **#endif** /\* LCD\_H\_ \*/ |

|  |
| --- |
| **lcd.c** |
| /\*  \* lcd.c  \*  \* Created on: Sep 23, 2016  \* Author: shuh  \*/  // Driverlib includes  **#include** "rom\_map.h"  **#include** "hw\_memmap.h"  **#include** "hw\_types.h"  **#include** "prcm.h"  **#include** "simplelink.h"  **#include** "uart.h"  // Common Interface Includes  **#include** "uart\_if.h"  //Project includes  **#include** "sd\_globals.h"  **#include** "lcd.h"  **#include** "s\_flash.h"  **#define** LCD\_LINE1 0  **#define** LCD\_LINE2 0x40  **#define** LCD\_LINE3 0x14  **#define** LCD\_LINE4 0x54  **typedef** **enum**  {  *LCD\_INIT*,  *CLEAR\_SCREEN*  } lcdCommandEnum;  **static** **void** **lcdPutCommand**(lcdCommandEnum cmdType) {  **switch** (cmdType) {  **case** *CLEAR\_SCREEN*:  MAP\_UARTCharPut(UARTA1\_BASE,0xFE);  MAP\_UARTCharPut(UARTA1\_BASE,0x51);  **break**;  **case** *LCD\_INIT*:  MAP\_UARTConfigSetExpClk(UARTA1\_BASE,MAP\_PRCMPeripheralClockGet(CONSOLE\_PERIPH),  9600, (UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |  UART\_CONFIG\_PAR\_NONE));  **osi\_Sleep**(5);  MAP\_UARTCharPut(UARTA1\_BASE,0xFE);  MAP\_UARTCharPut(UARTA1\_BASE,0x41);  **break**;  }  }  **void** **lcdSetPosition**(**unsigned** **int** position) {  **if** (position < 1 || position > 4)  **return**;  MAP\_UARTCharPut(UARTA1\_BASE,0xFE);  MAP\_UARTCharPut(UARTA1\_BASE,0x45);  **switch** (position) {  **case** 1:  MAP\_UARTCharPut(UARTA1\_BASE,LCD\_LINE1);  **break**;  **case** 2:  MAP\_UARTCharPut(UARTA1\_BASE,LCD\_LINE2);  **break**;  **case** 3:  MAP\_UARTCharPut(UARTA1\_BASE,LCD\_LINE3);  **break**;  **case** 4:  MAP\_UARTCharPut(UARTA1\_BASE,LCD\_LINE4);  **break**;  }  }  **void** **lcdClearScreen**(**void**) {  lcdPutCommand(*CLEAR\_SCREEN*);  **osi\_Sleep**(4);  }  **void** **lcdInit**(**void**) {  lcdPutCommand(*LCD\_INIT*);  **osi\_Sleep**(1);  }  **void** **lcdPutChar**(**unsigned** **char** lcdChar) {  MAP\_UARTCharPut(UARTA1\_BASE,lcdChar);  }  **void** **lcdPutString**(**unsigned** **char**\* str) {  **osi\_Sleep**(1);  **do**  {  lcdPutChar(\*str++);  }  **while**(\*str);  }  **void** **SmartDoorlockLCDDisplay**(sdLcdEnum lcdEnum) {  lcdClearScreen();  **switch** (lcdEnum) {  **case** *LCD\_DISP\_INIT*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  lcdPutString("Initializing");  **break**;  **case** *LCD\_DISP\_CONNECT\_AP*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  lcdPutString("Connecting to AP...");  lcdSetPosition(3);  lcdPutString("SSID: ");  lcdPutString((**unsigned** **char**\*)g\_ConfigData.SSID);  **break**;  **case** *LCD\_DISP\_CONNECT\_MQTT*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  lcdPutString("Connecting to");  lcdSetPosition(3);  lcdPutString("MQTT Broker...");  **break**;  **case** *LCD\_DISP\_ACTIVE*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  **switch** (g\_ConfigData.operationMode) {  **case** *OPER\_NFC\_IOT*:  lcdPutString("NFC / IoT Ready");  **break**;  **case** *OPER\_NFC\_ONLY*:  lcdPutString("NFC Ready");  **break**;  **case** *OPER\_IOT\_ONLY*:  lcdPutString("IoT Ready");  **break**;  }  **break**;  **case** *LCD\_DISP\_OPENING\_DOOR*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  lcdPutString("Opening Door...");  **break**;  **case** *LCD\_DISP\_NFC\_DISABLED*:  lcdPutString("NFC is disabled");  lcdSetPosition(2);  lcdPutString("Please enable NFC");  lcdSetPosition(3);  lcdPutString("From Oper. Setup");  **break**;  **case** *LCD\_DISP\_UNREGISTERED\_PHONE\_TAPPED*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  lcdPutString("Unregister Phone");  **break**;  **case** *LCD\_DISP\_EXITING\_APP*:  lcdPutString("Smart Doorlock");  lcdSetPosition(2);  lcdPutString("Exiting App.");  **break**;  **case** *LCD\_DISP\_REGISTERING\_PHONE*:  lcdPutString("Registering phone");  lcdSetPosition(3);  lcdPutString("Please remove phone");  lcdSetPosition(4);  lcdPutString("from the doorlock");  **break**;  **case** *LCD\_DISP\_REGISTER\_ACTIVE*:  lcdPutString("Registering phone");  lcdSetPosition(3);  lcdPutString("Please place phone");  lcdSetPosition(4);  lcdPutString("on the doorlock");  **break**;  **case** *LCD\_DISP\_NO\_PHONE\_REGISTERED*:  lcdPutString("No Phone Registered");  **break**;  **case** *LCD\_DISP\_FACTORY\_RESET*:  lcdPutString("Factory Resetting..");  lcdSetPosition(2);  lcdPutString("Please do not");  lcdSetPosition(3);  lcdPutString("turn off the power");  **break**;  **case** *LCD\_DISP\_REBOOTING*:  lcdPutString("Rebooting..");  lcdSetPosition(2);  lcdPutString("Please do not");  lcdSetPosition(3);  lcdPutString("turn off the power");  **break**;  **case** *LCD\_DISP\_UNREGISTER\_PHONE\_SUCCESS*:  lcdPutString("Phone Unregistered");  **break**;  **case** *LCD\_DISP\_FIRST\_TIME\_SETUP*:  lcdPutString("Welcome to");  lcdSetPosition(2);  lcdPutString("Smart Doorlock");  lcdSetPosition(3);  lcdPutString("We will walk you");  lcdSetPosition(4);  lcdPutString("Through Setup");  **break**;  **case** *LCD\_DISP\_WIFI\_SETUP\_NFC*:  lcdPutString("Wifi Config (NFC)");  lcdSetPosition(2);  lcdPutString("Select AP from");  lcdSetPosition(3);  lcdPutString("Android Application");  lcdSetPosition(4);  lcdPutString("Then tap phone");  **break**;  **case** *LCD\_DISP\_IOT\_DISABLED*:  lcdPutString("IoT is disabled");  lcdSetPosition(2);  lcdPutString("Please enable IoT");  lcdSetPosition(3);  lcdPutString("From Oper. Setup");  **break**;  **case** *LCD\_DISP\_WIFI\_SETUP\_NFC\_CONFIGURING*:  lcdPutString("Configuring Wi-Fi...");  lcdSetPosition(3);  lcdPutString("Please remove phone");  lcdSetPosition(4);  lcdPutString("from the doorlock");  **break**;  **case** *LCD\_DISP\_AP\_CONN\_FAILURE*:  lcdPutString("Failed to connect");  lcdSetPosition(2);  lcdPutString("to Access Point.");  lcdSetPosition(3);  lcdPutString("Please check your");  lcdSetPosition(4);  lcdPutString("Settings");  **break**;  **case** *LCD\_DISP\_MQTT\_CONN\_FAILURE*:  lcdPutString("Connection to MQTT");  lcdSetPosition(2);  lcdPutString("broker failed!");  **break**;  **case** *LCD\_DISP\_WIFI\_TEST\_LAN*:  lcdPutString("Testing Access Point");  lcdSetPosition(2);  lcdPutString("Connectivity...");  **break**;  **case** *LCD\_DISP\_WIFI\_TEST\_MQTT\_BROKER*:  lcdPutString("Ping Testing");  lcdSetPosition(2);  lcdPutString("MQTT Broker...");  **break**;  **case** *LCD\_DISP\_WIFI\_TEST\_PASS*:  lcdPutString("Wi-Fi Testing");  lcdSetPosition(2);  lcdPutString("Successful");  lcdSetPosition(3);  lcdPutString("IoT is Available");  **break**;  }  } |

|  |
| --- |
| **menu.h** |
| /\*  \* menu.h  \*  \* Created on: Oct 31, 2016  \* Author: shuh  \*/  **#ifndef** MENU\_H\_  **#define** MENU\_H\_  **#include** "keypad.h"  **extern** **void** MoveMenu(**int** menuOption);  **extern** **void** MoveConfigMenu(**int** menuOption);  **extern** **void** MenuProcessMain(buttonEnum pressedBtn);  **extern** **void** MenuProcessConfig(buttonEnum pressedBtn);  **extern** **void** MenuProcessConfigInner(buttonEnum pressedBtn);  **#endif** /\* MENU\_H\_ \*/ |

|  |
| --- |
| **menu.c** |
| /\*  \* menu.c  \*  \* Created on: Oct 31, 2016  \* Author: shuh  \*/  // Driverlib includes  **#include** "simplelink.h"  //Standard Library Includes  **#include** <string.h>  **#include** <stdio.h>  //Project Includes  **#include** "s\_flash.h"  **#include** "sd\_globals.h"  **#include** "menu.h"  **#include** "lcd.h"  **#include** "mcu.h"  **#include** "network.h"  **#define** MENU\_COUNT 3  **#define** CONFIG\_MENU\_COUNT 6  **#define** OPER\_COUNT 3  **static** **unsigned** **int** innerMenuOption = 0;  **const** **unsigned** **char** \*menuList[MENU\_COUNT] = {  "Active",  "Configuration",  "Reboot"  };  **const** **unsigned** **char** \*configMenuList[CONFIG\_MENU\_COUNT] = {  "Operation Setup",  "Register Phone",  "Unregister Phone",  "Setup Wifi",  "Test Wifi Conn.",  "Factory Reset"  };  **const** **unsigned** **char** \*menuOperList[OPER\_COUNT] = {  "NFC Only",  "IoT Only",  "NFC / IoT"  };  **void** **MoveMenu**(**int** menuOption) {  lcdClearScreen();  **int** i = 0;  **for** (i = 0; i < MENU\_COUNT; i++) {  lcdSetPosition(i+1);  i == menuOption ? lcdPutChar('>') : lcdPutChar(' ');  lcdPutString((**unsigned** **char**\*)menuList[i]);  }  }  **void** **MoveConfigMenu**(**int** menuOption) {  lcdClearScreen();  **int** i;  **int** menuIdx = 0;  **int** menuCount = 4;  **if** (menuOption >= 4) {  menuIdx = 4;  menuCount = CONFIG\_MENU\_COUNT - 4;  }  **for** (i = 0; i < menuCount; i++) {  lcdSetPosition(i+1);  i == (menuOption % 4) ? lcdPutChar('>') : lcdPutChar(' ');  lcdPutString((**unsigned** **char**\*)configMenuList[menuIdx]);  menuIdx++;  }  }  **void** **MoveUnregisterMenu**(**unsigned** **int** phoneIdx) {  **char** phoneId[19];  sprintf(phoneId,"%u: ", phoneIdx+1);  **char** strippedId[16];  memcpy(strippedId, g\_ConfigData.doorlockPhoneId[phoneIdx], **sizeof**(strippedId));  strncat(phoneId, strippedId, 16);  lcdClearScreen();  lcdPutString((**unsigned** **char**\*)phoneId);  lcdSetPosition(2);  **char** date[16] = "Date: ";  **char** dateStripped[10];  memcpy(dateStripped, g\_ConfigData.doorlockRegDate[phoneIdx], **sizeof**(dateStripped));  strncat(date, dateStripped, 10);  lcdPutString((**unsigned** **char**\*)date);  lcdSetPosition(3);  lcdPutString("[Up/Down]: Select");  lcdSetPosition(4);  lcdPutString("[OK]: Delete");  }  **void** **MoveOperMenu**(**unsigned** **int** operMenu) {  lcdClearScreen();  **switch** (g\_ConfigData.operationMode) {  **case** OPER\_NFC\_IOT:  lcdPutString("Current: NFC/IOT");  **break**;  **case** OPER\_NFC\_ONLY:  lcdPutString("Current: NFC Only");  **break**;  **case** OPER\_IOT\_ONLY:  lcdPutString("Current: IoT Only");  **break**;  **case** OPER\_NOT\_SET:  lcdPutString("Select Operation");  **break**;  }  **int** i;  **for** (i = 0; i < OPER\_COUNT; i++) {  lcdSetPosition(i+2);  i == operMenu ? lcdPutChar('>') : lcdPutChar(' ');  lcdPutString((**unsigned** **char**\*)menuOperList[i]);  }  }  **static** **void** **FactoryReset**() {  SmartDoorlockLCDDisplay(LCD\_DISP\_FACTORY\_RESET);  ManageConfigData(SF\_DELETE\_DATA\_RECORD);  osi\_Sleep(2000);  RebootMCU();  }  **static** **void** **SetOperationMode**() {  g\_ConfigData.operationMode = innerMenuOption;  ManageConfigData(SF\_WRITE\_DATA\_RECORD);  **if** (g\_firstTimeSetup) {  **if** (g\_ConfigData.operationMode == OPER\_NFC\_IOT || g\_ConfigData.operationMode == OPER\_NFC\_ONLY) {  g\_appMode = MODE\_REGISTER\_ACTIVE;  g\_currMenuOption = MENU\_REGISTER\_PHONE;  MenuProcessConfig(ENTER);  }  }  **else** {  SmartDoorlockLCDDisplay(LCD\_DISP\_REBOOTING);  osi\_Sleep(2000);  RebootMCU();  }  }  **static** **void** **UnregisterPhone**(**unsigned** **int** phoneIdx) {  **int** i;  memset(g\_ConfigData.doorlockPhoneId[phoneIdx], 0, **sizeof**(g\_ConfigData.doorlockPhoneId[phoneIdx]));  memset(g\_ConfigData.doorlockRegDate[phoneIdx], 0, **sizeof**(g\_ConfigData.doorlockRegDate[phoneIdx]));  **for** (i = phoneIdx; i < g\_ConfigData.regDoorlockCount-1; i++) {  strncpy(g\_ConfigData.doorlockPhoneId[i], g\_ConfigData.doorlockPhoneId[i+1], 40);  strncpy(g\_ConfigData.doorlockRegDate[i], g\_ConfigData.doorlockRegDate[i+1], 10);  }  g\_ConfigData.regDoorlockCount--;  ManageConfigData(SF\_WRITE\_DATA\_RECORD);  SmartDoorlockLCDDisplay(LCD\_DISP\_UNREGISTER\_PHONE\_SUCCESS);  osi\_Sleep(1000);  }  **void** **MenuProcessMain**(buttonEnum pressedBtn) {  **if** (pressedBtn == UP\_ARROW && g\_currMenuOption > 0) {  g\_currMenuOption--;  MoveMenu(g\_currMenuOption);  }  **else** **if** (pressedBtn == DOWN\_ARROW && g\_currMenuOption < MENU\_COUNT - 1) {  g\_currMenuOption++;  MoveMenu(g\_currMenuOption);  }  **else** **if** (pressedBtn == ENTER) {  **if** (g\_currMenuOption == MENU\_ACTIVE) {  g\_appMode = MODE\_ACTIVE;  SmartDoorlockLCDDisplay(LCD\_DISP\_ACTIVE);  }  **else** **if** (g\_currMenuOption == MENU\_CONFIG) {  g\_appMode = MODE\_CONFIG;  g\_currMenuOption = MENU\_OPERATION\_SETUP;  MoveConfigMenu(g\_currMenuOption);  }  **else** **if** (g\_currMenuOption == MENU\_REBOOT) {  SmartDoorlockLCDDisplay(LCD\_DISP\_EXITING\_APP);  g\_appMode = MODE\_EXIT;  **return**;  }  }  }  **void** **MenuProcessConfig**(buttonEnum pressedBtn) {  **if** (pressedBtn == CANCEL) {  g\_appMode = MODE\_MENU;  g\_currMenuOption = MENU\_CONFIG;  MoveMenu(g\_currMenuOption);  }  **else** **if** (pressedBtn == UP\_ARROW && g\_currMenuOption > 0) {  g\_currMenuOption--;  MoveConfigMenu(g\_currMenuOption);  }  **else** **if** (pressedBtn == DOWN\_ARROW && g\_currMenuOption < CONFIG\_MENU\_COUNT - 1) {  g\_currMenuOption++;  MoveConfigMenu(g\_currMenuOption);  }  **else** **if** (pressedBtn == ENTER) {  **if** (g\_currMenuOption == MENU\_REGISTER\_PHONE) {  **if** (g\_ConfigData.operationMode == OPER\_IOT\_ONLY) {  SmartDoorlockLCDDisplay(LCD\_DISP\_NFC\_DISABLED);  osi\_Sleep(1500);  MoveConfigMenu(g\_currMenuOption);  }  **else** {  g\_appMode = MODE\_REGISTER\_ACTIVE;  SmartDoorlockLCDDisplay(LCD\_DISP\_REGISTER\_ACTIVE);  }  }  **else** **if** (g\_currMenuOption == MENU\_UNREGISTER\_PHONE) {  **if** (g\_ConfigData.regDoorlockCount == 0) {  SmartDoorlockLCDDisplay(LCD\_DISP\_NO\_PHONE\_REGISTERED);  osi\_Sleep(1500);  MoveConfigMenu(g\_currMenuOption);  **return**;  }  innerMenuOption = 0;  g\_appMode = MODE\_UNREGISTER\_PHONE;  MoveUnregisterMenu(innerMenuOption);  }  **else** **if** (g\_currMenuOption == MENU\_WIFI\_TEST) {  **if** (g\_ConfigData.operationMode == OPER\_NFC\_ONLY) {  SmartDoorlockLCDDisplay(LCD\_DISP\_IOT\_DISABLED);  osi\_Sleep(2000);  MoveConfigMenu(g\_currMenuOption);  **return**;  }  g\_appMode = MODE\_WIFI\_TEST;  SmartDoorlockLCDDisplay(LCD\_DISP\_WIFI\_TEST\_LAN);  **long** lRetVal;  lRetVal = CheckLanConnection();  **if** (lRetVal < 0) {  SmartDoorlockLCDDisplay(LCD\_DISP\_AP\_CONN\_FAILURE);  osi\_Sleep(2000);  g\_appMode = MODE\_CONFIG;  **return**;  }  SmartDoorlockLCDDisplay(LCD\_DISP\_WIFI\_TEST\_MQTT\_BROKER);  lRetVal = CheckInternetConnection();  **if** (lRetVal < 0) {  SmartDoorlockLCDDisplay(LCD\_DISP\_MQTT\_CONN\_FAILURE);  osi\_Sleep(2000);  g\_appMode = MODE\_CONFIG;  **return**;  }  SmartDoorlockLCDDisplay(LCD\_DISP\_WIFI\_TEST\_PASS);  osi\_Sleep(2000);  g\_appMode = MODE\_CONFIG;  MoveConfigMenu(g\_currMenuOption);  **return**;  }  **else** **if** (g\_currMenuOption == MENU\_FACTORY\_RESET) {  FactoryReset();  **return**;  }  **else** **if** (g\_currMenuOption == MENU\_OPERATION\_SETUP) {  g\_appMode = MODE\_OPERATION\_SETUP;  **if** (g\_firstTimeSetup)  innerMenuOption = OPER\_NFC\_ONLY;  **else**  innerMenuOption = g\_ConfigData.operationMode;  MoveOperMenu(innerMenuOption);  }  **else** **if** (g\_currMenuOption == MENU\_WIFI\_CONFIG) {  **if** (g\_ConfigData.operationMode == OPER\_NFC\_ONLY) {  SmartDoorlockLCDDisplay(LCD\_DISP\_IOT\_DISABLED);  osi\_Sleep(2000);  MoveConfigMenu(g\_currMenuOption);  }  **else** {  SmartDoorlockLCDDisplay(LCD\_DISP\_WIFI\_SETUP\_NFC);  g\_appMode = MODE\_WIFI\_CONFIG\_NFC;  }  }  }  }  **void** **MenuProcessConfigInner**(buttonEnum pressedBtn) {  **if** (!g\_firstTimeSetup && pressedBtn == CANCEL) {  g\_appMode = MODE\_CONFIG;  MoveConfigMenu(g\_currMenuOption);  **return**;  }  **if** (g\_appMode == MODE\_UNREGISTER\_PHONE) {  **if** (pressedBtn == UP\_ARROW) {  **if** (innerMenuOption > 0) {  innerMenuOption--;  }  MoveUnregisterMenu(innerMenuOption);  }  **else** **if** (pressedBtn == DOWN\_ARROW) {  **if** (innerMenuOption < g\_ConfigData.regDoorlockCount - 1) {  innerMenuOption++;  }  MoveUnregisterMenu(innerMenuOption);  }  **else** **if** (pressedBtn == ENTER) {  UnregisterPhone(innerMenuOption);  **if** (g\_ConfigData.regDoorlockCount > 0) {  innerMenuOption = 0;  MoveUnregisterMenu(innerMenuOption);  }  **else** {  g\_appMode = MODE\_CONFIG;  MoveConfigMenu(g\_currMenuOption);  }  }  **return**;  }  **if** (g\_appMode == MODE\_OPERATION\_SETUP) {  **if** (pressedBtn == UP\_ARROW) {  **if** (innerMenuOption > 0) {  innerMenuOption--;  }  MoveOperMenu(innerMenuOption);  }  **else** **if** (pressedBtn == DOWN\_ARROW) {  **if** (innerMenuOption < OPER\_COUNT - 1) {  innerMenuOption++;  }  MoveOperMenu(innerMenuOption);  }  **else** **if** (pressedBtn == ENTER) {  SetOperationMode();  }  **return**;  }  } |

|  |
| --- |
| **mqtt\_client.h** |
| /\*  \* mqtt\_client.h  \*  \* Created on: Sep 26, 2016  \* Author: shuh  \*/  **#ifndef** MQTT\_CLIENT\_H\_  **#define** MQTT\_CLIENT\_H\_  **typedef** **enum**  {  BROKER\_DISCONNECTION,  DOORLOCK\_OPEN  }events;  **typedef** **struct**  {  **void** \* hndl;  events event;  }event\_msg;  **extern** **int** initMqtt();  **extern** **int** mqttConnect();  **extern** **int** attemptReconnect();  **extern** **void** Mqtt\_ClientExit();  **extern** **void** MqttPublishLockAccess(**unsigned** **char** \*data);  **extern** OsiMsgQ\_t g\_PBQueue;  **#endif** /\* MQTT\_CLIENT\_H\_ \*/ |

|  |
| --- |
| **mqtt\_client.c** |
| /\*  \* mqtt\_client.c  \*  \* Created on: Sep 26, 2016  \* Author: shuh  \*/  // Standard includes  **#include** <stdlib.h>  // Common Interface Includes  **#include** "common.h"  **#include** "sl\_mqtt\_client.h"  **#include** "uart\_if.h"  **#include** "network.h"  // Project Includes  **#include** "mqtt\_client.h"  /\*Operate Lib in MQTT 3.1 mode.\*/  **#define** MQTT\_3\_1\_1 false /\*MQTT 3.1.1 \*/  **#define** MQTT\_3\_1 true /\*MQTT 3.1\*/  **#define** WILL\_TOPIC "Client"  **#define** WILL\_MSG "Client Stopped"  **#define** WILL\_QOS QOS2  **#define** WILL\_RETAIN false  /\*Background receive task priority\*/  **#define** TASK\_PRIORITY 3  **#define** UART\_PRINT Report  /\*Defining Number of topics\*/  **#define** TOPIC\_COUNT 1  /\* Keep Alive Timer value\*/  **#define** KEEP\_ALIVE\_TIMER 25  /\*Retain Flag. Used in publish message. \*/  **#define** RETAIN 1  /\*Defining Broker IP address and port Number\*/  **#define** SERVER\_ADDRESS "54.210.38.182"  //#define SERVER\_ADDRESS "192.168.2.2"  **#define** PORT\_NUMBER 1883  **#define** SERVER\_MODE MQTT\_3\_1  /\*Specifying Receive time out for the Receive task\*/  **#define** RCV\_TIMEOUT 30  /\*Defining QOS levels\*/  **#define** QOS0 0  **#define** QOS1 1  **#define** QOS2 2  /\*Defining Subscription Topic Values\*/  **#define** TOPIC\_DOORLOCK\_CONTROL "/SmartDoorlock/DoorControl"  **#define** TOPIC\_PUB\_LOG "/SmartDoorlock/Log"  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // LOCAL FUNCTION PROTOTYPES  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void**  **Mqtt\_Recv**(**void** \*app\_hndl, **const** **char** \*topstr, **long** top\_len, **const** **void** \*payload,  **long** pay\_len, bool dup,**unsigned** **char** qos, bool retain);  **static** **void** **sl\_MqttEvt**(**void** \*app\_hndl,**long** evt, **const** **void** \*buf,  **unsigned** **long** len);  **static** **void** **sl\_MqttDisconnect**(**void** \*app\_hndl);  **int** **initMqtt**();  /\* library configuration \*/  SlMqttClientLibCfg\_t Mqtt\_Client={  1882,  TASK\_PRIORITY,  30,  true,  (**long**(\*)(**const** **char** \*, ...))UART\_PRINT  };  **typedef** **struct** connection\_config{  SlMqttClientCtxCfg\_t broker\_config;  **void** \*clt\_ctx;  **unsigned** **char** \*client\_id;  **unsigned** **char** \*usr\_name;  **unsigned** **char** \*usr\_pwd;  bool is\_clean;  **unsigned** **int** keep\_alive\_time;  SlMqttClientCbs\_t CallBAcks;  **int** num\_topics;  **char** \*topic[TOPIC\_COUNT];  **unsigned** **char** qos[TOPIC\_COUNT];  SlMqttWill\_t will\_params;  bool is\_connected;  }connect\_config;  /\* connection configuration \*/  connect\_config usr\_connect\_config[] =  {  {  {  {  SL\_MQTT\_NETCONN\_URL,  SERVER\_ADDRESS,  PORT\_NUMBER,  0,  0,  0,  NULL  },  SERVER\_MODE,  true,  },  NULL,  "SmartDoorlock\_HW1",  NULL,  NULL,  true,  KEEP\_ALIVE\_TIMER,  {Mqtt\_Recv, sl\_MqttEvt, sl\_MqttDisconnect},  TOPIC\_COUNT,  {TOPIC\_DOORLOCK\_CONTROL},  {QOS2},  {WILL\_TOPIC,WILL\_MSG,WILL\_QOS,WILL\_RETAIN},  false  }  };  /\*Message Queue\*/  OsiMsgQ\_t g\_PBQueue;  **void** \*app\_hndl = (**void**\*)usr\_connect\_config;  **int** **mqttConnect**() {  **int** iCount = 0;  connect\_config \*local\_con\_conf = (connect\_config \*)app\_hndl;  //create client context  local\_con\_conf[iCount].clt\_ctx =  **sl\_ExtLib\_MqttClientCtxCreate**(&local\_con\_conf[iCount].broker\_config,  &local\_con\_conf[iCount].CallBAcks,  &(local\_con\_conf[iCount]));  //  // Set Client ID  //  **sl\_ExtLib\_MqttClientSet**((**void**\*)local\_con\_conf[iCount].clt\_ctx,  SL\_MQTT\_PARAM\_CLIENT\_ID,  local\_con\_conf[iCount].client\_id,  **strlen**((**char**\*)(local\_con\_conf[iCount].client\_id)));  //  // Set will Params  //  **if**(local\_con\_conf[iCount].will\_params.will\_topic != NULL)  {  **sl\_ExtLib\_MqttClientSet**((**void**\*)local\_con\_conf[iCount].clt\_ctx,  SL\_MQTT\_PARAM\_WILL\_PARAM,  &(local\_con\_conf[iCount].will\_params),  **sizeof**(SlMqttWill\_t));  }  //  // setting username and password  //  **if**(local\_con\_conf[iCount].usr\_name != NULL)  {  **sl\_ExtLib\_MqttClientSet**((**void**\*)local\_con\_conf[iCount].clt\_ctx,  SL\_MQTT\_PARAM\_USER\_NAME,  local\_con\_conf[iCount].usr\_name,  **strlen**((**char**\*)local\_con\_conf[iCount].usr\_name));  **if**(local\_con\_conf[iCount].usr\_pwd != NULL)  {  **sl\_ExtLib\_MqttClientSet**((**void**\*)local\_con\_conf[iCount].clt\_ctx,  SL\_MQTT\_PARAM\_PASS\_WORD,  local\_con\_conf[iCount].usr\_pwd,  **strlen**((**char**\*)local\_con\_conf[iCount].usr\_pwd));  }  }  //  // connectin to the broker  //  **if**((**sl\_ExtLib\_MqttClientConnect**((**void**\*)local\_con\_conf[iCount].clt\_ctx,  local\_con\_conf[iCount].is\_clean,  local\_con\_conf[iCount].keep\_alive\_time) & 0xFF) != 0)  {  UART\_PRINT("\n\rBroker connect fail for conn no. %d \n\r",iCount+1);  //delete the context for this connection  **sl\_ExtLib\_MqttClientCtxDelete**(local\_con\_conf[iCount].clt\_ctx);  **return** -1;  }  **else**  {  UART\_PRINT("\n\rSuccess: conn to Broker no. %d\n\r ", iCount+1);  local\_con\_conf[iCount].is\_connected = true;  }  //  // Subscribe to topics  //  **if**(**sl\_ExtLib\_MqttClientSub**((**void**\*)local\_con\_conf[iCount].clt\_ctx,  local\_con\_conf[iCount].topic,  local\_con\_conf[iCount].qos, TOPIC\_COUNT) < 0)  {  UART\_PRINT("\n\r Subscription Error for conn no. %d\n\r", iCount+1);  UART\_PRINT("Disconnecting from the broker\r\n");  **sl\_ExtLib\_MqttClientDisconnect**(local\_con\_conf[iCount].clt\_ctx);  local\_con\_conf[iCount].is\_connected = false;  //delete the context for this connection  **sl\_ExtLib\_MqttClientCtxDelete**(local\_con\_conf[iCount].clt\_ctx);  **return** -1;  }  **else**  {  **int** iSub;  UART\_PRINT("Client subscribed on following topics:\n\r");  **for**(iSub = 0; iSub < local\_con\_conf[iCount].num\_topics; iSub++)  {  UART\_PRINT("%s\n\r", local\_con\_conf[iCount].topic[iSub]);  }  }  **return** 0;  }  **int** **attemptReconnect**() {  **if**(!IS\_CONNECTED(g\_ulStatus))  {  UART\_PRINT("device has disconnected from AP \n\r");  UART\_PRINT("retry connection to the AP\n\r");  **while**(!(IS\_CONNECTED(g\_ulStatus)) || !(IS\_IP\_ACQUIRED(g\_ulStatus)))  {  **osi\_Sleep**(10);  }  }  **return** mqttConnect();  }  **int** **initMqtt**() {  **long** lRetVal = -1;  lRetVal = **sl\_ExtLib\_MqttClientInit**(&Mqtt\_Client);  **if**(lRetVal != 0)  {  // lib initialization failed  UART\_PRINT("MQTT Client lib initialization failed\n\r");  **return** -1;  }  **return** 0;  }  **void** **MqttPublishLockAccess**(**unsigned** **char** \*data) {  connect\_config \*local\_con\_conf = (connect\_config \*)app\_hndl;  //unsigned char \*data\_sw2={"Push button sw2 is pressed on CC32XX device"};  **sl\_ExtLib\_MqttClientSend**((**void**\*)local\_con\_conf[0].clt\_ctx,//  TOPIC\_PUB\_LOG,data,**strlen**((**char**\*)data),QOS2,RETAIN);  UART\_PRINT("\n\r CC3200 Publishes the following message \n\r");  UART\_PRINT("Topic: %s\n\r",TOPIC\_PUB\_LOG);  UART\_PRINT("Data: %s\n\r",data);  }  **void** **Mqtt\_ClientExit**() {  **sl\_ExtLib\_MqttClientExit**();  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //! Defines Mqtt\_Pub\_Message\_Receive event handler.  //! Client App needs to register this event handler with sl\_ExtLib\_mqtt\_Init  //! API. Background receive task invokes this handler whenever MQTT Client  //! receives a Publish Message from the broker.  //!  //!\param[out] topstr => pointer to topic of the message  //!\param[out] top\_len => topic length  //!\param[out] payload => pointer to payload  //!\param[out] pay\_len => payload length  //!\param[out] retain => Tells whether its a Retained message or not  //!\param[out] dup => Tells whether its a duplicate message or not  //!\param[out] qos => Tells the Qos level  //!  //!\return none  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void**  **Mqtt\_Recv**(**void** \*app\_hndl, **const** **char** \*topstr, **long** top\_len, **const** **void** \*payload,  **long** pay\_len, bool dup,**unsigned** **char** qos, bool retain)  {  **char** \*topic\_str=(**char**\*)**malloc**(top\_len+1);  **memset**(topic\_str,'\0',top\_len+1);  **strncpy**(topic\_str, (**char**\*)topstr, top\_len);  topic\_str[top\_len]='\0';  **char** \*data\_str=(**char**\*)**malloc**(pay\_len+1);  **memset**(data\_str,'\0',pay\_len+1);  **strncpy**(data\_str, (**char**\*)payload, pay\_len);  data\_str[pay\_len]='\0';  **if**(**strncmp**(topic\_str, TOPIC\_DOORLOCK\_CONTROL, top\_len) == 0)  {  event\_msg msg;  msg.hndl = app\_hndl;  msg.event = *DOORLOCK\_OPEN*;  // write message indicating publish message  **osi\_MsgQWrite**(&g\_PBQueue,&msg,OSI\_NO\_WAIT);  }  UART\_PRINT("\n\rPublish Message Received");  UART\_PRINT("\n\rTopic: %s\n\r", topic\_str);  **free**(topic\_str);  UART\_PRINT(" [Qos: %d] ",qos);  **if**(retain)  UART\_PRINT(" [Retained]");  **if**(dup)  UART\_PRINT(" [Duplicate]");  UART\_PRINT("\n\rData is: %s\n\r", data\_str);  **free**(data\_str);  **return**;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //! Defines sl\_MqttEvt event handler.  //! Client App needs to register this event handler with sl\_ExtLib\_mqtt\_Init  //! API. Background receive task invokes this handler whenever MQTT Client  //! receives an ack(whenever user is in non-blocking mode) or encounters an error.  //!  //! param[out] evt => Event that invokes the handler. Event can be of the  //! following types:  //! MQTT\_ACK - Ack Received  //! MQTT\_ERROR - unknown error  //!  //!  //! \param[out] buf => points to buffer  //! \param[out] len => buffer length  //!  //! \return none  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void**  **sl\_MqttEvt**(**void** \*app\_hndl, **long** evt, **const** **void** \*buf,**unsigned** **long** len)  {  **int** i;  **switch**(evt)  {  **case** SL\_MQTT\_CL\_EVT\_PUBACK:  UART\_PRINT("PubAck:\n\r");  UART\_PRINT("%s\n\r",buf);  **break**;  **case** SL\_MQTT\_CL\_EVT\_SUBACK:  UART\_PRINT("\n\rGranted QoS Levels are:\n\r");  **for**(i=0;i<len;i++)  {  UART\_PRINT("QoS %d\n\r",((**unsigned** **char**\*)buf)[i]);  }  **break**;  **case** SL\_MQTT\_CL\_EVT\_UNSUBACK:  UART\_PRINT("UnSub Ack \n\r");  UART\_PRINT("%s\n\r",buf);  **break**;  **default**:  **break**;  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! callback event in case of MQTT disconnection  //!  //! \param app\_hndl is the handle for the disconnected connection  //!  //! return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void**  **sl\_MqttDisconnect**(**void** \*app\_hndl)  {  connect\_config \*local\_con\_conf;  event\_msg msg;  local\_con\_conf = app\_hndl;  msg.hndl = app\_hndl;  msg.event = *BROKER\_DISCONNECTION*;  UART\_PRINT("disconnect from broker %s\r\n",  (local\_con\_conf->broker\_config).server\_info.server\_addr);  local\_con\_conf->is\_connected = false;  //Derive the value of the local\_con\_conf or clt\_ctx from the message  **sl\_ExtLib\_MqttClientCtxDelete**(local\_con\_conf->clt\_ctx);  //  // write message indicating publish message  //  **osi\_MsgQWrite**(&g\_PBQueue,&msg,OSI\_NO\_WAIT);  } |

|  |
| --- |
| **network\_common.c** |
| /\*  \* network\_common.c  \*  \* Created on: 2016. 9. 7.  \* Author: Sokwhan  \*/  **#include** <stdlib.h>  // Simplelink includes  **#include** "simplelink.h"  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SimpleLink Asynchronous Event Handlers -- Start  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function serves as first level handler for HTTP GET/POST tokens  //! It runs under driver context and performs only operation that can run  //! from this context. For operations that can't is sets an indication of  //! received token and preempts the provisioning context.  //!  //! \param pSlHttpServerEvent Pointer indicating http server event  //! \param pSlHttpServerResponse Pointer indicating http server response  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \_SlEventPropogationStatus\_e **sl\_Provisioning\_HttpServerEventHdl**(  SlHttpServerEvent\_t \*apSlHttpServerEvent,  SlHttpServerResponse\_t \*apSlHttpServerResponse)  {  // Unused in this application  **return** *EVENT\_PROPAGATION\_CONTINUE*;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function serves as first level network application events handler.  //! It runs under driver context and performs only operation that can run  //! from this context. For operations that can't is sets an indication of  //! received token and preempts the provisioning context.  //!  //! \param apEventInfo Pointer to the net app event information  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \_SlEventPropogationStatus\_e **sl\_Provisioning\_NetAppEventHdl**(SlNetAppEvent\_t \*apNetAppEvent)  {  // Unused in this application  **return** *EVENT\_PROPAGATION\_CONTINUE*;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function serves as first level WLAN events handler.  //! It runs under driver context and performs only operation that can run  //! from this context. For operations that can't is sets an indication of  //! received token and preempts the provisioning context.  //!  //! \param apEventInfo Pointer to the WLAN event information  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \_SlEventPropogationStatus\_e **sl\_Provisioning\_WlanEventHdl**(SlWlanEvent\_t \*apEventInfo)  {  // Unused in this application  **return** *EVENT\_PROPAGATION\_CONTINUE*;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SimpleLink Asynchronous Event Handlers -- End  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

|  |
| --- |
| **network.h** |
| /\*  \* network.h  \*  \* Created on: 2016. 9. 7.  \* Author: Sokwhan  \*/  **#ifndef** NETWORK\_H\_  **#define** NETWORK\_H\_  **extern** **unsigned** **long** g\_ulStatus;  // Application specific status/error codes  **typedef** **enum**{  // Choosing -0x7D0 to avoid overlap w/ host-driver's error codes  *LAN\_CONNECTION\_FAILED* = -0x7D0,  *INTERNET\_CONNECTION\_FAILED* = *LAN\_CONNECTION\_FAILED* - 1,  *DEVICE\_NOT\_IN\_STATION\_MODE* = *INTERNET\_CONNECTION\_FAILED* - 1,  *STATUS\_CODE\_MAX* = -0xBB8  }e\_AppStatusCodes;  **extern** **int** **ConnectAP**(**const** **char**\* ssidName, **const** **char**\* securityKey);  **extern** **long** **Network\_IF\_DeInitDriver**(**void**);  **extern** **long** **Network\_IF\_DisconnectFromAP**();  **extern** **long** **CheckInternetConnection**();  **extern** **long** **CheckLanConnection**();  **#endif** /\* NETWORK\_H\_ \*/ |

|  |
| --- |
| **network.c** |
| /\* network.c  \*  \* Created on: 2016. 9. 7.  \* Author: Sokwhan  \*/  **#include** <string.h>  // Simplelink includes  **#include** "simplelink.h"  //Driverlib includes  **#include** "hw\_ints.h"  //Common interface includes  **#include** "gpio\_if.h"  **#include** "uart\_if.h"  **#include** "common.h"  //Project includes  **#include** "network.h"  **#define** HOST\_NAME "www.google.com"  //  // Values for below macros shall be modified for setting the 'Ping' properties  //  **#define** PING\_INTERVAL 1000 /\* In msecs \*/  **#define** PING\_TIMEOUT 3000 /\* In msecs \*/  **#define** PING\_PKT\_SIZE 20 /\* In bytes \*/  **#define** NO\_OF\_ATTEMPTS 3  **#define** WLAN\_CONNECT\_TIMEOUT 10000 /\* In msecs \*/  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // GLOBAL VARIABLES -- Start  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **unsigned** **long** g\_ulStatus = 0;//SimpleLink Status  **unsigned** **long** g\_ulPingPacketsRecv = 0; //Number of Ping Packets received  **unsigned** **long** g\_ulGatewayIP = 0; //Network Gateway IP address  **unsigned** **char** g\_ucConnectionSSID[SSID\_LEN\_MAX+1]; //Connection SSID  **unsigned** **char** g\_ucConnectionBSSID[BSSID\_LEN\_MAX]; //Connection BSSID  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // GLOBAL VARIABLES -- End  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // LOCAL FUNCTION PROTOTYPES  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **long** **WlanConnect**();  **void** **WlanStationMode**( **void** \*pvParameters );  **static** **void** **InitializeAppVariables**();  **static** **long** **ConfigureSimpleLinkToDefaultState**();  **long** **Network\_IF\_DeInitDriver**(**void**);  **long** **Network\_IF\_DisconnectFromAP**();  **void** **Network\_IF\_UnsetMCUMachineState**(**char** cStat);  **void** **Network\_IF\_SetMCUMachineState**(**char** cStat);  **unsigned** **long** **Network\_IF\_CurrentMCUState**();  **void** **Network\_IF\_ResetMCUStateMachine**();  **long** **Network\_IF\_GetHostIP**( **char**\* pcHostName,**unsigned** **long** \* pDestinationIP );  **long** **Network\_IF\_IpConfigGet**(**unsigned** **long** \*pulIP, **unsigned** **long** \*pulSubnetMask,  **unsigned** **long** \*pulDefaultGateway, **unsigned** **long** \*pulDNSServer);  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SimpleLink Asynchronous Event Handlers -- Start  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief The Function Handles WLAN Events  //!  //! \param[in] pWlanEvent - Pointer to WLAN Event Info  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **SimpleLinkWlanEventHandler**(SlWlanEvent\_t \*pWlanEvent)  {  **switch**(pWlanEvent->Event)  {  **case** SL\_WLAN\_CONNECT\_EVENT:  {  SET\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_CONNECTION*);  //  // Information about the connected AP (like name, MAC etc) will be  // available in 'slWlanConnectAsyncResponse\_t'-Applications  // can use it if required  //  // slWlanConnectAsyncResponse\_t \*pEventData = NULL;  // pEventData = &pWlanEvent->EventData.STAandP2PModeWlanConnected;  //  // Copy new connection SSID and BSSID to global parameters  **memcpy**(g\_ucConnectionSSID,pWlanEvent->EventData.  STAandP2PModeWlanConnected.ssid\_name,  pWlanEvent->EventData.STAandP2PModeWlanConnected.ssid\_len);  **memcpy**(g\_ucConnectionBSSID,  pWlanEvent->EventData.STAandP2PModeWlanConnected.bssid,  SL\_BSSID\_LENGTH);  UART\_PRINT("[WLAN EVENT] STA Connected to the AP: %s ,"  "BSSID: %x:%x:%x:%x:%x:%x\n\r",  g\_ucConnectionSSID,g\_ucConnectionBSSID[0],  g\_ucConnectionBSSID[1],g\_ucConnectionBSSID[2],  g\_ucConnectionBSSID[3],g\_ucConnectionBSSID[4],  g\_ucConnectionBSSID[5]);  }  **break**;  **case** SL\_WLAN\_DISCONNECT\_EVENT:  {  slWlanConnectAsyncResponse\_t\* pEventData = NULL;  CLR\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_CONNECTION*);  CLR\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_IP\_AQUIRED*);  pEventData = &pWlanEvent->EventData.STAandP2PModeDisconnected;  // If the user has initiated 'Disconnect' request,  //'reason\_code' is SL\_WLAN\_DISCONNECT\_USER\_INITIATED\_DISCONNECTION  **if**(SL\_WLAN\_DISCONNECT\_USER\_INITIATED\_DISCONNECTION == pEventData->reason\_code)  {  UART\_PRINT("[WLAN EVENT]Device disconnected from the AP: %s,"  "BSSID: %x:%x:%x:%x:%x:%x on application's request \n\r",  g\_ucConnectionSSID,g\_ucConnectionBSSID[0],  g\_ucConnectionBSSID[1],g\_ucConnectionBSSID[2],  g\_ucConnectionBSSID[3],g\_ucConnectionBSSID[4],  g\_ucConnectionBSSID[5]);  }  **else**  {  UART\_PRINT("[WLAN ERROR]Device disconnected from the AP AP: %s,"  "BSSID: %x:%x:%x:%x:%x:%x on an ERROR..!! \n\r",  g\_ucConnectionSSID,g\_ucConnectionBSSID[0],  g\_ucConnectionBSSID[1],g\_ucConnectionBSSID[2],  g\_ucConnectionBSSID[3],g\_ucConnectionBSSID[4],  g\_ucConnectionBSSID[5]);  }  **memset**(g\_ucConnectionSSID,0,**sizeof**(g\_ucConnectionSSID));  **memset**(g\_ucConnectionBSSID,0,**sizeof**(g\_ucConnectionBSSID));  }  **break**;  **default**:  {  UART\_PRINT("[WLAN EVENT] Unexpected event [0x%x]\n\r",  pWlanEvent->Event);  }  **break**;  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function handles network events such as IP acquisition, IP  //! leased, IP released etc.  //!  //! \param[in] pNetAppEvent - Pointer to NetApp Event Info  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **SimpleLinkNetAppEventHandler**(SlNetAppEvent\_t \*pNetAppEvent)  {  **switch**(pNetAppEvent->Event)  {  **case** SL\_NETAPP\_IPV4\_IPACQUIRED\_EVENT:  {  SlIpV4AcquiredAsync\_t \*pEventData = NULL;  SET\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_IP\_AQUIRED*);  //Ip Acquired Event Data  pEventData = &pNetAppEvent->EventData.ipAcquiredV4;  //Gateway IP address  g\_ulGatewayIP = pEventData->gateway;  UART\_PRINT("[NETAPP EVENT] IP Acquired: IP=%d.%d.%d.%d , "  "Gateway=%d.%d.%d.%d\n\r",  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,3),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,2),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,1),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.ip,0),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,3),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,2),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,1),  SL\_IPV4\_BYTE(pNetAppEvent->EventData.ipAcquiredV4.gateway,0));  }  **break**;  **default**:  {  UART\_PRINT("[NETAPP EVENT] Unexpected event [0x%x] \n\r",  pNetAppEvent->Event);  }  **break**;  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function handles HTTP server events  //!  //! \param[in] pServerEvent - Contains the relevant event information  //! \param[in] pServerResponse - Should be filled by the user with the  //! relevant response information  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **SimpleLinkHttpServerCallback**(SlHttpServerEvent\_t \*pHttpEvent,  SlHttpServerResponse\_t \*pHttpResponse)  {  // Unused in this application  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function handles General Events  //!  //! \param[in] pDevEvent - Pointer to General Event Info  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **SimpleLinkGeneralEventHandler**(SlDeviceEvent\_t \*pDevEvent)  {  //  // Most of the general errors are not FATAL are are to be handled  // appropriately by the application  //  UART\_PRINT("[GENERAL EVENT] - ID=[%d] Sender=[%d]\n\n",  pDevEvent->EventData.deviceEvent.status,  pDevEvent->EventData.deviceEvent.sender);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! This function handles socket events indication  //!  //! \param[in] pSock - Pointer to Socket Event Info  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **SimpleLinkSockEventHandler**(SlSockEvent\_t \*pSock)  {  **if**(pSock == NULL)  {  **return**;  }  //  // This application doesn't work w/ socket - Events are not expected  //  **switch**( pSock->Event )  {  **case** SL\_SOCKET\_TX\_FAILED\_EVENT:  **switch**( pSock->socketAsyncEvent.SockTxFailData.status)  {  **case** SL\_ECLOSE:  UART\_PRINT("[SOCK ERROR] - close socket (%d) operation "  "failed to transmit all queued packets\n\n",  pSock->socketAsyncEvent.SockTxFailData.sd);  **break**;  **default**:  UART\_PRINT("[SOCK ERROR] - TX FAILED : socket %d , reason "  "(%d) \n\n",  pSock->socketAsyncEvent.SockTxFailData.sd, pSock->socketAsyncEvent.SockTxFailData.status);  **break**;  }  **break**;  **case** SL\_SOCKET\_ASYNC\_EVENT:  **switch**(pSock->socketAsyncEvent.SockAsyncData.type)  {  **case** SSL\_ACCEPT:/\*accept failed due to ssl issue ( tcp pass)\*/  UART\_PRINT("[SOCK ERROR] - close socket (%d) operation"  "accept failed due to ssl issue\n\r",  pSock->socketAsyncEvent.SockAsyncData.sd);  **break**;  **case** RX\_FRAGMENTATION\_TOO\_BIG:  UART\_PRINT("[SOCK ERROR] -close scoket (%d) operation"  "connection less mode, rx packet fragmentation\n\r"  "> 16K, packet is being released",  pSock->socketAsyncEvent.SockAsyncData.sd);  **break**;  **case** OTHER\_SIDE\_CLOSE\_SSL\_DATA\_NOT\_ENCRYPTED:  UART\_PRINT("[SOCK ERROR] -close socket (%d) operation"  "remote side down from secure to unsecure\n\r",  pSock->socketAsyncEvent.SockAsyncData.sd);  **break**;  **default**:  UART\_PRINT("unknown sock async event: %d\n\r",  pSock->socketAsyncEvent.SockAsyncData.type);  }  **break**;  **default**:  UART\_PRINT("[SOCK EVENT] - Unexpected Event [%x0x]\n\n",pSock->Event);  **break**;  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SimpleLink Asynchronous Event Handlers -- End  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function handles ping report events  //!  //! \param[in] pPingReport - Ping report statistics  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void** **SimpleLinkPingReport**(SlPingReport\_t \*pPingReport)  {  SET\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_PING\_DONE*);  g\_ulPingPacketsRecv = pPingReport->PacketsReceived;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief This function initializes the application variables  //!  //! \param None  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **void** **InitializeAppVariables**()  {  g\_ulStatus = 0;  g\_ulPingPacketsRecv = 0;  g\_ulGatewayIP = 0;  **memset**(g\_ucConnectionSSID,0,**sizeof**(g\_ucConnectionSSID));  **memset**(g\_ucConnectionBSSID,0,**sizeof**(g\_ucConnectionBSSID));  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //! \brief This function puts the device in its default state. It:  //! - Set the mode to STATION  //! - Configures connection policy to Auto and AutoSmartConfig  //! - Deletes all the stored profiles  //! - Enables DHCP  //! - Disables Scan policy  //! - Sets Tx power to maximum  //! - Sets power policy to normal  //! - Unregister mDNS services  //! - Remove all filters  //!  //! \param none  //! \return On success, zero is returned. On error, negative is returned  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **long** **ConfigureSimpleLinkToDefaultState**()  {  SlVersionFull ver = {0};  \_WlanRxFilterOperationCommandBuff\_t RxFilterIdMask = {0};  **unsigned** **char** ucVal = 1;  **unsigned** **char** ucConfigOpt = 0;  **unsigned** **char** ucConfigLen = 0;  **unsigned** **char** ucPower = 0;  **long** lRetVal = -1;  **long** lMode = -1;  /\* lMode = sl\_Start(0, 0, 0);  ASSERT\_ON\_ERROR(lMode);\*/  // If the device is not in station-mode, try configuring it in station-mode  **if** (*ROLE\_STA* != lMode)  {  **if** (*ROLE\_AP* == lMode)  {  // If the device is in AP mode, we need to wait for this event  // before doing anything  **while**(!IS\_IP\_ACQUIRED(g\_ulStatus))  {  **#ifndef** SL\_PLATFORM\_MULTI\_THREADED  \_SlNonOsMainLoopTask();  **#endif**  }  }  // Switch to STA role and restart  lRetVal = **sl\_WlanSetMode**(*ROLE\_STA*);  ASSERT\_ON\_ERROR(lRetVal);  lRetVal = **sl\_Stop**(0xFF);  ASSERT\_ON\_ERROR(lRetVal);  lRetVal = **sl\_Start**(0, 0, 0);  ASSERT\_ON\_ERROR(lRetVal);  // Check if the device is in station again  **if** (*ROLE\_STA* != lRetVal)  {  // We don't want to proceed if the device is not coming up in STA-mode  ASSERT\_ON\_ERROR(*DEVICE\_NOT\_IN\_STATION\_MODE*);  }  }  // Get the device's version-information  ucConfigOpt = SL\_DEVICE\_GENERAL\_VERSION;  ucConfigLen = **sizeof**(ver);  lRetVal = **sl\_DevGet**(SL\_DEVICE\_GENERAL\_CONFIGURATION, &ucConfigOpt,  &ucConfigLen, (**unsigned** **char** \*)(&ver));  ASSERT\_ON\_ERROR(lRetVal);  UART\_PRINT("Host Driver Version: %s\n\r",SL\_DRIVER\_VERSION);  UART\_PRINT("Build Version %d.%d.%d.%d.31.%d.%d.%d.%d.%d.%d.%d.%d\n\r",  ver.NwpVersion[0],ver.NwpVersion[1],ver.NwpVersion[2],ver.NwpVersion[3],  ver.ChipFwAndPhyVersion.FwVersion[0],ver.ChipFwAndPhyVersion.FwVersion[1],  ver.ChipFwAndPhyVersion.FwVersion[2],ver.ChipFwAndPhyVersion.FwVersion[3],  ver.ChipFwAndPhyVersion.PhyVersion[0],ver.ChipFwAndPhyVersion.PhyVersion[1],  ver.ChipFwAndPhyVersion.PhyVersion[2],ver.ChipFwAndPhyVersion.PhyVersion[3]);  // Set connection policy to Auto + SmartConfig  // (Device's default connection policy)  lRetVal = **sl\_WlanPolicySet**(SL\_POLICY\_CONNECTION,  SL\_CONNECTION\_POLICY(1, 0, 0, 0, 1), NULL, 0);  ASSERT\_ON\_ERROR(lRetVal);  // Remove all profiles  lRetVal = **sl\_WlanProfileDel**(0xFF);  ASSERT\_ON\_ERROR(lRetVal);  //  // Device in station-mode. Disconnect previous connection if any  // The function returns 0 if 'Disconnected done', negative number if already  // disconnected Wait for 'disconnection' event if 0 is returned, Ignore  // other return-codes  //  lRetVal = **sl\_WlanDisconnect**();  **if**(0 == lRetVal)  {  // Wait  **while**(IS\_CONNECTED(g\_ulStatus))  {  **#ifndef** SL\_PLATFORM\_MULTI\_THREADED  \_SlNonOsMainLoopTask();  **#endif**  }  }  // Enable DHCP client  lRetVal = **sl\_NetCfgSet**(*SL\_IPV4\_STA\_P2P\_CL\_DHCP\_ENABLE*,1,1,&ucVal);  ASSERT\_ON\_ERROR(lRetVal);  // Disable scan  ucConfigOpt = SL\_SCAN\_POLICY(0);  lRetVal = **sl\_WlanPolicySet**(SL\_POLICY\_SCAN , ucConfigOpt, NULL, 0);  ASSERT\_ON\_ERROR(lRetVal);  // Set Tx power level for station mode  // Number between 0-15, as dB offset from max power - 0 will set max power  ucPower = 0;  lRetVal = **sl\_WlanSet**(SL\_WLAN\_CFG\_GENERAL\_PARAM\_ID,  WLAN\_GENERAL\_PARAM\_OPT\_STA\_TX\_POWER, 1, (**unsigned** **char** \*)&ucPower);  ASSERT\_ON\_ERROR(lRetVal);  // Set PM policy to normal  lRetVal = **sl\_WlanPolicySet**(SL\_POLICY\_PM , SL\_NORMAL\_POLICY, NULL, 0);  ASSERT\_ON\_ERROR(lRetVal);  // Unregister mDNS services  lRetVal = **sl\_NetAppMDNSUnRegisterService**(0, 0);  ASSERT\_ON\_ERROR(lRetVal);  // Remove all 64 filters (8\*8)  **memset**(RxFilterIdMask.FilterIdMask, 0xFF, 8);  lRetVal = **sl\_WlanRxFilterSet**(SL\_REMOVE\_RX\_FILTER, (\_u8 \*)&RxFilterIdMask,  **sizeof**(\_WlanRxFilterOperationCommandBuff\_t));  ASSERT\_ON\_ERROR(lRetVal);  lRetVal = **sl\_Stop**(SL\_STOP\_TIMEOUT);  ASSERT\_ON\_ERROR(lRetVal);  InitializeAppVariables();  **return** lRetVal; // Success  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //! \brief This function checks the LAN connection by pinging the AP's gateway  //!  //! \param None  //!  //! \return 0 on success, negative error-code on error  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long** **CheckLanConnection**()  {  SlPingStartCommand\_t pingParams = {0};  SlPingReport\_t pingReport = {0};  **long** lRetVal = -1;  CLR\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_PING\_DONE*);  g\_ulPingPacketsRecv = 0;  // Set the ping parameters  pingParams.PingIntervalTime = PING\_INTERVAL;  pingParams.PingSize = PING\_PKT\_SIZE;  pingParams.PingRequestTimeout = PING\_TIMEOUT;  pingParams.TotalNumberOfAttempts = NO\_OF\_ATTEMPTS;  pingParams.Flags = 0;  pingParams.Ip = g\_ulGatewayIP;  // Check for LAN connection  lRetVal = **sl\_NetAppPingStart**((SlPingStartCommand\_t\*)&pingParams, SL\_AF\_INET,  (SlPingReport\_t\*)&pingReport, SimpleLinkPingReport);  ASSERT\_ON\_ERROR(lRetVal);  // Wait for NetApp Event  **while**(!IS\_PING\_DONE(g\_ulStatus))  {  **#ifndef** SL\_PLATFORM\_MULTI\_THREADED  \_SlNonOsMainLoopTask();  **#endif**  }  **if**(0 == g\_ulPingPacketsRecv)  {  //Problem with LAN connection  ASSERT\_ON\_ERROR(*LAN\_CONNECTION\_FAILED*);  }  // LAN connection is successful  **return** SUCCESS;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //! \brief This function checks the internet connection by pinging  //! the external-host (HOST\_NAME)  //!  //! \param None  //!  //! \return 0 on success, negative error-code on error  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long** **CheckInternetConnection**()  {  SlPingStartCommand\_t pingParams = {0};  SlPingReport\_t pingReport = {0};  **unsigned** **long** ulIpAddr = 0;  **long** lRetVal = -1;  CLR\_STATUS\_BIT(g\_ulStatus, *STATUS\_BIT\_PING\_DONE*);  g\_ulPingPacketsRecv = 0;  // Set the ping parameters  pingParams.PingIntervalTime = PING\_INTERVAL;  pingParams.PingSize = PING\_PKT\_SIZE;  pingParams.PingRequestTimeout = PING\_TIMEOUT;  pingParams.TotalNumberOfAttempts = NO\_OF\_ATTEMPTS;  pingParams.Flags = 0;  pingParams.Ip = g\_ulGatewayIP;  // Get external host IP address  lRetVal = **sl\_NetAppDnsGetHostByName**((**signed** **char**\*)HOST\_NAME, **sizeof**(HOST\_NAME),  &ulIpAddr, SL\_AF\_INET);  ASSERT\_ON\_ERROR(lRetVal);  // Replace the ping address to match HOST\_NAME's IP address  pingParams.Ip = ulIpAddr;  // Try to ping HOST\_NAME  lRetVal = **sl\_NetAppPingStart**((SlPingStartCommand\_t\*)&pingParams, SL\_AF\_INET,  (SlPingReport\_t\*)&pingReport, SimpleLinkPingReport);  ASSERT\_ON\_ERROR(lRetVal);  // Wait  **while**(!IS\_PING\_DONE(g\_ulStatus))  {  // Wait for Ping Event  **#ifndef** SL\_PLATFORM\_MULTI\_THREADED  \_SlNonOsMainLoopTask();  **#endif**  }  **if** (0 == g\_ulPingPacketsRecv)  {  // Problem with internet connection  ASSERT\_ON\_ERROR(*INTERNET\_CONNECTION\_FAILED*);  }  // Internet connection is successful  **return** SUCCESS;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief Connecting to a WLAN Accesspoint  //!  //! This function connects to the required AP (SSID\_NAME) with Security  //! parameters specified in te form of macros at the top of this file  //!  //! \param None  //!  //! \return None  //!  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **long** **WlanConnect**(**const** **char**\* ssidName, **const** **char**\* password)  {  SlSecParams\_t secParams = {0};  **long** lRetVal = 0;  secParams.Key = (**signed** **char**\*)password;  secParams.KeyLen = **strlen**(password);  secParams.Type = SL\_SEC\_TYPE\_WPA;  lRetVal = **sl\_WlanConnect**((**signed** **char**\*)ssidName, **strlen**(ssidName), 0, &secParams, 0);  ASSERT\_ON\_ERROR(lRetVal);  **unsigned** **long** timeout = 0;  // Wait for WLAN Event  **while**((!IS\_CONNECTED(g\_ulStatus)) || (!IS\_IP\_ACQUIRED(g\_ulStatus)))  {  **if** (timeout > WLAN\_CONNECT\_TIMEOUT) {  **return** FAILURE;  }  // Toggle LEDs to Indicate Connection Progress  GPIO\_IF\_Toggle(10);  **osi\_Sleep**(250);  GPIO\_IF\_Toggle(10);  **osi\_Sleep**(250);  timeout += 500;  }  **return** SUCCESS;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief Start simplelink, connect to the ap and run the ping test  //!  //! This function starts the simplelink, connect to the ap and start the ping  //! test on the default gateway for the ap  //!  //! \param[in] pvParameters - Pointer to the list of parameters that  //! can bepassed to the task while creating it  //!  //! \return None  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **int** **ConnectAP**(**const** **char**\* ssidName, **const** **char**\* securityKey)  {  **long** lRetVal = -1;  InitializeAppVariables();  //  // Following function configure the device to default state by cleaning  // the persistent settings stored in NVMEM (viz. connection profiles &  // policies, power policy etc)  //  // Applications may choose to skip this step if the developer is sure  // that the device is in its default state at start of applicaton  //  // Note that all profiles and persistent settings that were done on the  // device will be lost  //  lRetVal = ConfigureSimpleLinkToDefaultState();  **if**(lRetVal < 0)  {  **if** (*DEVICE\_NOT\_IN\_STATION\_MODE* == lRetVal)  {  UART\_PRINT("Failed to configure the device in its default state\n\r");  }  **return** lRetVal;  }  UART\_PRINT("Device is configured in default state \n\r");  //  // Assumption is that the device is configured in station mode already  // and it is in its default state  //  lRetVal = **sl\_Start**(0, 0, 0);  **if** (lRetVal < 0 || *ROLE\_STA* != lRetVal)  {  UART\_PRINT("Failed to start the device \n\r");  **return** lRetVal;  }  UART\_PRINT("Device started as STATION \n\r");  //  //Connecting to WLAN AP  //  lRetVal = WlanConnect(ssidName,securityKey);  **if**(lRetVal < 0)  {  UART\_PRINT("Failed to establish connection w/ an AP \n\r");  **return** lRetVal;  }  UART\_PRINT("Connection established w/ AP and IP is aquired \n\r");  **return** 0;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Network\_IF\_DeInitDriver  //! The function de-initializes a CC3200 device  //!  //! \param None  //!  //! \return On success, zero is returned. On error, other  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long**  **Network\_IF\_DeInitDriver**(**void**)  {  **long** lRetVal = -1;  UART\_PRINT("SL Disconnect...\n\r");  //  // Disconnect from the AP  //  lRetVal = Network\_IF\_DisconnectFromAP();  //  // Stop the simplelink host  //  **sl\_Stop**(SL\_STOP\_TIMEOUT);  //  // Reset the state to uninitialized  //  Network\_IF\_ResetMCUStateMachine();  **return** lRetVal;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Disconnect Disconnects from an Access Point  //!  //! \param none  //!  //! \return 0 disconnected done, other already disconnected  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long** **Network\_IF\_DisconnectFromAP**()  {  **long** lRetVal = 0;  **if** (IS\_CONNECTED(g\_ulStatus))  {  lRetVal = **sl\_WlanDisconnect**();  **if**(0 == lRetVal)  {  // Wait  **while**(IS\_CONNECTED(g\_ulStatus))  {  **#ifndef** SL\_PLATFORM\_MULTI\_THREADED  \_SlNonOsMainLoopTask();  **#else**  **osi\_Sleep**(1);  **#endif**  }  **return** lRetVal;  }  **else**  {  **return** lRetVal;  }  }  **else**  {  **return** lRetVal;  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Network\_IF\_IpConfigGet Get the IP Address of the device.  //!  //! \param pulIP IP Address of Device  //! \param pulSubnetMask Subnetmask of Device  //! \param pulDefaultGateway Default Gateway value  //! \param pulDNSServer DNS Server  //!  //! \return On success, zero is returned. On error, -1 is returned  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long** **Network\_IF\_IpConfigGet**(**unsigned** **long** \*pulIP, **unsigned** **long** \*pulSubnetMask,  **unsigned** **long** \*pulDefaultGateway, **unsigned** **long** \*pulDNSServer)  {  **unsigned** **char** isDhcp;  **unsigned** **char** len = **sizeof**(SlNetCfgIpV4Args\_t);  **long** lRetVal = -1;  SlNetCfgIpV4Args\_t ipV4 = {0};  lRetVal = **sl\_NetCfgGet**(*SL\_IPV4\_STA\_P2P\_CL\_GET\_INFO*,&isDhcp,&len,  (**unsigned** **char** \*)&ipV4);  ASSERT\_ON\_ERROR(lRetVal);  \*pulIP=ipV4.ipV4;  \*pulSubnetMask=ipV4.ipV4Mask;  \*pulDefaultGateway=ipV4.ipV4Gateway;  \*pulDefaultGateway=ipV4.ipV4DnsServer;  **return** lRetVal;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Network\_IF\_GetHostIP  //!  //! \brief This function obtains the server IP address using a DNS lookup  //!  //! \param[in] pcHostName The server hostname  //! \param[out] pDestinationIP This parameter is filled with host IP address.  //!  //! \return On success, +ve value is returned. On error, -ve value is returned  //!  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long** **Network\_IF\_GetHostIP**( **char**\* pcHostName,**unsigned** **long** \* pDestinationIP )  {  **long** lStatus = 0;  lStatus = **sl\_NetAppDnsGetHostByName**((**signed** **char** \*) pcHostName,  **strlen**(pcHostName),  pDestinationIP, SL\_AF\_INET);  ASSERT\_ON\_ERROR(lStatus);  UART\_PRINT("Get Host IP succeeded.\n\rHost: %s IP: %d.%d.%d.%d \n\r\n\r",  pcHostName, SL\_IPV4\_BYTE(\*pDestinationIP,3),  SL\_IPV4\_BYTE(\*pDestinationIP,2),  SL\_IPV4\_BYTE(\*pDestinationIP,1),  SL\_IPV4\_BYTE(\*pDestinationIP,0));  **return** lStatus;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief Reset state from the state machine  //!  //! \param None  //!  //! \return none  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Network\_IF\_ResetMCUStateMachine**()  {  g\_ulStatus = 0;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief Return the current state bits  //!  //! \param None  //!  //! \return none  //!  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **unsigned** **long** **Network\_IF\_CurrentMCUState**()  {  **return** g\_ulStatus;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief sets a state from the state machine  //!  //! \param cStat Status of State Machine defined in e\_StatusBits  //!  //! \return none  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Network\_IF\_SetMCUMachineState**(**char** cStat)  {  SET\_STATUS\_BIT(g\_ulStatus, cStat);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! \brief Unsets a state from the state machine  //!  //! \param cStat Status of State Machine defined in e\_StatusBits  //!  //! \return none  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Network\_IF\_UnsetMCUMachineState**(**char** cStat)  {  CLR\_STATUS\_BIT(g\_ulStatus, cStat);  } |

|  |
| --- |
| **nfc.h** |
| /\*  \* nfc.h  \*  \* Created on: 2016. 10. 12.  \* Author: Sokwhan  \*/  **#ifndef** NFC\_H\_  **#define** NFC\_H\_  **typedef** **enum**  {  *NFC\_NONE*,  *NFC\_INVALID\_PAYLOAD*,  *NFC\_REG\_PHONE*,  *NFC\_UNREGISTERED\_PHONE*,  *NFC\_OPEN\_DOORLOCK*,  *NFC\_WIFI\_CONFIG*  } nfcCmdEnum;  **extern** **void** **NFCInit**();  **extern** nfcCmdEnum **readNFCTag**();  **extern** **char** nfcCmdPayload[100];  **extern** **char** nfcCmdPayload2[100];  **extern** **char** nfcCmdPayload3[100];  **#endif** /\* NFC\_H\_ \*/ |

|  |
| --- |
| **nfc.c** |
| /\*  \* nfc.c  \*  \* Created on: 2016. 10. 12.  \* Author: Sokwhan  \*/  //Driver lib includes  **#include** "rom\_map.h"  **#include** "hw\_memmap.h"  **#include** "hw\_types.h"  **#include** "prcm.h"  **#include** "simplelink.h"  **#include** "uart.h"  **#include** "gpio.h"  **#include** "hw\_ints.h"  **#include** "interrupt.h"  **#include** "timer.h"  //Common interface includes  **#include** "common.h"  **#include** "timer\_if.h"  **#include** "gpio\_if.h"  **#include** "uart\_if.h"  //Project includes  **#include** "nfc.h"  **#include** "trf797x.h"  **#include** "trf7970BoosterPack.h"  **#include** "iso15693.h"  **#include** "iso14443a.h"  **#include** "spi\_l.h"  //===============================================================  /\*\*\*\*\*\*\*\*\*\* GLOBAL VARIABLES TRF7970A \*\*\*\*\*\*\*\*\*\*/  //===============================================================  u08\_t i\_reg = 0x01; // INTERRUPT REGISTER  u08\_t irq\_flag = 0x00;  s08\_t rxtx\_state = 1; // USED FOR TRANSMIT RECEIVE BYTE COUNT  u08\_t stand\_alone\_flag = 1;  **const** **char** \*CMD\_DELIM\_CHAR = "|";  **char** nfcCmdPayload[100];  **char** nfcCmdPayload2[100];  **char** nfcCmdPayload3[100];  **void** NFCInit() {  //Turn off TRF7970A CS  SPI\_TRF\_CS\_OFF;  // GPIO interrupt setting  // TRF7970 IRQ  GPIOIntInit(GPIOA1\_BASE, GPIO\_PIN\_4, INT\_GPIOA1, Trf797xIRQ, GPIO\_RISING\_EDGE, INT\_PRIORITY\_LVL\_1);  // Set Clock Frequency and Modulation  Trf797xCommunicationSetup();  }  **static** nfcCmdEnum parsePayload(**char** \*ndef\_content) {  **char** \*splitStr;  splitStr = strtok (ndef\_content,CMD\_DELIM\_CHAR);  nfcCmdEnum cmd = NFC\_NONE;  **int** idx = 0;  **while** (splitStr != NULL)  {  **if** (idx == 0) {  **if** (strcmp(splitStr,"DOORLOCK\_REGISTRATION") == 0) {  cmd = NFC\_REG\_PHONE;  }  **else** **if** (strcmp(splitStr,"DOORLOCK\_CONTROL") == 0) {  cmd = NFC\_OPEN\_DOORLOCK;  }  **else** **if** (strcmp(splitStr,"WIFI\_SETUP") == 0) {  cmd = NFC\_WIFI\_CONFIG;  }  }  **else** **if** (idx == 1) {  strcpy (nfcCmdPayload, splitStr);  }  **else** **if** (idx == 2) {  strcpy (nfcCmdPayload2, splitStr);  }  **else** **if** (idx == 3) {  strcpy (nfcCmdPayload3, splitStr);  }  splitStr = strtok (NULL,CMD\_DELIM\_CHAR);  idx++;  }  **return** cmd;  }  nfcCmdEnum readNFCTag() {  g\_ndef\_content\_received = 0;  // TRF IRQ disable and clear  IRQ\_OFF;  // TRF disable  TRF\_OFF;  // delay at least 10 ms  osi\_Sleep(100);  // Enter LPM3  TRF\_ON;  // Must wait at least 4.8 mSec to allow TRF7970A to initialize.  osi\_Sleep(5);  //ISO15693FindTag(); // Scan for 15693 tags  ISO14443aFindTag();  **if**(g\_ndef\_content\_received) {  nfcCmdEnum cmd = parsePayload(g\_ndef\_content);  **return** cmd;  }  osi\_Sleep(150);  **return** NFC\_NONE;  } |

|  |
| --- |
| **pinmux.h** |
| /\*  \* Smart Doorlock  \*  \* pinmux.h  \*  \* Created on: 2016. 8. 27.  \* Author: Sokwhan  \*/  **#ifndef** PINMUX\_H\_  **#define** PINMUX\_H\_  **extern** **void** **PinMuxConfig**(**void**);  **#endif** /\* PINMUX\_H\_ \*/ |

|  |
| --- |
| **pinmux.c** |
| /\*  \* Smart Doorlock  \*  \* pinmux.c  \*  \* Created on: 2016. 8. 27.  \* Author: Sokwhan  \*/  **#include** "pinmux.h"  **#include** "hw\_types.h"  **#include** "hw\_memmap.h"  **#include** "hw\_gpio.h"  **#include** "pin.h"  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "gpio.h"  **#include** "prcm.h"  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  **PinMuxConfig**(**void**)  {  //  // Enable Peripheral Clocks  //  MAP\_PRCMPeripheralClkEnable(PRCM\_UARTA0, PRCM\_RUN\_MODE\_CLK);  MAP\_PRCMPeripheralClkEnable(PRCM\_UARTA1, PRCM\_RUN\_MODE\_CLK);  MAP\_PRCMPeripheralClkEnable(PRCM\_GSPI, PRCM\_RUN\_MODE\_CLK);  MAP\_PRCMPeripheralClkEnable(PRCM\_GPIOA0, PRCM\_RUN\_MODE\_CLK);  MAP\_PRCMPeripheralClkEnable(PRCM\_GPIOA1, PRCM\_RUN\_MODE\_CLK);  MAP\_PRCMPeripheralClkEnable(PRCM\_TIMERA2, PRCM\_RUN\_MODE\_CLK);  //  // Configure PIN\_05 for SPI0 GSPI\_CLK  //  MAP\_PinTypeSPI(PIN\_05, PIN\_MODE\_7);  //  // Configure PIN\_06 for SPI0 GSPI\_MISO (DOUT)  //  MAP\_PinTypeSPI(PIN\_06, PIN\_MODE\_7);  //  // Configure PIN\_07 for SPI0 GSPI\_MOSI (DIN)  //  MAP\_PinTypeSPI(PIN\_07, PIN\_MODE\_7);  //  // Configure PIN\_08 for SPI0 GSPI\_CS  //  MAP\_PinTypeSPI(PIN\_08, PIN\_MODE\_7);  //  // Configure PIN\_03 for TRF IRQ Input (GPIO 12)  //  MAP\_PinTypeGPIO(PIN\_03, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA1\_BASE, 0x10, GPIO\_DIR\_MODE\_IN);  //  // Configure PIN\_02 for TRF enable Output (GPIO11, LED D5)  //  MAP\_PinTypeGPIO(PIN\_02, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA1\_BASE, 0x8, GPIO\_DIR\_MODE\_OUT);  //  // Configure PIN\_04 for Solenoid Motor (GPIO Output 13)  //  MAP\_PinTypeGPIO(PIN\_04, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA1\_BASE, 0x20, GPIO\_DIR\_MODE\_OUT);  // GPIO Definitions for keypad  // Pins 58/59/60/61 - GPIO 3,4,5,6  // Configure PIN\_58 for GPIO Input 3  //  MAP\_PinTypeGPIO(PIN\_58, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA0\_BASE, 0x8, GPIO\_DIR\_MODE\_IN);  //  // Configure PIN\_59 for GPIO Input 4  //  MAP\_PinTypeGPIO(PIN\_59, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA0\_BASE, 0x10, GPIO\_DIR\_MODE\_IN);  //  // Configure PIN\_60 for GPIO Input 5  //  MAP\_PinTypeGPIO(PIN\_60, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA0\_BASE, 0x20, GPIO\_DIR\_MODE\_IN);  //  // Configure PIN\_61 for GPIO Input 6  //  MAP\_PinTypeGPIO(PIN\_61, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA0\_BASE, 0x40, GPIO\_DIR\_MODE\_IN);  //  // Configure PIN\_63 for GPIO Output  //  MAP\_PinTypeGPIO(PIN\_63, PIN\_MODE\_0, false);  MAP\_GPIODirModeSet(GPIOA1\_BASE, 0x1, GPIO\_DIR\_MODE\_OUT);  //  // Configure PIN\_55 for UART0 UART0\_TX  //  MAP\_PinTypeUART(PIN\_55, PIN\_MODE\_3);  //  // Configure PIN\_57 for UART0 UART0\_RX  //  MAP\_PinTypeUART(PIN\_57, PIN\_MODE\_3);  //  // Configure PIN\_01 for UART1 UART1\_TX (LCD)  //  MAP\_PinTypeUART(PIN\_01, PIN\_MODE\_7);  } |

|  |
| --- |
| **s\_flash.h** |
| /\*  \* s\_flash.h  \*  \* Created on: Oct 23, 2016  \* Author: shuh  \*/  **#ifndef** S\_FLASH\_H\_  **#define** S\_FLASH\_H\_  /\* Configuration Data Access options \*/  **#define** SF\_DELETE\_DATA\_RECORD (1)  **#define** SF\_CREATE\_DATA\_RECORD (2)  **#define** SF\_WRITE\_DATA\_RECORD (3)  **#define** SF\_READ\_DATA\_RECORD (4)  **#define** SF\_TEST\_DATA\_RECORD (5)  **typedef** **struct**  {  **unsigned** **char** operationMode;  **char** SSID[40];  **char** Password[40];  **int** regDoorlockCount;  **char** doorlockPhoneId[10][40];  **char** doorlockRegDate[10][10];  }ConfigData\_t;  // initialize configuration data  **extern** ConfigData\_t g\_ConfigData;  **extern** **long** ManageConfigData(**unsigned** **char** Mode);  **#endif** /\* S\_FLASH\_H\_ \*/ |

|  |
| --- |
| **s\_flash.c** |
| /\*  \* s\_flash.c  \*  \* Created on: Oct 23, 2016  \* Author: shuh  \*/  **#include** "sd\_globals.h"  **#include** "simplelink.h"  **#include** "s\_flash.h"  **#define** CONFIG\_FILE\_NAME "SD\_Config.txt"  **long** g\_isFileHandle; // Configuration Record File Handler  // initialize configuration data  ConfigData\_t g\_ConfigData =  {  *OPER\_NOT\_SET*, \  "SW\_Private", \  "smartdoorlock", \  0, \  {"Test"}, \  {"01/01/1900"}  };  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Manages the configuration data in the s-flash  //!  //! \param Mode indicates type of requested operation  //!  //! Possible Mode values: SF\_TEST\_DATA\_RECORD, SF\_CREATE\_DATA\_RECORD, SF\_WRITE\_DATA\_RECORD, SF\_READ\_DATA\_RECORD, SF\_DELETE\_DATA\_RECORD  //!  //!  //! \return 0 on success.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **long** **ManageConfigData**(**unsigned** **char** Mode)  {  **long** lRetVal;  **unsigned** **long** ulToken;  **switch** (Mode)  {  **case** SF\_TEST\_DATA\_RECORD:  lRetVal = **sl\_FsOpen**((**unsigned** **char** \*)CONFIG\_FILE\_NAME, FS\_MODE\_OPEN\_READ, &ulToken, &g\_isFileHandle);  **sl\_FsClose**(g\_isFileHandle,0,0,NULL);  **break**;  **case** SF\_CREATE\_DATA\_RECORD:  **sl\_FsOpen**((**unsigned** **char** \*) CONFIG\_FILE\_NAME, FS\_MODE\_OPEN\_CREATE(1024,*\_FS\_FILE\_OPEN\_FLAG\_COMMIT*|*\_FS\_FILE\_PUBLIC\_WRITE*), &ulToken,&g\_isFileHandle);  **sl\_FsWrite**(g\_isFileHandle, 0, (**unsigned** **char** \*) &g\_ConfigData, **sizeof**(g\_ConfigData));  **sl\_FsClose**(g\_isFileHandle,0,0,NULL);  **break**;  **case** SF\_WRITE\_DATA\_RECORD:  **sl\_FsOpen**((**unsigned** **char** \*) CONFIG\_FILE\_NAME, FS\_MODE\_OPEN\_WRITE, &ulToken,&g\_isFileHandle);  **sl\_FsWrite**(g\_isFileHandle, 0, (**unsigned** **char** \*) &g\_ConfigData, **sizeof**(g\_ConfigData));  **sl\_FsClose**(g\_isFileHandle,0,0,NULL);  **break**;  **case** SF\_READ\_DATA\_RECORD:  **sl\_FsOpen**((**unsigned** **char** \*) CONFIG\_FILE\_NAME,FS\_MODE\_OPEN\_READ, &ulToken,&g\_isFileHandle);  **sl\_FsRead**(g\_isFileHandle,0, (**unsigned** **char** \*) &g\_ConfigData, **sizeof**(g\_ConfigData));  **sl\_FsClose**(g\_isFileHandle,0,0,NULL);  **break**;  **case** SF\_DELETE\_DATA\_RECORD:  **sl\_FsDel**((**unsigned** **char** \*) CONFIG\_FILE\_NAME,0);  **break**;  **default**:  lRetVal = -1;  **break**;  }  **return** lRetVal;  } |

|  |
| --- |
| **spi\_l.h** |
| /\*  \* spi.h  \*  \* Created on: Sep 30, 2016  \* Author: shuh  \*/  **#ifndef** SPI\_L\_H\_  **#define** SPI\_L\_H\_  //Common Interface Includes  **#include** "spi.h"  **#include** "gpio\_if.h"  //#define SPI\_IF\_BIT\_RATE 2000000  **#define** SPI\_IF\_BIT\_RATE 2000000  **#define** SPI\_TRF\_CS\_ON MAP\_SPICSEnable(GSPI\_BASE);  **#define** SPI\_TRF\_CS\_OFF MAP\_SPICSDisable(GSPI\_BASE);  **extern** **void** SPIInit(**void**);  **#endif** /\* SPI\_L\_H\_ \*/ |

|  |
| --- |
| **spi\_l.c** |
| /\*  \* spi.c  \*  \* Created on: Sep 30, 2016  \* Author: shuh  \*/  //Driver Lib Includes  **#include** "rom\_map.h"  **#include** "hw\_memmap.h"  **#include** "hw\_types.h"  **#include** "prcm.h"  // Project Includes  **#include** "spi\_l.h"  **void** **SPIInit**(**void**) {  MAP\_PRCMPeripheralReset(PRCM\_GSPI);  // Reset SPI  MAP\_SPIReset(GSPI\_BASE);  // Configure SPI interface  //SPI\_SUB\_MODE\_1 = [POL = 0, PHA = 1]  MAP\_SPIConfigSetExpClk(GSPI\_BASE,MAP\_PRCMPeripheralClockGet(PRCM\_GSPI),  SPI\_IF\_BIT\_RATE,SPI\_MODE\_MASTER,SPI\_SUB\_MODE\_1,  (SPI\_SW\_CTRL\_CS |  SPI\_4PIN\_MODE |  SPI\_TURBO\_OFF |  SPI\_CS\_ACTIVELOW |  SPI\_WL\_8));  // Enable SPI for communication  MAP\_SPIEnable(GSPI\_BASE);  } |

|  |
| --- |
| **timer\_if.h** |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // timer\_if.h  //  // timer interface header file: Prototypes and Macros for timer APIs  //  // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/  //  //  // Redistribution and use in source and binary forms, with or without  // modification, are permitted provided that the following conditions  // are met:  //  // Redistributions of source code must retain the above copyright  // notice, this list of conditions and the following disclaimer.  //  // Redistributions in binary form must reproduce the above copyright  // notice, this list of conditions and the following disclaimer in the  // documentation and/or other materials provided with the  // distribution.  //  // Neither the name of Texas Instruments Incorporated nor the names of  // its contributors may be used to endorse or promote products derived  // from this software without specific prior written permission.  //  // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#ifndef** \_\_TIMER\_IF\_H\_\_  **#define** \_\_TIMER\_IF\_H\_\_  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  // If building with a C++ compiler, make all of the definitions in this header  // have a C binding.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#ifdef** \_\_cplusplus  **extern** "C"  {  **#endif**  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* MACROS \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **#define** PERIODIC\_TEST\_CYCLES 80000000  **#define** PERIODIC\_TEST\_LOOPS 5  **extern** **void** Timer\_IF\_Init( **unsigned** **long** ePeripheralc, **unsigned** **long** ulBase,  **unsigned** **long** ulConfig, **unsigned** **long** ulTimer, **unsigned** **long** ulValue);  **extern** **void** Timer\_IF\_IntSetup(**unsigned** **long** ulBase, **unsigned** **long** ulTimer,  **void** (\*TimerBaseIntHandler)(**void**));  **extern** **void** Timer\_IF\_InterruptClear(**unsigned** **long** ulBase);  **extern** **void** Timer\_IF\_Start(**unsigned** **long** ulBase, **unsigned** **long** ulTimer,  **unsigned** **long** ulValue);  **extern** **void** Timer\_IF\_Stop(**unsigned** **long** ulBase, **unsigned** **long** ulTimer);  **extern** **void** Timer\_IF\_ReLoad(**unsigned** **long** ulBase, **unsigned** **long** ulTimer,  **unsigned** **long** ulValue);  **extern** **unsigned** **int** Timer\_IF\_GetCount(**unsigned** **long** ulBase, **unsigned** **long** ulTimer);  **void** Timer\_IF\_DeInit(**unsigned** **long** ulBase,**unsigned** **long** ulTimer);  **void** A2CounterLoad(**unsigned** **long** ulTimer);  **void** A2CounterEnable(**void**);  **void** A2CounterDisable(**void**);  **void** A2CounterStop(**void**);  **void** A2CounterInit ( **void** (\*pfnHandler)(**void**) );  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  // Mark the end of the C bindings section for C++ compilers.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#ifdef** \_\_cplusplus  }  **#endif**  **#endif** // \_\_TIMER\_IF\_H\_\_ |

|  |
| --- |
| **timer\_if.c** |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // timer\_if.c  //  // timer interface file: contains different interface functions for timer APIs  //  // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/  //  //  // Redistribution and use in source and binary forms, with or without  // modification, are permitted provided that the following conditions  // are met:  //  // Redistributions of source code must retain the above copyright  // notice, this list of conditions and the following disclaimer.  //  // Redistributions in binary form must reproduce the above copyright  // notice, this list of conditions and the following disclaimer in the  // documentation and/or other materials provided with the  // distribution.  //  // Neither the name of Texas Instruments Incorporated nor the names of  // its contributors may be used to endorse or promote products derived  // from this software without specific prior written permission.  //  // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#include** "hw\_types.h"  **#include** "hw\_memmap.h"  **#include** "hw\_ints.h"  **#include** "debug.h"  **#include** "interrupt.h"  **#include** "timer.h"  **#include** "hw\_timer.h"  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "prcm.h"  **#include** "timer\_if.h"  **#ifdef** USE\_TIRTOS  **#include** <stdlib.h>  **#include** "osi.h"  **#endif**  **#ifdef** USE\_TIRTOS  **static** **unsigned** **char**  **GetPeripheralIntNum**(**unsigned** **long** ulBase, **unsigned** **long** ulTimer)  {  **if**(ulTimer == TIMER\_A)  {  **switch**(ulBase)  {  **case** TIMERA0\_BASE:  **return** INT\_TIMERA0A;  **case** TIMERA1\_BASE:  **return** INT\_TIMERA1A;  **case** TIMERA2\_BASE:  **return** INT\_TIMERA2A;  **case** TIMERA3\_BASE:  **return** INT\_TIMERA3A;  **default**:  **return** INT\_TIMERA0A;  }  }  **else** **if**(ulTimer == TIMER\_B)  {  **switch**(ulBase)  {  **case** TIMERA0\_BASE:  **return** INT\_TIMERA0B;  **case** TIMERA1\_BASE:  **return** INT\_TIMERA1B;  **case** TIMERA2\_BASE:  **return** INT\_TIMERA2B;  **case** TIMERA3\_BASE:  **return** INT\_TIMERA3B;  **default**:  **return** INT\_TIMERA0B;  }  }  **else**  {  **return** INT\_TIMERA0A;  }  }  **#endif**  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Initializing the Timer  //!  //! \param ePeripheral is the peripheral which need to be initialized.  //! \param ulBase is the base address for the timer.  //! \param ulConfig is the configuration for the timer.  //! \param ulTimer selects amoung the TIMER\_A or TIMER\_B or TIMER\_BOTH.  //! \param ulValue is the timer prescale value which must be between 0 and  //! 255 (inclusive) for 16/32-bit timers and between 0 and 65535 (inclusive)  //! for 32/64-bit timers.  //! This function  //! 1. Enables and reset the peripheral for the timer.  //! 2. Configures and set the prescale value for the timer.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_Init**( **unsigned** **long** ePeripheral, **unsigned** **long** ulBase, **unsigned**  **long** ulConfig, **unsigned** **long** ulTimer, **unsigned** **long** ulValue)  {  //  // Initialize GPT A0 (in 32 bit mode) as periodic down counter.  //  MAP\_PRCMPeripheralClkEnable(ePeripheral, PRCM\_RUN\_MODE\_CLK);  MAP\_PRCMPeripheralReset(ePeripheral);  MAP\_TimerConfigure(ulBase,ulConfig);  MAP\_TimerPrescaleSet(ulBase,ulTimer,ulValue);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! setting up the timer  //!  //! \param ulBase is the base address for the timer.  //! \param ulTimer selects between the TIMER\_A or TIMER\_B or TIMER\_BOTH.  //! \param TimerBaseIntHandler is the pointer to the function that handles the  //! interrupt for the Timer  //!  //! This function  //! 1. Register the function handler for the timer interrupt.  //! 2. enables the timer interrupt.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_IntSetup**(**unsigned** **long** ulBase, **unsigned** **long** ulTimer,  **void** (\*TimerBaseIntHandler)(**void**))  {  //  // Setup the interrupts for the timer timeouts.  //  **#ifdef** USE\_TIRTOS  **if**(ulTimer == TIMER\_BOTH)  {  **osi\_InterruptRegister**(GetPeripheralIntNum(ulBase, TIMER\_A),  TimerBaseIntHandler, INT\_PRIORITY\_LVL\_1);  **osi\_InterruptRegister**(GetPeripheralIntNum(ulBase, TIMER\_B),  TimerBaseIntHandler, INT\_PRIORITY\_LVL\_1);  }  **else**  {  **osi\_InterruptRegister**(GetPeripheralIntNum(ulBase, ulTimer),  TimerBaseIntHandler, INT\_PRIORITY\_LVL\_1);  }    **#else**  MAP\_TimerIntRegister(ulBase, ulTimer, TimerBaseIntHandler);  **#endif**    **if**(ulTimer == TIMER\_BOTH)  {  MAP\_TimerIntEnable(ulBase, TIMER\_TIMA\_TIMEOUT|TIMER\_TIMB\_TIMEOUT);  }  **else**  {  MAP\_TimerIntEnable(ulBase, ((ulTimer == TIMER\_A) ? TIMER\_TIMA\_TIMEOUT :  TIMER\_TIMB\_TIMEOUT));  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! clears the timer interrupt  //!  //! \param ulBase is the base address for the timer.  //!  //! This function  //! 1. clears the interrupt with given base.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_InterruptClear**(**unsigned** **long** ulBase)  {  **unsigned** **long** ulInts;  ulInts = MAP\_TimerIntStatus(ulBase, true);  //  // Clear the timer interrupt.  //  MAP\_TimerIntClear(ulBase, ulInts);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! starts the timer  //!  //! \param ulBase is the base address for the timer.  //! \param ulTimer selects amoung the TIMER\_A or TIMER\_B or TIMER\_BOTH.  //! \param ulValue is the the number of clock cycles after which the timer will  //! run out and gives the interrupt.  //!  //! This function  //! 1. Load the Timer with the specified value.  //! 2. enables the timer.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_Start**(**unsigned** **long** ulBase, **unsigned** **long** ulTimer,  **unsigned** **long** ulValue)  {  MAP\_TimerLoadSet(ulBase,ulTimer,ulValue);  //  // Enable the GPT  //  MAP\_TimerEnable(ulBase,ulTimer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! disable the timer  //!  //! \param ulBase is the base address for the timer.  //! \param ulTimer selects amoung the TIMER\_A or TIMER\_B or TIMER\_BOTH.  //!  //! This function  //! 1. disables the interupt.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_Stop**(**unsigned** **long** ulBase, **unsigned** **long** ulTimer)  {  //  // Disable the GPT  //  MAP\_TimerDisable(ulBase,ulTimer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! De-Initialize the timer  //!  //! \param uiGPTBaseAddr  //! \param ulTimer  //!  //! This function  //! 1. disable the timer interrupts  //! 2. unregister the timer interrupt  //!  //! \return None.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_DeInit**(**unsigned** **long** ulBase,**unsigned** **long** ulTimer)  {  //  // Disable the timer interrupt  //  MAP\_TimerIntDisable(ulBase,TIMER\_TIMA\_TIMEOUT|TIMER\_TIMB\_TIMEOUT);  //  // Unregister the timer interrupt  //  MAP\_TimerIntUnregister(ulBase,ulTimer);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! starts the timer  //!  //! \param ulBase is the base address for the timer.  //! \param ulTimer selects between the TIMER A and TIMER B.  //! \param ulValue is the the number of clock cycles after which the timer will  //! run out and gives the interrupt.  //!  //! This function  //! 1. Reload the Timer with the specified value.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** **Timer\_IF\_ReLoad**(**unsigned** **long** ulBase, **unsigned** **long** ulTimer,  **unsigned** **long** ulValue)  {  MAP\_TimerLoadSet(ulBase,ulTimer,ulValue);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! starts the timer  //!  //! \param ulBase is the base address for the timer.  //! \param ulTimer selects amoung the TIMER\_A or TIMER\_B or TIMER\_BOTH.  //!  //! This function  //! 1. returns the timer value.  //!  //! \return Timer Value.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **unsigned** **int** **Timer\_IF\_GetCount**(**unsigned** **long** ulBase, **unsigned** **long** ulTimer)  {  **unsigned** **long** ulCounter;  ulCounter = MAP\_TimerValueGet(ulBase, ulTimer);  **return** 0xFFFFFFFF - ulCounter;  }  **void** **A2CounterLoad**(**unsigned** **long** ulTimer)  {  **TimerLoadSet**(TIMERA2\_BASE, TIMER\_A, ulTimer);  }  **void** **A2CounterEnable**(**void**)  {  **TimerEnable**(TIMERA2\_BASE, TIMER\_A);  }  **void** **A2CounterDisable**(**void**)  {  **TimerDisable**(TIMERA2\_BASE, TIMER\_A);  }  **void** **A2CounterStop**(**void**)  {  **TimerIntClear**(TIMERA2\_BASE, TIMER\_A);  }  **void** **A2CounterInit** ( **void** (\*pfnHandler)(**void**) )  {  // MAP\_PRCMPeripheralClkEnable(PRCM\_TIMERA1, PRCM\_RUN\_MODE\_CLK);  //  // MAP\_TimerConfigure(ulBase,ulConfig);  // MAP\_TimerPrescaleSet(ulBase,ulTimer,ulValue);  **PRCMPeripheralReset**(PRCM\_TIMERA2);  **TimerDisable**(TIMERA2\_BASE, TIMER\_A);  **TimerConfigure**(TIMERA2\_BASE,TIMER\_CFG\_ONE\_SHOT);  // TimerPrescaleSet(TIMERA2\_BASE, TIMER\_A, 80);  **TimerIntRegister**(TIMERA2\_BASE, TIMER\_A, pfnHandler);  **TimerIntClear**(TIMERA2\_BASE, TIMER\_A);  **TimerIntEnable**(TIMERA2\_BASE,TIMER\_TIMA\_TIMEOUT);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  // Close the Doxygen group.  //! @}  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

|  |
| --- |
| **uart\_if.h** |
| /\*  \* uart\_if.h  \*  \* Created on: Oct 22, 2016  \* Author: shuh  \*/  **#ifndef** UART\_IF\_H\_  **#define** UART\_IF\_H\_  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* MACROS \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **#define** UART\_BAUD\_RATE 115200  **#define** SYSCLK 80000000  **#define** CONSOLE UARTA0\_BASE  **#define** CONSOLE\_PERIPH PRCM\_UARTA0  //  // Define the size of UART IF buffer for RX  //  **#define** UART\_IF\_BUFFER 64  //  // Define the UART IF buffer  //  **extern** **unsigned** **char** g\_ucUARTBuffer[];  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* FUNCTION PROTOTYPES \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **extern** **void** DispatcherUARTConfigure(**void**);  **extern** **void** DispatcherUartSendPacket(**unsigned** **char** \*inBuff, **unsigned** **short** usLength);  **extern** **int** GetCmd(**char** \*pcBuffer, **unsigned** **int** uiBufLen);  **extern** **void** InitTerm(**void**);  **extern** **void** ClearTerm(**void**);  **extern** **void** Message(**char** \*format);  **extern** **void** Error(**char** \*format,...);  **extern** **int** Report(**char** \*format, ...);  **void** UartPutChar(**unsigned** **char** ch);  **void** UartPutCrlf(**void**);  **extern** **void** UartSendCString(**char** \*format);  **void** UartPutByte(**unsigned** **char** ch);  **void** UartPutByteHex(**unsigned** **char** ch);  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  // Mark the end of the C bindings section for C++ compilers.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#endif** /\* UART\_IF\_H\_ \*/ |

|  |
| --- |
| **uart\_if.c** |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // uart\_if.c  //  // uart interface file: Prototypes and Macros for UARTLogger  //  // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/  //  //  // Redistribution and use in source and binary forms, with or without  // modification, are permitted provided that the following conditions  // are met:  //  // Redistributions of source code must retain the above copyright  // notice, this list of conditions and the following disclaimer.  //  // Redistributions in binary form must reproduce the above copyright  // notice, this list of conditions and the following disclaimer in the  // documentation and/or other materials provided with the  // distribution.  //  // Neither the name of Texas Instruments Incorporated nor the names of  // its contributors may be used to endorse or promote products derived  // from this software without specific prior written permission.  //  // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#include** <stdarg.h>  **#include** <stdlib.h>  **#include** <stdio.h>  **#include** <string.h>  **#include** "osi.h"  **#include** "hw\_types.h"  **#include** "hw\_memmap.h"  **#include** "prcm.h"  **#include** "pin.h"  **#include** "uart.h"  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "uart\_if.h"  /\* BIOS module Headers \*/  **#include** <ti/sysbios/BIOS.h>  **#include** <ti/sysbios/knl/Clock.h>  **#include** <ti/sysbios/knl/Task.h>  **#include** <ti/sysbios/knl/Semaphore.h>  Semaphore\_Struct semStruct;  Semaphore\_Handle semHandle;  OsiLockObj\_t g\_IntNotiSLockObj;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Global variable indicating command is present  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **static** **unsigned** **long** \_\_Errorlog;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Global variable indicating input length  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **unsigned** **int** ilen=1;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Initialization  //!  //! This function  //! 1. Configures the UART to be used.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  InitTerm()  {  **#ifndef** NOTERM  MAP\_UARTConfigSetExpClk(CONSOLE,MAP\_PRCMPeripheralClockGet(CONSOLE\_PERIPH),  UART\_BAUD\_RATE, (UART\_CONFIG\_WLEN\_8 | UART\_CONFIG\_STOP\_ONE |  UART\_CONFIG\_PAR\_NONE));  **#endif**  \_\_Errorlog = 0;  /\* Semaphore\_Params semParams;  Construct a Semaphore object to be use as a resource lock, inital count 0  Semaphore\_Params\_init(&semParams);  Semaphore\_construct(&semStruct, 1, &semParams);  Obtain instance handle  semHandle = Semaphore\_handle(&semStruct);\*/  **if** (osi\_LockObjCreate(&g\_IntNotiSLockObj) < 0) {  Report("Semaphore creation failure!\n\r");  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Outputs a character string to the console  //!  //! \param str is the pointer to the string to be printed  //!  //! This function  //! 1. prints the input string character by character on to the console.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  Message(**char** \*str)  {  **#ifndef** NOTERM  **if**(str != NULL)  {  **while**(\*str!='\0')  {  MAP\_UARTCharPut(CONSOLE,\*str++);  }  }  **#endif**  }  **void**  UartPutChar(**unsigned** **char** ch)  {  MAP\_UARTCharPut(CONSOLE,ch);  }  **void** UartPutByte(**unsigned** **char** ch)  {  **char** str[4];  sprintf ( str, "%x", (ch >> 4) & 0x0F );  MAP\_UARTCharPut(CONSOLE,str[0]);  sprintf ( str, "%x", ch & 0x0F );  MAP\_UARTCharPut(CONSOLE,str[0]);  }  **void** UartPutByteHex(**unsigned** **char** ch)  {  **char** str[10];  **int** i = 0;  sprintf(str, "%x", ch);  **while**(str[i]!='\0')  {  MAP\_UARTCharPut(CONSOLE,str[i]);  i=i+1;  }  }  **void**  UartPutCrlf(**void**){  MAP\_UARTCharPut(CONSOLE,'\r');  MAP\_UARTCharPut(CONSOLE,'\n');  }  **void**  UartSendCString(**char** \*str)  {  **#ifndef** NOTERM  Report(str);  /\*char \*copiedStr;  copiedStr = malloc(strlen(str));  strcpy(copiedStr,str);  Get access to resource  //Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  osi\_SyncObjWait(&g\_IntNotiSyncObj, OSI\_WAIT\_FOREVER);  osi\_SyncObjClear(&g\_IntNotiSyncObj);  if(str != NULL)  {  while(\*copiedStr!='\0')  {  MAP\_UARTCharPut(CONSOLE,\*copiedStr++);  }  }  osi\_SyncObjSignal(&g\_IntNotiSyncObj);  //Semaphore\_post(semHandle);\*/  **#endif**  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Clear the console window  //!  //! This function  //! 1. clears the console window.  //!  //! \return none  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  ClearTerm()  {  Message("\33[2J\r");  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Error Function  //!  //! \param  //!  //! \return none  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void**  Error(**char** \*pcFormat, ...)  {  **#ifndef** NOTERM  **char** cBuf[256];  va\_list list;  va\_start(list,pcFormat);  vsnprintf(cBuf,256,pcFormat,list);  Message(cBuf);  **#endif**  \_\_Errorlog++;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Get the Command string from UART  //!  //! \param pucBuffer is the command store to which command will be populated  //! \param ucBufLen is the length of buffer store available  //!  //! \return Length of the bytes received. -1 if buffer length exceeded.  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **int**  GetCmd(**char** \*pcBuffer, **unsigned** **int** uiBufLen)  {  **char** cChar;  **int** iLen = 0;  //  // Wait to receive a character over UART  //  cChar = MAP\_UARTCharGet(CONSOLE);  //  // Echo the received character  //  MAP\_UARTCharPut(CONSOLE, cChar);  iLen = 0;  //  // Checking the end of Command  //  **while**((cChar != '\r') && (cChar !='\n') )  {  //  // Handling overflow of buffer  //  **if**(iLen >= uiBufLen)  {  **return** -1;  }  //  // Copying Data from UART into a buffer  //  **if**(cChar != '\b')  {  \*(pcBuffer + iLen) = cChar;  iLen++;  }  **else**  {  //  // Deleting last character when you hit backspace  //  **if**(iLen)  {  iLen--;  }  }  //  // Wait to receive a character over UART  //  cChar = MAP\_UARTCharGet(CONSOLE);  //  // Echo the received character  //  MAP\_UARTCharPut(CONSOLE, cChar);  }  \*(pcBuffer + iLen) = '\0';  Report("\n\r");  **return** iLen;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! prints the formatted string on to the console  //!  //! \param format is a pointer to the character string specifying the format in  //! the following arguments need to be interpreted.  //! \param [variable number of] arguments according to the format in the first  //! parameters  //! This function  //! 1. prints the formatted error statement.  //!  //! \return count of characters printed  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **int** Report(**char** \*pcFormat, ...)  {  /\*char \*copiedStr;  copiedStr = malloc(strlen(str));  strcpy(copiedStr,str);  Get access to resource  //Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);  osi\_SyncObjWait(&g\_IntNotiSyncObj, OSI\_WAIT\_FOREVER);  osi\_SyncObjClear(&g\_IntNotiSyncObj);  if(str != NULL)  {  while(\*copiedStr!='\0')  {  MAP\_UARTCharPut(CONSOLE,\*copiedStr++);  }  }  osi\_SyncObjSignal(&g\_IntNotiSyncObj);  //Semaphore\_post(semHandle);\*/  //osi\_LockObjLock(&g\_IntNotiSLockObj, OSI\_WAIT\_FOREVER);  //osi\_SyncObjClear(&g\_IntNotiSyncObj);  **int** iRet = 0;  **#ifndef** NOTERM  **char** \*pcBuff, \*pcTemp;  **int** iSize = 256;  va\_list list;  pcBuff = (**char**\*)malloc(iSize);  **if**(pcBuff == NULL)  {  **return** -1;  }  **while**(1)  {  va\_start(list,pcFormat);  iRet = vsnprintf(pcBuff,iSize,pcFormat,list);  va\_end(list);  **if**(iRet > -1 && iRet < iSize)  {  **break**;  }  **else**  {  iSize\*=2;  **if**((pcTemp=realloc(pcBuff,iSize))==NULL)  {  Message("Could not reallocate memory\n\r");  iRet = -1;  **break**;  }  **else**  {  pcBuff=pcTemp;  }  }  }  Message(pcBuff);  free(pcBuff);  //osi\_LockObjUnlock(&g\_IntNotiSLockObj);  **#endif**  **return** iRet;  } |

|  |
| --- |
| **iso14443a.h** |
| /\*  \* File Name: iso14443.h  \*  \* Description: Headers and Defines for ISO14443A Specific Functions  \*  \* Copyright (C) 2016 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_ISO14443A\_H\_  **#define** \_ISO14443A\_H\_  //================================================================  **#include** "trf797x.h"  **#include** "type\_4\_ndef.h"  //===============================================================  //Polling Defines  **#define** REQA 0x26  **#define** WUPA 0x52  //Anticollision Defines  **#define** SEL\_CASCADE1 0x93  **#define** SEL\_CASCADE2 0x95  **#define** SEL\_CASCADE3 0x97  **#define** NVB\_INIT 0x20  **#define** NVB\_FULL 0x70  **#define** CT 0x88  //RATS Defines  **#define** RATS\_CMD 0xE0  **#define** RATS\_PARAM 0x70  //PPS Defines  **#define** PPSS 0xD0  **#define** PPS0 0x11  **#define** PPS1\_106 0x00  **#define** PPS1\_212 0x05  **#define** PPS1\_424 0x0A  **#define** PPS1\_848 0x0F  **typedef** **enum**  {  ISO14443A\_UID\_UNKNOWN = 0x00,  ISO14443A\_UID\_SINGLE = 0x04, // Four Bytes  ISO14443A\_UID\_DOUBLE = 0x07, // Seven Bytes  ISO14443A\_UID\_TRIPLE = 0x0A // Ten Bytes  }tISO14443A\_UidSize;  **typedef** **enum**  {  CASCADE1,  CASCADE2,  CASCADE3,  UID\_COMPLETE,  UID\_INCOMPLETE  }tISO14443A\_UidStatus;  **typedef** **enum**  {  NO\_COLLISION,  COLLISION,  NO\_RESPONSE,  COLLISION\_ERROR  }tCollisionStatus;  //===============================================================  u08\_t Iso14443a\_TagSelection(u08\_t ui8Command);  tCollisionStatus Iso14443a\_AnticollisionLoop(tISO14443A\_UidStatus sCascade);  u08\_t Iso14443a\_PollingCommand(u08\_t ui8Command);  tCollisionStatus Iso14443a\_AnticollisionCommand(tISO14443A\_UidStatus sCascade, u08\_t ui8NVB, u08\_t \* pui8UID);  u08\_t Iso14443a\_SelectCommand(tISO14443A\_UidStatus sCascade, u08\_t \* pui8UID, **bool** bSendCT);  u08\_t Iso14443a\_Halt(**void**);  u08\_t Iso14443a\_Rats(**void**);  u08\_t Iso14443a\_Pps(**void**);  **bool** Iso14443a\_StoreUid(tISO14443A\_UidStatus sCascade, u08\_t \* pui8UID);  u08\_t Iso14443a\_Type2\_Read4Blocks(u08\_t ui8StartBlock);  **bool** Iso14443a\_Get\_Type4ACompliance(**void**);  **void** Iso14443a\_Set\_RecursionCount(u08\_t ui8RecursionCount);  u08\_t \* Iso14443a\_Get\_Uid(**void**);  tISO14443A\_UidSize Iso14443a\_Get\_UidSize(**void**);  **extern** **void** ISO14443aFindTag();  **extern** u08\_t g\_ndef\_content\_received;  //===============================================================  **#endif** |

|  |
| --- |
| **iso14443a.c** |
| /\*  \* File Name: iso14443a.c  \*  \* Description: ISO14443A Specific Functions  \*  \* Copyright (C) 2016 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#include** "hw\_memmap.h"  **#include** "iso14443a.h"  **#include** "trf7970BoosterPack.h"  **#include** "uart\_if.h"  **#include** "gpio.h"  **#include** "gpio\_if.h"  //===============================================================  // Global Variables  //===============================================================  **extern** u08\_t g\_ui8TrfBuffer[NFC\_FIFO\_SIZE];  **static** **volatile** tTRF797x\_Status g\_sTrfStatus;  **static** tISO14443A\_UidSize g\_sUidSize = *ISO14443A\_UID\_UNKNOWN*;  **static** u08\_t g\_pui8CompleteUid[10] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};  **static** u08\_t g\_pui8PartialUid[5] = {0x00,0x00,0x00,0x00,0x00};  **static** u08\_t g\_ui8UidPos = 0;  **static** u08\_t g\_ui8ValidUidByteCount = 0;  **static** u08\_t g\_ui8ValidUidBitCount = 0;  **static** u08\_t g\_ui8ValidBits;  **static** u08\_t g\_ui8RecursionCount = 0;  **static** u08\_t g\_ui8MaxRecurviseCalls = 5;  **static** u08\_t g\_ui8Iso14443aSAK;  **static** bool g\_bType4ACompliant = false;  **static** u08\_t g\_ui8AtsSupportedBitrates = 0x00; // This is used to store the ATS reply for TA(1) which contains the Tags supported bitrates - needed to determine PPS request parameters.  u08\_t g\_ndef\_content\_received;  //===============================================================  //  // Iso14443a\_TagSelection - Process to detect and select  // ISO14443A/NFC Type 2/4A Tag Platform compliant tags.  //  // \param ui8Command is the Polling command to be issued by  // the TRF797x for ISO14443A tag detection.  //  // This function issues polling command, processes the ATQA, and  // handles sending correct anticollision and selection commands.  //  // When a collision occurs, this function will call the  // anticollision function to handle the collision.  //  // \return ui8Status returns whether or not an ISO14443A  // compliant tag has been successfully selected.  //  //===============================================================  u08\_t **Iso14443a\_TagSelection**(u08\_t ui8Command)  {  u08\_t ui8Index = 0;  u08\_t ui8LoopCount = 0;  u08\_t ui8Status = STATUS\_FAIL;  bool bSendCT = false;  tISO14443A\_UidStatus sUidProgress = *CASCADE1*;  tCollisionStatus sCollisionStatus = *NO\_COLLISION*;  // Clear UID to store new one  **for**(ui8Index = 0; ui8Index < 10; ui8Index++)  {  g\_pui8CompleteUid[ui8Index] = 0x00;  }  // Clear partial UID buffer  **for** (ui8LoopCount = 0; ui8LoopCount < 5; ui8LoopCount++)  {  g\_pui8PartialUid[ui8LoopCount] = 0x00;  }  g\_ui8UidPos = 0; // Reset UID Position Marker  g\_ui8ValidUidByteCount = 0; // Reset Valid Bytes Received Counter  g\_ui8ValidUidBitCount = 0; // Reset Valid Bits Received Counter  g\_ui8Iso14443aSAK = 0; // Reset the SAK  g\_bType4ACompliant = false; // Reset Type 4A Compliance  g\_ui8AtsSupportedBitrates = 0; // Reset the ATS Reply for TA(1)  g\_ui8ValidBits = 0; // Clear Valid Bit global  // Poll for a ISO14443A tag  **if** (Iso14443a\_PollingCommand(ui8Command))  {  **if** (g\_sTrfStatus == *RX\_COMPLETE*)  {  // Check ATQA Response for UID size  **if** ((g\_ui8TrfBuffer[0] & 0xC0) == 0x00)  {  g\_sUidSize = *ISO14443A\_UID\_SINGLE*;  }  **else** **if** ((g\_ui8TrfBuffer[0] & 0xC0) == 0x40)  {  g\_sUidSize = *ISO14443A\_UID\_DOUBLE*;  }  **else** **if** ((g\_ui8TrfBuffer[0] & 0xC0) == 0x80)  {  g\_sUidSize = *ISO14443A\_UID\_TRIPLE*;  }  **else**  {  g\_sUidSize = *ISO14443A\_UID\_UNKNOWN*;  }  }  **else**  {  // Collision occurred, UID size not known  g\_sUidSize = *ISO14443A\_UID\_UNKNOWN*;  }  }  **else**  {  // No response to polling command, exit function  ui8Status = STATUS\_FAIL;  g\_ui8RecursionCount = 0; // Reset the recursion count for the anticollision loops  **return** ui8Status;  }  **while** (sUidProgress != *UID\_COMPLETE*)  {  sCollisionStatus = Iso14443a\_AnticollisionCommand(sUidProgress, NVB\_INIT, &g\_pui8CompleteUid[0]); // Call anticollision loop function  // Process the response  **if** (sCollisionStatus == *NO\_COLLISION*)  {  // Store the UID and keep track if the CT byte needs to be sent  bSendCT = Iso14443a\_StoreUid(sUidProgress,&g\_ui8TrfBuffer[0]);  // Issue Select command  **if** (Iso14443a\_SelectCommand(sUidProgress,&g\_pui8CompleteUid[g\_ui8UidPos],bSendCT)) // Issue the Select Command  {  // If successful, use SAK information to determine if the UID is complete  **if** ((g\_ui8Iso14443aSAK & BIT2) == 0x00)  {  // UID complete, set status to success  ui8Status = STATUS\_SUCCESS;  sUidProgress = *UID\_COMPLETE*;  **if** (g\_sUidSize == *ISO14443A\_UID\_UNKNOWN*)  {  **if** (sUidProgress == *CASCADE1*)  {  g\_sUidSize = *ISO14443A\_UID\_SINGLE*;  }  **else** **if** (sUidProgress == *CASCADE2*)  {  g\_sUidSize = *ISO14443A\_UID\_DOUBLE*;  }  **else** **if** (sUidProgress == *CASCADE3*)  {  g\_sUidSize = *ISO14443A\_UID\_TRIPLE*;  }  }  }  **else**  {  // UID is not Complete, increase cascade level, update UidSize as well  **if** (sUidProgress == *CASCADE1*)  {  sUidProgress = *CASCADE2*;  **if** (g\_sUidSize == *ISO14443A\_UID\_UNKNOWN*)  {  g\_sUidSize = *ISO14443A\_UID\_DOUBLE*;  }  }  **else** **if** (sUidProgress == *CASCADE2*)  {  sUidProgress = *CASCADE3*;  **if** (g\_sUidSize == *ISO14443A\_UID\_UNKNOWN*)  {  g\_sUidSize = *ISO14443A\_UID\_TRIPLE*;  }  }  **else**  {  // Either Cascade was already CASCADE3 or an error occured, so break to hit re-try loop  sUidProgress = *UID\_INCOMPLETE*;  **break**;  }  }  }  **else**  {  // Break to hit the re-try loop  sUidProgress = *UID\_INCOMPLETE*;  **break**;  }  }  **else** **if** (sCollisionStatus == *COLLISION*)  {  Report("COLLISION Occurred\n\r");  // If a collision occurs, call the Anticollision loop to handle tag collisions  sCollisionStatus = Iso14443a\_AnticollisionLoop(sUidProgress);  // Check if the anticollision loop is successful  **if** (sCollisionStatus == *NO\_COLLISION*)  {  // If successful, use SAK information to determine if the UID is complete  **if** ((g\_ui8Iso14443aSAK & BIT2) == 0x00)  {  // UID complete, set status to success  ui8Status = STATUS\_SUCCESS;  sUidProgress = *UID\_COMPLETE*;  }  **else**  {  // UID is not Complete, increase cascade level, update UidSize as well  **if** (sUidProgress == *CASCADE1*)  {  sUidProgress = *CASCADE2*;  }  **else** **if** (sUidProgress == *CASCADE2*)  {  sUidProgress = *CASCADE3*;  }  **else**  {  // Either Cascade was already CASCADE3 or an error occured, so break to hit re-try loop  sUidProgress = *UID\_INCOMPLETE*;  **break**;  }  }  }  **else**  {  **break**;  }  }  **else**  {  // Other error occurred, do not proceed  sUidProgress = *UID\_INCOMPLETE*;  g\_ui8RecursionCount = 0; // Reset the recursion count for the anticollision loops  **return** ui8Status;  }  }  **if** (sUidProgress == *UID\_INCOMPLETE*)  { // Some error occurred, attempt to find the tag again  **if** (g\_ui8RecursionCount < g\_ui8MaxRecurviseCalls)  {  Report("UID Incomplete: Recursively calling TAG find\n\r");  g\_ui8RecursionCount++;  ui8Status = Iso14443a\_TagSelection(ui8Command);  }  **else**  {  g\_ui8RecursionCount = 0; // Reset the recursion count for the anticollision loops  **return** ui8Status;  }  }  // This won't repetively trigger after the recursive call of Iso14443a\_TagSelection since the sUidProgress will not change  **if** (sUidProgress == *UID\_COMPLETE*)  {  **#ifdef** ENABLE\_HOST  // UID Completed  UartPutCrlf();  Report("Anticollison Completed");  UartPutCrlf();  // Output UID to UART Terminal  Report("ISO14443A UID: ");  UartPutChar('[');  **if** (g\_sUidSize == ISO14443A\_UID\_UNKNOWN) // Assume ID is a single if it has not been defined to this point.  {  g\_sUidSize = ISO14443A\_UID\_SINGLE;  }  **for** (ui8LoopCount=0; ui8LoopCount<g\_sUidSize; ui8LoopCount++)  {  UartPutByte(g\_pui8CompleteUid[ui8LoopCount]);  }  UartPutChar(']');  UartPutCrlf();  **#endif**  // Output compliance to ISO14443-4  **if** (g\_ui8Iso14443aSAK & BIT5)  {  **#ifdef** ENABLE\_HOST  Report("Tag is ISO14443-4 Compliant");  UartPutCrlf();  **#endif**  g\_bType4ACompliant = true;  }  **else**  {  **#ifdef** ENABLE\_HOST  Report("Tag is not ISO14443-4 Compliant");  UartPutCrlf();  **#endif**  g\_bType4ACompliant = false;  }  **#ifdef** ENABLE\_HOST  UartPutCrlf();  **#endif**  }  g\_ui8RecursionCount = 0; // Reset the recursion count for the anticollision loops after anticollision is finished  **return** ui8Status;  }  //===============================================================  //  // Iso14443a\_AnticollisionLoop - Issue the polling command for  // ISO14443A compliant tags.  //  // \param sCascade is the current anticollision cascade.  //  // This function handles the ISO14443A anticollision procedures  // including dealing with receiving broken bytes and issuing  // anticollisions based on those broken bytes. It will run until  // a complete UID is received unless an error occurs.  //  // This function uses a recursive call for the anticollision  // process.  //  // \return sStatus returns a status based on what tag response is  // received following the most recently issued Anticollision  // command.  //  //===============================================================  tCollisionStatus **Iso14443a\_AnticollisionLoop**(tISO14443A\_UidStatus sCascade)  {  u08\_t ui8NVB = NVB\_INIT;  u08\_t ui8NVBytes = 0;  u08\_t ui8NVBits = 0;  u08\_t ui8NVBitCount = 0;  u08\_t ui8CollisionPosition = 0;  u08\_t ui8LoopCount = 0;  tCollisionStatus sStatus = *COLLISION*;  // Note: The g\_ui8UidPos will be used differently in this function in that it will track where to place received  // valid UID bytes rather than mark the location of the first byte as it does in all other functions.  // When a full UID is received, the Iso14443a\_StoreUid function will be called which will restore the g\_ui8UidPos  // value to what is expected by the rest of the firmware. If a microcontroller with larger RAM reserves is used,  // a seperate global variable could be used instead.  ui8CollisionPosition = Trf797xGetCollisionPosition(); // Get the collision position information from the TRF driver  ui8NVBytes = (ui8CollisionPosition >> 4) - 2; // This represents the number of known valid bytes of the UID  ui8NVBitCount = ui8CollisionPosition & 0x07; // This represents the number of known valid bits of the UID (can't be more than 8 or else it would be a valid byte)  g\_ui8ValidUidBitCount = ui8NVBitCount; // Set the valid bit count to be equal to the received value from the TRF  // Use the number of bits received to generate the value of the bits received so far  **for**(ui8LoopCount = 0; ui8LoopCount < ui8NVBitCount; ui8LoopCount++)  {  ui8NVBits = (ui8NVBits << 1) + 1; // Store the info for the valid bits  }  **if** (g\_ui8ValidUidByteCount < ui8NVBytes)  {  **if** ((ui8NVBytes-g\_ui8ValidUidByteCount) > 5)  {  sStatus = *COLLISION\_ERROR*;  **return** sStatus;  }  // Store the received bytes of the UID in a storage buffer  **for** (ui8LoopCount = 0; ui8LoopCount < (ui8NVBytes-g\_ui8ValidUidByteCount); ui8LoopCount++)  {  g\_pui8PartialUid[ui8LoopCount+g\_ui8UidPos] |= g\_ui8TrfBuffer[ui8LoopCount];  }  g\_ui8UidPos = ui8LoopCount+g\_ui8UidPos; // Set the UID Position indicator to the next array index  **if** (g\_ui8UidPos > 4)  {  sStatus = *COLLISION\_ERROR*;  **return** sStatus;  }  g\_ui8ValidUidByteCount = ui8NVBytes; // Set the Valid byte count equal to what was received by the TRF  // Store the received bits of the UID in a storage buffer  // "The valid bits shall be part of the UID CLn that was received before a collision occurred  // followed by a (0)b or (1)b, decided by the PCD. A typical implementation adds a (1)b."  **if** (g\_ui8ValidUidBitCount < 7)  {  // Since the valid bits are not at the maximum amount of bits allowed, add the extra bit at the end of the UID  g\_pui8PartialUid[g\_ui8UidPos] = ((g\_ui8TrfBuffer[g\_ui8ValidUidByteCount] & ui8NVBits));  g\_ui8ValidBits = g\_pui8PartialUid[g\_ui8UidPos]; // Save the current valid bits in a variable  // NVB is equivalent to the value received in the TRF797x Collision Position Register plus one for the extra bit added per the standard.  ui8NVB = ui8CollisionPosition+1; // "The PCD shall assign NVB with a value that specifies the number of valid bits of UID CLn."  g\_ui8ValidUidBitCount++; // Increment the valid bit count by one to mark the extra bit which was added per specifications.  }  **else**  {  g\_pui8PartialUid[g\_ui8UidPos] = (g\_ui8TrfBuffer[g\_ui8ValidUidByteCount] & ui8NVBits);  g\_ui8ValidBits = g\_pui8PartialUid[g\_ui8UidPos]; // Save the current valid bits in a variable  // NVB is equivalent to the value received in the TRF797x Collision Position Register plus one for the extra bit added per the standard.  ui8NVB = ui8CollisionPosition; // "The PCD shall assign NVB with a value that specifies the number of valid bits of UID CLn."  }  }  **else**  {  // Update the valid bits based on the current valid bits as well as the newly received valid bits from the tag response  // "The valid bits shall be part of the UID CLn that was received before a collision occurred  // followed by a (0)b or (1)b, decided by the PCD. A typical implementation adds a (1)b."  **if** (g\_ui8ValidUidBitCount < 7)  {  // Since the valid bits are not at the maximum amount of bits allowed, add the extra bit at the end of the UID  g\_pui8PartialUid[g\_ui8UidPos] = (g\_ui8ValidBits | ((g\_ui8TrfBuffer[0] & ui8NVBits) << (g\_ui8ValidUidBitCount-ui8NVBitCount)));  g\_ui8ValidBits = g\_pui8PartialUid[g\_ui8UidPos]; // Save the current valid bits in a variable  ui8NVB = ui8CollisionPosition+1; // "The PCD shall assign NVB with a value that specifies the number of valid bits of UID CLn."  g\_ui8ValidUidBitCount++; // Increment the valid bit count by one to mark the extra bit which was added per specifications.  }  **else**  {  g\_pui8PartialUid[g\_ui8UidPos] = (g\_ui8ValidBits | ((g\_ui8TrfBuffer[0] & ui8NVBits) << (g\_ui8ValidUidBitCount-ui8NVBitCount)));  g\_ui8ValidBits = g\_pui8PartialUid[g\_ui8UidPos]; // Save the current valid bits in a variable  ui8NVB = ui8CollisionPosition; // "The PCD shall assign NVB with a value that specifies the number of valid bits of UID CLn."  }  }  McuDelayMillisecond(1); // Small delay prior to sending out packet.  sStatus = Iso14443a\_AnticollisionCommand(sCascade,ui8NVB,&g\_pui8PartialUid[0]); // Issue anti-collision command with the partial UID  **if** (sStatus == *NO\_COLLISION*)  {  // No Collision means the anticollision command was successful and the remaining bytes were received  g\_pui8PartialUid[g\_ui8UidPos] = g\_pui8PartialUid[g\_ui8UidPos] | g\_ui8TrfBuffer[0]; // Combine broken byte with 1st received byte to finalize the 1st byte of the UID  g\_ui8UidPos++; // Increment the UID Position Indicator  g\_ui8ValidUidBitCount = 0; // Reset the valid bit counter  g\_ui8ValidBits = 0; // Reset the valid bit variable  // Store the other bytes  **for** (ui8LoopCount = 0; ui8LoopCount < (5-g\_ui8UidPos); ui8LoopCount++)  {  g\_pui8PartialUid[ui8LoopCount+g\_ui8UidPos] = g\_ui8TrfBuffer[(ui8LoopCount+1)]; // Store remaining received UID bytes into the partial UID buffer  }  // Issue the Select Command with the fully received UID  **if** (Iso14443a\_SelectCommand(sCascade,&g\_pui8PartialUid[0],false))  {  // Received the SAK from the Select Command (This is stored in a global within the Select function)  // If the UID Size is not known yet  **if** (g\_sUidSize == *ISO14443A\_UID\_UNKNOWN*)  {  // Use SAK information to determine if the UID is complete  **if** ((g\_ui8Iso14443aSAK & BIT2) == 0x00)  {  **if** (sCascade == *CASCADE1*)  {  g\_sUidSize = *ISO14443A\_UID\_SINGLE*;  }  **else** **if** (sCascade == *CASCADE2*)  {  g\_sUidSize = *ISO14443A\_UID\_DOUBLE*;  }  **else** **if** (sCascade == *CASCADE3*)  {  g\_sUidSize = *ISO14443A\_UID\_TRIPLE*;  }  }  **else**  {  // UID is not complete, update UidSize as well  **if** (sCascade == *CASCADE1*)  {  g\_sUidSize = *ISO14443A\_UID\_DOUBLE*;  }  **else** **if** (sCascade == *CASCADE2*)  {  g\_sUidSize = *ISO14443A\_UID\_TRIPLE*;  }  }  }  // Store the partial UID into the global UID buffer  Iso14443a\_StoreUid(sCascade,&g\_pui8PartialUid[0]);  // Clear partial UID buffer in order to handle future collisions  **for** (ui8LoopCount = 0; ui8LoopCount < 5; ui8LoopCount++)  {  g\_pui8PartialUid[ui8LoopCount] = 0x00;  }  // Set status to NO\_COLLISION and exit function  sStatus = *NO\_COLLISION*;  **return** sStatus;  }  **else**  {  sStatus = *COLLISION\_ERROR*;  }  }  **else** **if** (sStatus == *COLLISION*)  {  // If a collision occurred, check the Recursion Counter and then call Anticollision Loop function again if the condition is met  **if** (g\_ui8RecursionCount < g\_ui8MaxRecurviseCalls)  {  g\_ui8RecursionCount++; // Increment Recursion Counter  sStatus = Iso14443a\_AnticollisionLoop(sCascade); // Recursive call of Anticollision Loop  }  **else**  {  sStatus = *COLLISION\_ERROR*;  }  }  **else**  {  // For all other statuses, return the status that was received  }  **return** sStatus;  }  //===============================================================  //  // Iso14443a\_PollingCommand - Issue the polling command for  // ISO14443A compliant tags.  //  // \param ui8Command is the polling command to be issued.  //  // This function sends the Polling command based on the inputted  // command (either REQA or WUPA).  //  // \return ui8Status returns whether or not an ISO14443A  // compliant tag has responded to the Polling command.  //  //===============================================================  u08\_t **Iso14443a\_PollingCommand**(u08\_t ui8Command)  {  u08\_t ui8Offset = 0;  u08\_t ui8Status = STATUS\_FAIL;  **if** (Trf797xGetIsoControlValue() != 0x88)  {  // Trf797x has not been properly configured for ISO14443A  Trf797xWriteIsoControl(0x88); // Configure the TRF797x for ISO14443A @ 106kbps and Receive without CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x90; // Send without CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x0F; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = ui8Command; // Send the polling command from function input - either REQA (0x26) or WUPA (0x52)  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the ISO14443A Polling Command  Trf797xIrqWaitTimeout(3,30); // 3 millisecond TX timeout, 10 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  //Report("Querying Iso14443a\_PollingCommand status\n\r");  //StatusPrint(g\_sTrfStatus);  **if** (g\_sTrfStatus == *RX\_COMPLETE*) // Tag detected - could be either a single or collided tag  {  ui8Status = STATUS\_SUCCESS;  }  **else** **if** (g\_sTrfStatus == *PROTOCOL\_ERROR*)  {  ui8Status = STATUS\_SUCCESS; // "A PCD detecting a collision in any bit of (b16 to b1) shall commence with the first step of the anticollision loop."  }  **return** ui8Status;  }  //===============================================================  //  // Iso14443a\_AnticollisionCommand - Issue Anticollsion command  // for ISO14443A compliant tags.  //  // \param sCascade is the current anticollision cascade  // \param ui8NVB is the NVB value for the UID bytes and bits  // \param pui8UID is the location of the UID bytes/bits to send.  //  // This function sends the Anticollision command based on the  // current cascade level.  //  // \return sStatus returns a status based on what tag response is  // received following the Anticollision command.  //  //===============================================================  tCollisionStatus **Iso14443a\_AnticollisionCommand**(tISO14443A\_UidStatus sCascade, u08\_t ui8NVB, u08\_t \* pui8UID)  {  u08\_t ui8Offset = 0;  u08\_t ui8LoopCount = 0;  u08\_t ui8UidLength = 0;  u08\_t ui8RxLength = 0;  u08\_t ui8Select = SEL\_CASCADE1;  tCollisionStatus sStatus = *NO\_COLLISION*;  **if** (sCascade == *CASCADE1*)  {  ui8Select = SEL\_CASCADE1;  }  **else** **if** (sCascade == *CASCADE2*)  {  ui8Select = SEL\_CASCADE2;  }  **else** **if** (sCascade == *CASCADE3*)  {  ui8Select = SEL\_CASCADE3;  }  **else**  {  **return** sStatus = *COLLISION\_ERROR*;  }  **if** (Trf797xGetIsoControlValue() != 0x88)  {  // Trf797x has not been properly configured for ISO14443A with no RX CRC  Trf797xWriteIsoControl(0x88); // Configure the TRF797x for ISO14443A @ 106kbps and Receive without CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x90; // Transmit without CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  **if**((ui8NVB & 0x07) != 0x00) // Length of packet in bytes - lower nibble and broken bits of transmit byte length  {  g\_ui8TrfBuffer[ui8Offset++] = (ui8NVB & 0xF0) | (((ui8NVB & 0x07) << 1) + 1); // Set the number of broken bits, last bit is 1 means broken byte  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = ui8NVB & 0xF0; // No broken bits  }  g\_ui8TrfBuffer[ui8Offset++] = ui8Select; // Select Command; can be 0x93, 0x95 or 0x97  g\_ui8TrfBuffer[ui8Offset++] = ui8NVB; // Number of valid bits  ui8UidLength = (ui8NVB >> 4) - 2;  **if** ((ui8NVB & 0x0F) != 0x00)  {  ui8UidLength++;  }  **for** (ui8LoopCount = 0; ui8LoopCount < ui8UidLength; ui8LoopCount++)  {  g\_ui8TrfBuffer[ui8Offset++] = pui8UID[ui8LoopCount]; // UID Bytes  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Select Command  Trf797xIrqWaitTimeout(25,50); // 5 millisecond TX timeout, 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == *RX\_COMPLETE*)  {  ui8RxLength = Trf797xGetRxBytesReceived();  **if** (ui8RxLength > 1)  {  sStatus = *NO\_COLLISION*;  }  **else**  {  sStatus = *COLLISION\_ERROR*;  }  }  **else** **if** (g\_sTrfStatus == *PROTOCOL\_ERROR*)  {  sStatus = *COLLISION*;  }  **else** **if** (g\_sTrfStatus == *NO\_RESPONSE\_RECEIVED*)  {  sStatus = *NO\_RESPONSE*;  }  **else**  {  // Do nothing  sStatus = *COLLISION\_ERROR*;  }  **return** sStatus;  }  //===============================================================  //  // Iso14443a\_SelectCommand - Issue Select command for ISO14443A  // compliant tags.  //  // \param sCascade is the current anticollision cascade.  // \param pui8UID is the location of the UID bytes to send.  // \param bSendCT determines if the CT byte must be sent.  //  // This function issues the Select command based on the current  // cascade level.  //  // \return ui8Status returns whether or not an ISO14443A  // compliant tag has responded to the Select command.  //  //===============================================================  u08\_t **Iso14443a\_SelectCommand**(tISO14443A\_UidStatus sCascade, u08\_t \* pui8UID, bool bSendCT)  {  u08\_t ui8Offset = 0;  u08\_t ui8Status = STATUS\_FAIL;  u08\_t ui8Select = SEL\_CASCADE1;  // Sending select command and will receive a SAK response which has a CRC.  **if** (Trf797xGetIsoControlValue() != 0x08)  {  // Trf797x has not been properly configured for ISO14443A with RX CRC  Trf797xWriteIsoControl(0x08); // Configure the TRF797x for ISO14443A @ 106kbps and Receive with CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(10);  }  **if** (sCascade == *CASCADE1*)  {  ui8Select = SEL\_CASCADE1;  }  **else** **if** (sCascade == *CASCADE2*)  {  ui8Select = SEL\_CASCADE2;  }  **else** **if** (sCascade == *CASCADE3*)  {  ui8Select = SEL\_CASCADE3;  }  **else**  {  **return** ui8Status = STATUS\_FAIL;  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Transmit with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x70; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = ui8Select; // Select Command; can be 0x93, 0x95 or 0x97  g\_ui8TrfBuffer[ui8Offset++] = NVB\_FULL; // Number of valid bits  **if** (bSendCT == true)  {  g\_ui8TrfBuffer[ui8Offset++] = 0x88; // CT  g\_ui8TrfBuffer[ui8Offset++] = \*pui8UID; // UID Bytes  g\_ui8TrfBuffer[ui8Offset++] = \*(pui8UID + 1);  g\_ui8TrfBuffer[ui8Offset++] = \*(pui8UID + 2);  g\_ui8TrfBuffer[ui8Offset++] = ( 0x88 ^ \*pui8UID ^ \*(pui8UID + 1) ^ \*(pui8UID + 2) ); // Calculate BCC Byte  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = \*pui8UID; // UID Bytes  g\_ui8TrfBuffer[ui8Offset++] = \*(pui8UID + 1);  g\_ui8TrfBuffer[ui8Offset++] = \*(pui8UID + 2);  g\_ui8TrfBuffer[ui8Offset++] = \*(pui8UID + 3);  g\_ui8TrfBuffer[ui8Offset++] = ( \*pui8UID ^ \*(pui8UID + 1) ^ \*(pui8UID + 2) ^ \*(pui8UID + 3) ); // Calculate BCC Byte  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Select Command  Trf797xIrqWaitTimeout(20,80); // 5 millisecond TX timeout, 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  //Report("Iso14443a\_SelectCommand trfStatus: ");  //StatusPrint(g\_sTrfStatus);  **if** (g\_sTrfStatus == *RX\_COMPLETE*)  {  ui8Status = STATUS\_SUCCESS;  g\_ui8Iso14443aSAK = g\_ui8TrfBuffer[0];  }  **else**  {  // Do nothing  }  **return** ui8Status;  }  //===============================================================  //  // Iso14443a\_Halt - Issue the Halt command to the currently  // selected ISO14443A compliant tag.  //  // This function sends the Polling command based on the inputted  // command (either REQA or WUPA).  //  // \return ui8Status returns STATUS\_FAIL if the tag erroneously  // responded to the Halt command.  //  //===============================================================  u08\_t **Iso14443a\_Halt**(**void**)  {  u08\_t ui8Offset = 0;  **if** (Trf797xGetIsoControlValue() != 0x88)  {  // Trf797x has not been properly configured for ISO14443A  Trf797xWriteIsoControl(0x88); // Configure the TRF797x for ISO14443A @ 106kbps and Receive without CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x90; // Send without CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x50; // Halt Command  g\_ui8TrfBuffer[ui8Offset++] = 0x00;  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Halt Command  Trf797xIrqWaitTimeout(3,10); // 3 millisecond TX timeout, 10 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus != *NO\_RESPONSE\_RECEIVED*) // If PICC gives a response to the command, this means the Halt command failed or had an error  {  g\_sTrfStatus = *PROTOCOL\_ERROR*;  Trf797xSetTrfStatus(g\_sTrfStatus);  **#ifdef** ENABLE\_HOST  Report("Halt command error \n \r");  **#endif**  **return** STATUS\_FAIL;  }  **else**  {  **return** STATUS\_SUCCESS;  }  }  //===================================================================  //  // Iso14443a\_Rats - Issue the RATS command to the currently selected  // ISO14443A compliant tag.  //  // This function sends the RATS command to activate an ISO14443A  // compliant tag for data exchange.  //  // \return ui8Status returns whether or not the selected ISO14443A  // compliant tag responded to the RATS command.  //  //===================================================================  u08\_t **Iso14443a\_Rats**(**void**)  {  u08\_t ui8Offset = 0;  u08\_t ui8RxLength = 0;  u08\_t ui8Status = STATUS\_FAIL;  **#ifdef** ENABLE\_HOST  u08\_t ui8LoopCount = 0;  **#endif**  **if** (Trf797xGetIsoControlValue() != 0x08)  {  // Trf797x has not been properly configured for ISO14443A  Trf797xWriteIsoControl(0x08); // Configure the TRF797x for ISO14443A @ 106kbps and Receive with CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  }  // Buffer setup for FIFO writing  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = RATS\_CMD; //RATS Command  g\_ui8TrfBuffer[ui8Offset++] = RATS\_PARAM; //RATS Parameters: 128 byte max receive and CID = 0  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the RATS command  Trf797xIrqWaitTimeout(3,10); // 3 millisecond TX timeout, 10 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  // If data received  **if**(g\_sTrfStatus == *RX\_COMPLETE*)  {  ui8RxLength = Trf797xGetRxBytesReceived();  **if** (g\_ui8TrfBuffer[0] == ui8RxLength)  {  ui8Status = STATUS\_SUCCESS;  ui8Offset = 0;  // Print out TL  **#ifdef** ENABLE\_HOST  Report("ISO14443A ATS Response - TL: ");  UartPutByte(g\_ui8TrfBuffer[ui8Offset++]);  UartPutCrlf();  // If TL is greater than 1, at minimum Format Byte T0 will be present  **if** (g\_ui8TrfBuffer[0] > 1)  {  // Print out the value of T0  Report("ISO14443A ATS Response - T0: ");  UartPutByte(g\_ui8TrfBuffer[ui8Offset++]);  UartPutCrlf();  **if** (g\_ui8TrfBuffer[1] & 0x10)  {  g\_ui8AtsSupportedBitrates = g\_ui8TrfBuffer[ui8Offset++];  // TA(1) has been received, print it out  Report("ISO14443A ATS Response - TA(1): ");  UartPutByte(g\_ui8AtsSupportedBitrates);  UartPutCrlf();  }  **if** (g\_ui8TrfBuffer[1] & 0x20)  {  // TB(1) has been received, print it out  Report("ISO14443A ATS Response - TB(1): ");  UartPutByte(g\_ui8TrfBuffer[ui8Offset++]);  UartPutCrlf();  }  **if** (g\_ui8TrfBuffer[1] & 0x40)  {  // TC(1) has been received, print it out  Report("ISO14443A ATS Response - TC(1): ");  UartPutByte(g\_ui8TrfBuffer[ui8Offset++]);  UartPutCrlf();  }  **if** (ui8RxLength > ui8Offset)  {  // Historical Bytes have been received, print out all of them  Report("ISO14443A ATS Response - Historical Bytes: ");  **for** (ui8LoopCount = ui8Offset; ui8LoopCount < ui8RxLength; ui8LoopCount++)  {  UartPutByte(g\_ui8TrfBuffer[ui8LoopCount]);  }  UartPutCrlf();  }  }  UartPutCrlf();  **#endif**  }  **else**  {  ui8Status = STATUS\_FAIL;  }  }  **else**  {  ui8Status = STATUS\_FAIL;  }  **return** ui8Status;  }  //===================================================================  //  // Iso14443a\_Pps - Issue the PPS command to the currently selected  // ISO14443A compliant tag.  //  // This function sends the PPS command to modify the over the air  // data rate of the selected ISO14443A compliant tag.  //  // \return ui8Status returns whether or not the selected ISO14443A  // compliant tag responded to the PPS command.  //  //===================================================================  u08\_t **Iso14443a\_Pps**(**void**)  {  u08\_t ui8Offset = 0;  u08\_t ui8Status = STATUS\_FAIL;  u08\_t ui8PPSBitrate;  // Check if PPS is supported based on last received ATS reply  **if** ((g\_ui8AtsSupportedBitrates == 0x00) || (g\_ui8AtsSupportedBitrates == 0x80))  {  **#ifdef** ENABLE\_HOST  Report("Tag does not support data rates above 106kbps. No PPS issued.");  UartPutCrlf();  UartPutCrlf();  **#endif**  **return** ui8Status = STATUS\_SUCCESS;  }  **if** (Trf797xGetIsoControlValue() != 0x08)  {  // Trf797x has not been properly configured for ISO14443A  Trf797xWriteIsoControl(0x08); // Configure the TRF797x for ISO14443A @ 106kbps and Receive with CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  }  ui8PPSBitrate = PPS1\_106; // Set the PPS bit rate to 106kbps for best range performance  // It is recommended to keep the data rate low to get better transmission ranges  // Buffer setup for FIFO writing  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x30; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = PPSS; // PPS Command  g\_ui8TrfBuffer[ui8Offset++] = PPS0;  g\_ui8TrfBuffer[ui8Offset++] = ui8PPSBitrate; // Send PPS Bit Rate  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the PPS Command  Trf797xIrqWaitTimeout(3,10); // 3 millisecond TX timeout, 10 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  // If data received  **if**(g\_sTrfStatus == *RX\_COMPLETE*)  {  // Check PPS response  **if** (g\_ui8TrfBuffer[0] == PPSS)  {  ui8Status = STATUS\_SUCCESS;  // Execute Bitrate Change  McuDelayMillisecond(1);  **if** (ui8PPSBitrate == PPS1\_106)  {  Trf797xWriteIsoControl(0x08); // Configure the TRF797x for ISO14443A @ 106kbps and Receive with CRC  **#ifdef** ENABLE\_HOST  Report("PPS Successful: ISO14443A Bit Rate Set to 106kpbs");  UartPutCrlf();  UartPutCrlf();  **#endif**  }  **else** **if** (ui8PPSBitrate == PPS1\_212)  {  Trf797xWriteIsoControl(0x09); // Configure the TRF797x for ISO14443A @ 212kbps and Receive with CRC  **#ifdef** ENABLE\_HOST  Report("PPS Successful: ISO14443A Bit Rate Set to 212kpbs");  UartPutCrlf();  UartPutCrlf();  **#endif**  }  **else** **if** (ui8PPSBitrate == PPS1\_424)  {  Trf797xWriteIsoControl(0x0A); // Configure the TRF797x for ISO14443A @ 424kbps and Receive with CRC  **#ifdef** ENABLE\_HOST  Report("PPS Successful: ISO14443A Bit Rate Set to 424kpbs");  UartPutCrlf();  UartPutCrlf();  **#endif**  }  **else** **if** (ui8PPSBitrate == PPS1\_848)  {  Trf797xWriteIsoControl(0x0B); // Configure the TRF797x for ISO14443A @ 848kbps and Receive with CRC  **#ifdef** ENABLE\_HOST  Report("PPS Successful: ISO14443A Bit Rate Set to 848kpbs");  UartPutCrlf();  UartPutCrlf();  **#endif**  }  **else**  {  // Do Nothing  }  McuDelayMillisecond(6);  }  **else**  {  ui8Status = STATUS\_FAIL;  **#ifdef** ENABLE\_HOST  Report("Error: PPS Reply Does Not Match Transmitted Frame");  UartPutCrlf();  **#endif**  }  }  **else**  {  ui8Status = STATUS\_FAIL;  **#ifdef** ENABLE\_HOST  Report("Error: PPS Reply Not Received");  UartPutCrlf();  **#endif**  }  **return** ui8Status;  }  //===============================================================  //  // Iso14443a\_StoreUid - Store the received UID bytes into the  // global g\_pui8CompleteUid buffer.  //  // \param sCascade is the current anticollision cascade.  // \param pui8UID is the location of the UID bytes to store.  //  // This function stores the received UID bytes into the global  // UID buffer. It will also parse out the CT byte and set a flag  // to indicate if the CT byte was present.  //  // \return bSendCT returns whether or not the next transmission  // will require the CT byte to be sent in addition to the UID  // bytes.  //  //===============================================================  bool **Iso14443a\_StoreUid**(tISO14443A\_UidStatus sCascade, u08\_t \* pui8UID)  {  bool bSendCT = false;  u08\_t ui8Offset = 0;  **if** ((g\_sUidSize == *ISO14443A\_UID\_SINGLE*) && (sCascade == *CASCADE1*))  {  // UID has no CT, so store all bytes normally.  ui8Offset = 0;  bSendCT = false;  }  **else** **if** ((g\_sUidSize == *ISO14443A\_UID\_DOUBLE*) && (sCascade == *CASCADE2*))  {  // UID has no CT, so store all bytes normally.  ui8Offset = 0;  bSendCT = false;  }  **else** **if** ((g\_sUidSize == *ISO14443A\_UID\_TRIPLE*) && (sCascade == *CASCADE3*))  {  // UID has no CT, so store all bytes normally.  ui8Offset = 0;  bSendCT = false;  }  **else**  {  **if** (pui8UID[0] == 0x88)  {  // UID has a CT, set bool to return that a CT must be sent  ui8Offset = 1; // Set offset to account for the location of the CT byte  bSendCT = true; // Set variable to tell Select Command to include a CT in addition to UID bytes  }  **else**  {  // UID has no CT, so store all bytes normally.  ui8Offset = 0;  bSendCT = false;  }  }  // Store UID based on the current Cascade level  **if** (sCascade == *CASCADE1*)  {  g\_pui8CompleteUid[0] = pui8UID[0+ui8Offset];  g\_pui8CompleteUid[1] = pui8UID[1+ui8Offset];  g\_pui8CompleteUid[2] = pui8UID[2+ui8Offset];  g\_pui8CompleteUid[3] = pui8UID[3+ui8Offset]; // BCC Byte or last byte of UID  g\_ui8UidPos = 0; // Update the UID Position indicator to the first byte of the newly stored UID for when the Select command is issued  }  **else** **if** (sCascade == *CASCADE2*)  {  g\_pui8CompleteUid[3] = pui8UID[0+ui8Offset]; // Override the BCC from prior Cascade as it is no longer needed  g\_pui8CompleteUid[4] = pui8UID[1+ui8Offset];  g\_pui8CompleteUid[5] = pui8UID[2+ui8Offset];  g\_pui8CompleteUid[6] = pui8UID[3+ui8Offset]; // BCC Byte or last byte of UID  g\_ui8UidPos = 3; // Update the UID Position indicator to the first byte of the newly stored UID for when the Select command is issued  }  **else** **if** (sCascade == *CASCADE3*)  {  g\_pui8CompleteUid[6] = pui8UID[0]; // Override the BCC from prior Cascade as it is no longer needed  g\_pui8CompleteUid[7] = pui8UID[1];  g\_pui8CompleteUid[8] = pui8UID[2];  g\_pui8CompleteUid[9] = pui8UID[3];  bSendCT = false; // Ensure no accidental sending of the CT occurs incase uid6 for a Triple Size UID = 0x88 (which is permitted per ISO14443-3 specifications)  g\_ui8UidPos = 6; // Update the UID Position indicator to the first byte of the newly stored UID for when the Select command is issued  }  **return** bSendCT;  }  //===============================================================  //  // Iso14443a\_Type2\_Read4Blocks - Reads out four blocks of data  // from NFC Type 2 Tag Platforms.  //  // \param ui8StartBlock is the block number to start reading the  // tag data from.  //  // This function will issue a Read Block command with the  // provided starting block number. The Read Block command  // automatically will read out four blocks of data from the Type  // 2 Tag.  //  // \return ui8Status returns whether or not the tag data was  // successfully read.  //  //===============================================================  u08\_t **Iso14443a\_Type2\_Read4Blocks**(u08\_t ui8StartBlock)  {  u08\_t ui8Offset = 0;  u08\_t ui8Status = STATUS\_FAIL;  **#ifdef** ENABLE\_HOST  u08\_t ui8LoopCount1 = 1;  u08\_t ui8LoopCount2 = 0;  **#endif**  **if** (Trf797xGetIsoControlValue() != 0x08)  {  // Trf797x has not been properly configured for ISO14443A  Trf797xWriteIsoControl(0x08); // Configure the TRF797x for ISO14443A @ 106kbps and Receive with CRC  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x30; // Read Command  g\_ui8TrfBuffer[ui8Offset++] = ui8StartBlock; // Starting from Block # (called Bno)  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Type 2 Read Command  Trf797xIrqWaitTimeout(5,30); // 5 millisecond TX timeout, 30 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if**(g\_sTrfStatus == *RX\_COMPLETE*) // If block data has been received  {  ui8Status = STATUS\_SUCCESS; // Mark tag has been successfully read  **#ifdef** ENABLE\_HOST  **for**(ui8LoopCount2 = 0; ui8LoopCount2 < 4; ui8LoopCount2++)  {  Report("NFC Type 2 Block ");  UartPutByte(ui8StartBlock++);  Report(": [");  **for**(ui8LoopCount1 = (ui8LoopCount2\*4); ui8LoopCount1 < 4+(ui8LoopCount2\*4); ui8LoopCount1++)  {  UartPutByte(g\_ui8TrfBuffer[ui8LoopCount1]); // Print out the received data  }  UartPutChar(']');  UartPutCrlf();  }  **#endif**  }  **else**  {  // Otherwise return a fail  ui8Status = STATUS\_FAIL;  }  **return** ui8Status;  }  //===============================================================  //  // Iso14443a\_Get\_Type4ACompliance - Fetches g\_bType4ACompliant value  //  // This function allows for higher layers to fetch the current  // Type 4A NDEF compliance information.  //  // \return g\_bType4ACompliant returns the current compliance for  // Type 4A NDEF.  //  //===============================================================  bool **Iso14443a\_Get\_Type4ACompliance**(**void**)  {  **return** g\_bType4ACompliant;  }  //===============================================================  //  // Iso14443a\_Set\_RecursionCount - Sets the g\_ui8RecursionCount  // variable  //  // \param  //  // This function allows for higher layers to adjust the global  // recursion count. Useful for resetting it prior to running  // anticollision routines.  //  // \return None.  //  //===============================================================  **void** **Iso14443a\_Set\_RecursionCount**(u08\_t ui8RecursionCount)  {  g\_ui8RecursionCount = ui8RecursionCount;  }  //===============================================================  //  // Iso14443a\_Get\_Uid - Fetches the ISO14443A Tag UID.  //  // This function allows for higher layers to fetch the tag UID of  // an ISO14443A tag. In the current implementation, the UID  // stored is from the most recent tag which finished the  // anticollision procedure.  //  // \return g\_pui8CompleteUid returns the currently stored UID.  //  //===============================================================  u08\_t \* **Iso14443a\_Get\_Uid**(**void**)  {  **return** g\_pui8CompleteUid;  }  //===============================================================  //  // Iso14443a\_Get\_UidSize - Fetches the UID size of the most  // recently read ISO14443A tag.  //  // This function allows for higher layers to fetch the size of  // the current UID for an ISO14443A tag.  //  // \return g\_sUidSize returns the current UID size.  //  //===============================================================  tISO14443A\_UidSize **Iso14443a\_Get\_UidSize**(**void**)  {  **return** g\_sUidSize;  }  //===============================================================  //  // Nfc\_Iso14443a\_Type4NdefApp - Customizeable application to read  // NDEF data from a Type 4A NDEF Formatted tag.  //  // Tags which are Type 4 compliant are activated via RATS.  // If the tag contains an NDEF message, then the NDEF data is  // read from it.  //  // \return None.  //  //===============================================================  **void** **Nfc\_Iso14443a\_Type4NdefApp**(**void**)  {  **if** (Iso14443a\_Rats() == STATUS\_SUCCESS)  {  Ndef\_SetBlockNumberBit(0);  **if** (NDEF\_ApplicationSelect() == STATUS\_SUCCESS) // Selects NDEF Application  {  Report("Payload Received: %s\n\r",g\_ndef\_content);  g\_ndef\_content\_received = 1;  }  **else**  {  Report("ISO14443a: Select AID failure\n\r");  }  }  **else**  {  Report("ISO14443a: RATS Failure\n\r");  }  }  **void** **ISO14443aFindTag**(**void**)  {  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  Trf797xWriteInitiatorSetup(0x88); // Configure the TRF797x for ISO14443A @ 106kbps and Receive no CRC  IRQ\_CLR; // PORT2 interrupt flag clear  IRQ\_ON;  // When a PICC is exposed to an unmodulated operating field  // it shall be able to accept a quest within 5 ms.  // PCDs should periodically present an unmodulated field of at least  // 5.1 ms duration. (ISO14443-3)  McuDelayMillisecond(6);  Iso14443a\_Set\_RecursionCount(0); // Clear the recursion count for anticollision loops  **if** (Iso14443a\_TagSelection(REQA) == STATUS\_SUCCESS) // Do a complete anticollision sequence as described in ISO14443-3 standard for type A  {  **if** (Iso14443a\_Get\_Type4ACompliance() == true)  {  //Report("Type 4A Tag\n\r");  Nfc\_Iso14443a\_Type4NdefApp(); // For a Type 4A compliant tag, the tag is put into Layer 4, and in order to attempt to read/write NDEF contents  }  }  Trf797xTurnRfOff();  } |

|  |
| --- |
| **iso15693.h** |
| /\*  \* {iso15693.h}  \*  \* {ISO15693 header file}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_ISO15693\_H\_  **#define** \_ISO15693\_H\_  //================================================================  **#include** "trf797x.h"  **#include** "types.h"  **#include** "uart.h"  //===============================================================  // Premade Request Flags  **#define** T5T\_SINGLE\_SLOT\_INVENTORY 0x26 // Tables 3 & 5 in ISO15693-3; single-subcarrier, high tag data rate, one slot  **#define** T5T\_SIXTEEN\_SLOT\_INVENTORY 0x06 // Tables 3 & 5 in ISO15693-3; single-subcarrier, high tag data rate, sixteen slots  // Individual Flags  **#define** T5T\_REQ\_FLAG\_SUB\_CARRIER 0x01  **#define** T5T\_REQ\_FLAG\_HIGH\_DATA 0x02  **#define** T5T\_REQ\_FLAG\_INVENTORY 0x04  **#define** T5T\_REQ\_FLAG\_EXTENDED 0x08  **#define** T5T\_REQ\_FLAG\_SELECT 0x10  **#define** T5T\_REQ\_FLAG\_ADDRESSED 0x20  **#define** T5T\_REQ\_FLAG\_OPTION 0x40  **#define** T5T\_INV\_FLAG\_AFI 0x10  **#define** T5T\_INV\_FLAG\_SINGLE\_SLOT 0x20  **#define** T5T\_INV\_FLAG\_OPTION 0x40  **#define** T5T\_RESP\_FLAG\_NO\_ERROR 0x00  **#define** T5T\_RESP\_FLAG\_EXTENDED 0x08  //===============================================================  u08\_t Iso15693\_SingleSlotInventory(**void**);  u08\_t Iso15693\_Anticollision(u08\_t ui8ReqFlags, u08\_t ui8MaskLength, u08\_t ui8Afi);  u16\_t Iso15693\_GetSystemInfo(u08\_t ui8ReqFlag);  u16\_t Iso15693\_GetSystemInfoExtended(u08\_t ui8ReqFlag);  u08\_t Iso15693\_ReadSingleBlock(u08\_t ui8ReqFlag, u08\_t ui8BlockNumber);  u08\_t Iso15693\_ReadMultipleBlocks(u08\_t ui8ReqFlag, u08\_t ui8FirstBlock, u08\_t ui8NumberOfBlocks);  u08\_t Iso15693\_ReadSingleBlockExtended(u08\_t ui8ReqFlag, u16\_t ui16StartBlock);  u08\_t Iso15693\_WriteSingleBlock(u08\_t ui8ReqFlag, u08\_t ui8BlockNumber, u08\_t ui8BlockSize, u08\_t \* pui8BlockData);  u08\_t \* Iso15693\_Get\_Uid(**void**);  u08\_t Iso15693\_Get\_TagCount(**void**);  **void** Iso15693\_Reset\_TagCount(**void**);  **extern** **void** ISO15693FindTag(**void**);  //===============================================================  **#endif** |

|  |
| --- |
| **iso15693.c** |
| /\*  \* {iso15693.c}  \*  \* {ISO15693 Specific Functions & Anti-collision}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#include** "hw\_types.h"  **#include** <string.h>  **#include** <stdio.h>  **#include** "iso15693.h"  **#include** "trf7970BoosterPack.h"  **#include** "uart\_if.h"  **#include** "timer.h"  **#include** "timer\_if.h"  **#include** "utils.h"  **#include** "gpio.h"  **#include** "gpio\_if.h"  **#include** "hw\_memmap.h"  **#include** "uart\_if.h"  **#include** "osi.h"  //===============================================================  // Global Variables  //===============================================================  **extern** uint8\_t g\_ui8TrfBuffer[NFC\_FIFO\_SIZE];  **static** **volatile** tTRF797x\_Status g\_sTrfStatus;  **static** uint8\_t g\_pui8Iso15693UId[8];  **static** uint8\_t g\_pui8AnticollisionMaskBuffer[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};  **static** uint8\_t g\_ui8TagDetectedCount = 0;  //===============================================================  //  // Iso15693\_SingleSlotInventory - Issue a single slot Inventory  // command for ISO15693 tags.  //  // This function issues a single slot Inventory command for tag  // detection of ISO15693 tags. If a tag is found, the UID is  // stored in the g\_pui8Iso15693UId buffer.  //  // If UART is enabled, the tag ID is sent out to a host via UART  // as well.  //  // \return ui8Status returns either STATUS\_SUCCESS or STATUS\_FAIL  // to indicate if the Inventory command resulted in a successful  // tag detection or not.  //  //===============================================================  uint8\_t Iso15693\_SingleSlotInventory(**void**)  {  uint8\_t ui8Offset = 0;  uint8\_t ui8LoopCount = 0;  uint8\_t ui8Status = STATUS\_FAIL;  uint8\_t ui8RssiLevel;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x30; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x26; // ISO15693 flags  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // Inventory command code  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Mask Length = 0 (Also not sending AFI)  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the ISO15693 Inventory Command  Trf797xIrqWaitTimeout(5,15); // 5 millisecond TX timeout, 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  ui8Status = STATUS\_SUCCESS;  ui8RssiLevel = Trf797xReadRssiLevels(); // Read the RSSI levels  // UID Starts at the 3rd received bit (1st is flags and 2nd is DSFID)  **for** (ui8LoopCount = 2; ui8LoopCount < 10; ui8LoopCount++)  {  g\_pui8Iso15693UId[ui8LoopCount-2] = g\_ui8TrfBuffer[ui8LoopCount]; // Store UID into a Buffer  }  g\_ui8TagDetectedCount++;  // Print out UID and RSSI level to UART Host  UartPutCrlf();  UartSendCString("ISO15693/NFC-V UID: ");  UartPutChar('[');  **for** (ui8LoopCount = 0; ui8LoopCount < 8; ui8LoopCount++)  {  UartPutByte(g\_pui8Iso15693UId[7-ui8LoopCount]); // Send UID to host  }  UartPutChar(']');  UartPutCrlf();  UartSendCString("RSSI LEVEL: ");  UartPutChar('[');  UartPutByte(ui8RssiLevel); // Send RSSI level to host  UartPutChar(']');  UartPutCrlf();  }  }  **else**  {  ui8Status = STATUS\_FAIL;  }  // clear any IRQs  Trf797xResetIrqStatus();  **return** ui8Status;  }  //===============================================================  //  // Iso15693\_Anticollision - Issue an Inventory command for either  // 1 or 16 slot anticollision of ISO15693 tags.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  // \param pui8Mask are the masked nibbles to issue with the  // Inventory command  // \param ui8MaskLength is the number of significant bits in the  // mask value.  // \param ui8Afi is the AFI to be issued with command (if AFI  // flag is included in ui8ReqFlag)  //  // This function issues a single or sixteens lot Inventory  // command for tag detection of ISO15693 tags. If a tag is found,  // the UID is stored in the g\_pui8Iso15693UId buffer. The process  // will run until all ISO15693 detected have responded with their  // UID's.  //  // The function uses a recursive call for the anticollision  // process. Since the UID is stored inside of a buffer, only  // the last ISO15693 tag to respond with it's UID will have  // that UID stored in order to read data from the tag.  //  // If UART is enabled, the UID of each ISO15693 tag detected is  // sent out to a host via UART.  //  // \return ui8Status returns STATUS\_SUCCESS if the anticollision  // function resulted in a successful tag detection. Otherwise,  // returns STATUS\_FAIL.  //  //===============================================================  uint8\_t Iso15693\_Anticollision(uint8\_t ui8ReqFlags, uint8\_t ui8MaskLength, uint8\_t ui8Afi)  {  uint8\_t ui8Offset = 0;  uint8\_t ui8LoopCount1 = 1;  uint8\_t ui8LoopCount2 = 1;  uint8\_t ui8SlotCount;  uint16\_t ui16TransmitByteLength;  uint16\_t ui16SlotNumber = 0x0000;  uint8\_t ui8MaskValue;  uint8\_t ui8MaskByteCount;  uint8\_t ui8Status = STATUS\_FAIL;  **if** ((ui8ReqFlags & BIT5) == 0x00) // Check if Bit 5 in the Request Flag to determine the number of slots  {  ui8SlotCount = 16; // If Bit 5 is cleared, then use 16 slots  }  **else**  {  ui8SlotCount = 1; // If Bit 5 is set, then use 1 slot  }  ui8MaskByteCount = (((ui8MaskLength >> 2) + 1) >> 1); // Set ui8MaskByteCount based on the inputted Mask Length  // ui8MaskByteCount will be 1 for length = 4 or 8,  // ui8MaskByteCount will be 2 for length = 12 or 16,  // and so on  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  // Format Anti-collision command packet  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  // Calculate how long the output byte will be  **if** (ui8ReqFlags & 0x10) // Check if AFI will be included or not  {  ui16TransmitByteLength = ui8MaskByteCount + 4; // Set packet size = Mask Value + Mask Length + AFI Byte + ISO15693 Command Code + ISO15693 Request Flags  }  **else**  {  ui16TransmitByteLength = ui8MaskByteCount + 3; // Set packet size = Mask Value + Mask Length + ISO15693 Command Code + ISO15693 Request Flags  }  g\_ui8TrfBuffer[ui8Offset++] = (uint8\_t) (ui16TransmitByteLength >> 8); // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = (uint8\_t) (ui16TransmitByteLength << 4); // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlags; // ISO15693 Request Flags  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // Inventory Request Command Code  **if** (ui8ReqFlags & 0x10) // Check if AFI will be included or not  {  g\_ui8TrfBuffer[ui8Offset++] = ui8Afi; // Optional AFI Byte  g\_ui8TrfBuffer[ui8Offset++] = ui8MaskLength; // Mask Length  **if** (ui8MaskLength > 0)  {  **for** (ui8LoopCount1 = 0; ui8LoopCount1 < ui8MaskByteCount; ui8LoopCount1++)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8AnticollisionMaskBuffer[(ui8MaskByteCount-ui8LoopCount1)]; // Fill in inputted Mask Values  }  }  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = ui8MaskLength; // Mask Length  **if** (ui8MaskLength > 0)  {  **for** (ui8LoopCount1 = 0; ui8LoopCount1 < ui8MaskByteCount; ui8LoopCount1++)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8AnticollisionMaskBuffer[((ui8MaskByteCount-1)-ui8LoopCount1)]; // Fill in inputted Mask Values  }  }  }  Trf797xEnableSlotCounter();  Trf797xResetIrqStatus();  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the ISO15693 Inventory Command  Trf797xIrqWaitTimeoutTxOnly(5); // 5 millisecond TX timeout  **for** (ui8LoopCount2 = 1; ui8LoopCount2 <= ui8SlotCount; ui8LoopCount2++) // There will be either 1 or 16 available time slots  {  Trf797xIrqWaitTimeoutRxOnly(15); // 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus(); // Get the TRF797x Status  **switch** (g\_sTrfStatus)  {  **case** RX\_COMPLETE: // If data has been received, then UID is in the buffer  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  ui8Status = STATUS\_SUCCESS;  // UID Starts at the 3rd received bit (1st is flags and 2nd is DSFID)  **for** (ui8LoopCount1 = 2; ui8LoopCount1 < 10; ui8LoopCount1++)  {  g\_pui8Iso15693UId[ui8LoopCount1-2] = g\_ui8TrfBuffer[ui8LoopCount1]; // Store UID to a Buffer  }  g\_ui8TagDetectedCount++;  // Print out UID and RSSI level to UART Host  UartPutCrlf();  UartSendCString("ISO15693/NFC-V UID: ");  UartPutChar('[');  **for** (ui8LoopCount1 = 0; ui8LoopCount1 < 8; ui8LoopCount1++)  {  UartPutByte(g\_pui8Iso15693UId[7-ui8LoopCount1]); // Send UID to host  }  UartPutChar(']');  UartPutCrlf();  }  **break**;  **case** PROTOCOL\_ERROR: // A collision has occurred for this slot  ui16SlotNumber |= (0x01 << (ui8LoopCount2-1)); // Mark a collision occurred in the correct Slot Number bit.  McuDelayMillisecond(5); // Allow time for tag to finish responding before issuing EOF  **break**;  **case** NO\_RESPONSE\_RECEIVED: // No Response was received, break out of the switch statement as there is no tag present  **break**;  **case** NO\_RESPONSE\_RECEIVED\_15693: // No Response was received, break out of the switch statement as there is no tag present  **break**;  **default**:  **break**;  }  Trf797xReset(); // FIFO has to be reset before receiving the next response  **if** ((ui8SlotCount == 16) && (ui8LoopCount2 < 16)) // If 16 slots used, and the last slot as not been reached, then send EOF (i.e. next slot indicator)  {  Trf797xStopDecoders();  Trf797xRunDecoders();  Trf797xTransmitNextSlot();  }  **else** **if** ((ui8SlotCount == 16) && (ui8LoopCount2 == 16)) // Once at the end of slot 16, then stop the slot counter  {  Trf797xStopDecoders();  Trf797xDisableSlotCounter();  }  **else** **if** (ui8SlotCount == 1) // Only 1 slot to be used, no more anti-collision to be performed  {  **break**;  }  }  Trf797xDisableSlotCounter();  ui8MaskLength = ui8MaskLength + 4; // The mask length is a multiple of 4 bits  ui8MaskByteCount = (((ui8MaskLength >> 2) + 1) >> 1); // Set ui8MaskByteCount based on the inputted Mask Length  // ui8MaskByteCount is 1 for length = 4 or 8,  // ui8MaskByteCount is 2 for length = 12 or 16,  // and so on  // If the slot number pointer is not 0, the slot count is 16 (to indicate anticollision is needed),  // the mask length doesn't exceed 60 bits, and the slot number is not 16 then proceed to recursive function call  **while** ((ui16SlotNumber != 0x00)  && (ui8SlotCount == 16)  && (ui8MaskLength < 61))  {  ui8MaskValue = 0x00;  ui8LoopCount1 = 0;  **while** (ui8LoopCount1 < 16)  {  **if** ((ui16SlotNumber & (0x01 << ui8LoopCount1)) != 0x00)  {  ui8MaskValue = ui8LoopCount1;  ui16SlotNumber &= ~(0x01 << ui8LoopCount1); // Clear that slot bit from the array  **break**;  }  ui8LoopCount1++;  }  **if** ((ui8MaskLength & 0x04) == 0x00)  {  ui8MaskValue = ui8MaskValue << 4; // Shift slot pointer if mask length doesn't have Bit 2 (0x04) set (since it is a multiple of 4 bits)  }  **else**  { // Otherwise re-copy the mask values  **for** (ui8LoopCount1 = 7; ui8LoopCount1 > 0; ui8LoopCount1--)  {  g\_pui8AnticollisionMaskBuffer[ui8LoopCount1] = g\_pui8AnticollisionMaskBuffer[ui8LoopCount1 - 1];  }  g\_pui8AnticollisionMaskBuffer[0] &= 0x00; // And set the mask value for the first byte in the array = 0  }  g\_pui8AnticollisionMaskBuffer[0] |= ui8MaskValue; // Now update the mask value of the first byte based on the slot number pointer  McuDelayMillisecond(2);  ui8Status = Iso15693\_Anticollision(ui8ReqFlags, ui8MaskLength, ui8Afi); // Issue a recursive call with new Mask  // Restore the Global AnticollisionMaskBuffer with the values from the current anticollision function.  **if** ((ui8MaskLength & 0x04) == 0x00)  {  // If mask length doesn't have Bit 2 (0x04) set (since it is a multiple of 4 bits) - clear the upper nibble which is where the new mask value was placed  g\_pui8AnticollisionMaskBuffer[0] &= 0x0F;  }  **else**  { // Otherwise re-shift the mask values  **for** (ui8LoopCount1 = 0; ui8LoopCount1 < 7; ui8LoopCount1++)  {  g\_pui8AnticollisionMaskBuffer[ui8LoopCount1] = g\_pui8AnticollisionMaskBuffer[ui8LoopCount1 + 1];  }  g\_pui8AnticollisionMaskBuffer[7] = 0x00; // And set the mask value for the first byte in the array = 0  }  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui8Status;  }  //===============================================================  //  // Iso15693\_GetSystemInfo - Issue the Get System Information  // command for ISO15693 tags.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  //  // This function issues a Get System Information command for  // ISO15693 tags. This can be used to determine how many blocks  // of data can be read from the tag.  //  // If UART is enabled, the contents of the Get System Information  // response is sent out to a host via UART as well.  //  // \return ui16NumberOfBlocks returns the number of blocks  // contained in the ISO15693 tag.  //  //===============================================================  uint16\_t Iso15693\_GetSystemInfo(uint8\_t ui8ReqFlag)  {  uint8\_t ui8Offset = 0;  uint16\_t ui16NumberOfBlocks = 0x00;  uint8\_t ui8LoopCount = 1;  uint8\_t ui8RxLength = 0;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = 0xA0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlag; // ISO15693 flags  g\_ui8TrfBuffer[ui8Offset++] = 0x2B; // Get System Information command code  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[0]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[1]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[2]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[3]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[4]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[5]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[6]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[7]; // Tag UID  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Get System Information command  Trf797xIrqWaitTimeout(5,15); // 5 millisecond TX timeout, 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  // Output received data to UART  UartSendCString("Get Sys Info ");  UartSendCString("Data: ");  UartPutChar('[');  ui8RxLength = Trf797xGetRxBytesReceived();  **for** (ui8LoopCount = 1; ui8LoopCount < ui8RxLength; ui8LoopCount++)  {  UartPutByte(g\_ui8TrfBuffer[ui8LoopCount]); // Send Get System Info data to host  }  UartPutChar(']');  UartPutCrlf();  // Check to see that no error flags were sent and that there is a block number data available  **if** (g\_ui8TrfBuffer[0] == 0x00 && ((g\_ui8TrfBuffer[1] & 0x07) == 0x07))  {  ui16NumberOfBlocks = g\_ui8TrfBuffer[12];  }  **else** **if** (g\_ui8TrfBuffer[0] == 0x00 && (((g\_ui8TrfBuffer[1] & 0x07) == 0x06) || ((g\_ui8TrfBuffer[1] & 0x07) == 0x05)))  {  ui16NumberOfBlocks = g\_ui8TrfBuffer[11];  }  **else** **if** (g\_ui8TrfBuffer[0] == 0x00 && ((g\_ui8TrfBuffer[1] & 0x07) == 0x04))  {  ui16NumberOfBlocks = g\_ui8TrfBuffer[10];  }  }  }  **else** **if** ((g\_sTrfStatus == NO\_RESPONSE\_RECEIVED) || (g\_sTrfStatus == NO\_RESPONSE\_RECEIVED\_15693))  {  // Case for TI HF-I Pro and Standard tags  ui16NumberOfBlocks = 0x0A;  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui16NumberOfBlocks;  }  //===============================================================  //  // Iso15693\_GetSystemInfoExtended - Issue the Get System  // Information command for ISO15693 tags with the protocol  // extention flag set.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  //  // This function issues a Get System Information command for  // ISO15693 tags with the Protocol Extension bit set in the  // request flags. This can be used to determine how many blocks  // of data can be read from the tag.  //  // If UART is enabled, the contents of the Get System Information  // response is sent out to a host via UART as well.  //  // \return ui16NumberOfBlocks returns the number of blocks  // contained in the ISO15693 tag.  //  //===============================================================  uint16\_t Iso15693\_GetSystemInfoExtended(uint8\_t ui8ReqFlag)  {  uint8\_t ui8Offset = 0;  uint16\_t ui16NumberOfBlocks = 0x00;  uint8\_t ui8LoopCount = 1;  uint8\_t ui8RxLength = 0;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = 0xA0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlag | 0x08; // ISO15693 flags + protocol extension bit  g\_ui8TrfBuffer[ui8Offset++] = 0x2B; // Get System Information command code  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[0]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[1]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[2]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[3]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[4]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[5]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[6]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[7]; // Tag UID  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Get System Information command  Trf797xIrqWaitTimeout(5,20); // 5 millisecond TX timeout, 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  UartSendCString("Get Sys Info ");  UartSendCString("Data: ");  UartPutChar('[');  ui8RxLength = Trf797xGetRxBytesReceived();  // Output received data to UART  **for** (ui8LoopCount = 1; ui8LoopCount < ui8RxLength; ui8LoopCount++)  {  UartPutByte(g\_ui8TrfBuffer[ui8LoopCount]); // Send Get System Info data to host  }  UartPutChar(']');  UartPutCrlf();  ui16NumberOfBlocks = 0x0800; // Set block size in order to read STM M24LR64 tag  }  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui16NumberOfBlocks;  }  //===============================================================  //  // Iso15693\_ReadSingleBlock - Issue the Read Single Block command  // for ISO15693 tags.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  // \param ui8BlockNumber is the block number to read data from.  //  // This function issues a Read Single Block with the specified  // request flags and block number to read data from an ISO15693  // tag.  //  // If UART is enabled, the data read from the ISO15693 tag is  // sent out to a host via UART as well.  //  // \return ui8Status returns either STATUS\_SUCCESS or STATUS\_FAIL  // to indicate if the Read Single Block was successful or not.  //  //===============================================================  uint8\_t Iso15693\_ReadSingleBlock(uint8\_t ui8ReqFlag, uint8\_t ui8BlockNumber)  {  uint8\_t ui8Offset = 0;  uint8\_t ui8Status = STATUS\_FAIL;  uint8\_t ui8LoopCount = 1;  uint8\_t ui8RxLength = 0;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = 0xB0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = 0x30; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlag; // ISO15693 flags  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Read Single Block command code  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[0]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[1]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[2]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[3]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[4]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[5]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[6]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[7]; // Tag UID  }  g\_ui8TrfBuffer[ui8Offset++] = ui8BlockNumber; // Block # (variable, for HF-I Plus device can go to 0x3F, Pro and Standard handled with "error" response flags)  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Get System Information command  Trf797xIrqWaitTimeout(5,15); // 5 millisecond TX timeout, 15 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  UartSendCString("NFC-V Block ");  UartPutByte(ui8BlockNumber); // Output block number  UartSendCString(" Data: ");  UartPutChar('[');  ui8RxLength = Trf797xGetRxBytesReceived();  **if** (ui8ReqFlag & BIT6) // Handle case for Option Flag causing one extra byte to be transmitted.  {  ui8Offset = 2;  }  **else**  {  ui8Offset = 1;  }  // Output received data to UART  **for** (ui8LoopCount = ui8Offset; ui8LoopCount < ui8RxLength; ui8LoopCount++)  {  UartPutByte(g\_ui8TrfBuffer[ui8LoopCount]); // Send out data read from tag to host  }  UartPutChar(']');  UartPutCrlf();  // Response received  ui8Status = STATUS\_SUCCESS;  }  **else** // An error has been sent back in the response byte  {  // Indicates when an error occurs or block addresses are unreachable - useful for debugging  UartSendCString("NFC-V Block ");  UartPutByte(ui8BlockNumber); // Output block number  UartSendCString(" Error");  UartPutCrlf();  UartSendCString("ISO15693 Error Code: ");  UartPutByte(g\_ui8TrfBuffer[1]); // Output ISO15693 error code  UartPutCrlf();  // Response with error  ui8Status = STATUS\_FAIL;  }  }  **else**  {  // No response  ui8Status = STATUS\_FAIL;  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui8Status;  }  //===============================================================  //  // Iso15693\_ReadMultipleBlocks - Issue the Read Multiple Block  // command for ISO15693 tags.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  // \param ui8FirstBlock is the starting block number to read data  // from.  // \param ui8NumberOfBlocks is the amount of blocks to read data  // from.  //  // This function issues a Read Multiple Block with the specified  // request flags, the starting block number, and the number  // blocks to read data from an ISO15693 tag.  //  // If UART is enabled, the data read from the ISO15693 tag is  // sent out to a host via UART as well.  //  // \return ui8Status returns either STATUS\_SUCCESS or STATUS\_FAIL  // to indicate if the Read Single Block was successful or not.  //  //===============================================================  uint8\_t Iso15693\_ReadMultipleBlocks(uint8\_t ui8ReqFlag, uint8\_t ui8FirstBlock, uint8\_t ui8NumberOfBlocks)  {  uint8\_t ui8Offset = 0;  uint8\_t ui8Status = STATUS\_FAIL;  uint8\_t ui8LoopCount1 = 0;  uint8\_t ui8LoopCount2 = 0;  uint8\_t ui8RxLength = 0;  uint8\_t ui8BlockSize = 0;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = 0xC0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = 0x40; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlag; // ISO15693 flags  g\_ui8TrfBuffer[ui8Offset++] = 0x23; // Read Multiple Block command code  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[0]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[1]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[2]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[3]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[4]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[5]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[6]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[7]; // Tag UID  }  g\_ui8TrfBuffer[ui8Offset++] = ui8FirstBlock; // Number of the first block to read from  **if** (ui8NumberOfBlocks > 0)  {  g\_ui8TrfBuffer[ui8Offset++] = ui8NumberOfBlocks-1; // Index for number of blocks to be read - this value is one less than  }  **else**  {  // Invalid count provided  **return** ui8Status = STATUS\_FAIL;  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Get System Information command  Trf797xIrqWaitTimeout(5,15+ui8NumberOfBlocks); // 5 millisecond TX timeout, 15 millisecond RX timeout - adding number of blocks to extend timeout for larger read requests  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  ui8RxLength = Trf797xGetRxBytesReceived();  **if** (ui8ReqFlag & BIT6) // Handle case for Option Flag causing one extra byte to be transmitted.  {  ui8Offset = 2;  }  **else**  {  ui8Offset = 1;  }  ui8LoopCount1 = ui8RxLength-ui8Offset;  **while** (ui8LoopCount1 > 0)  {  **if** (ui8LoopCount1 > ui8NumberOfBlocks)  {  ui8LoopCount1 = ui8LoopCount1 - ui8NumberOfBlocks;  }  **else**  {  ui8LoopCount1 = 0;  }  ui8BlockSize++;  }  **for** (ui8LoopCount2 = 0; ui8LoopCount2 < ui8NumberOfBlocks; ui8LoopCount2++)  {  UartSendCString("NFC-V Block ");  UartPutByte(ui8FirstBlock+ui8LoopCount2); // Output block number  UartSendCString(" Data: ");  UartPutChar('[');  // Output received data to UART  **for** (ui8LoopCount1 = 0; ui8LoopCount1 < ui8BlockSize; ui8LoopCount1++)  {  UartPutByte(g\_ui8TrfBuffer[ui8Offset++]); // Send out data read from tag to host  }  UartPutChar(']');  UartPutCrlf();  }  // Response received  ui8Status = STATUS\_SUCCESS;  }  **else** // An error has been sent back in the response byte  {  // Indicates when an error occurs or block addresses are unreachable - useful for debugging  UartSendCString("NFC-V Block ");  UartPutByte(ui8FirstBlock); // Output block number  UartSendCString(" Error");  UartPutCrlf();  UartSendCString("ISO15693 Error Code: ");  UartPutByte(g\_ui8TrfBuffer[1]); // Output ISO15693 error code  UartPutCrlf();  // Response with error  ui8Status = STATUS\_FAIL;  }  }  **else**  {  // No response  ui8Status = STATUS\_FAIL;  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui8Status;  }  //===============================================================  //  // Iso15693\_ReadSingleBlockExtended - Issue the Read Single Block  // command for ISO15693 tags with the protocol extention flag set  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  // \param ui16BlockNumber is the block number to read data from.  //  // This function issues a Read Single Block with the block number  // and the specified request flags, including the Protocol  // Extension bit, to read data from ISO15693 tags which require  // the use of extended protocol commands.  //  // If UART is enabled, the data read from the ISO15693 tag is  // sent out to a host via UART as well.  //  // \return ui8Status returns either STATUS\_SUCCESS or STATUS\_FAIL  // to indicate if the Read Multiple Block was successful or not.  //  //===============================================================  uint8\_t Iso15693\_ReadSingleBlockExtended(uint8\_t ui8ReqFlag, uint16\_t ui16BlockNumber)  {  uint8\_t ui8Offset = 0;  uint8\_t ui8Status = STATUS\_FAIL;  uint8\_t ui8LoopCount = 1;  uint8\_t ui8RxLength = 0;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = 0xC0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = 0x40; // Length of packet in bytes - lower and broken nibbles of transmit byte length  }  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlag | 0x08; // ISO15693 flags with protocol extension bit set  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Read Single Block command code  **if** (ui8ReqFlag & 0x20)  {  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[0]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[1]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[2]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[3]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[4]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[5]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[6]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[7]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = (uint8\_t) (ui16BlockNumber & 0xFF); // Block # (variable, for this device it can go to 0xFF)  g\_ui8TrfBuffer[ui8Offset++] = (uint8\_t) ((ui16BlockNumber >> 8) & 0xFF); // Block # (variable, for this device it can go to 0x07)  }  **else**  {  g\_ui8TrfBuffer[ui8Offset++] = (uint8\_t) (ui16BlockNumber & 0xFF); // Block # (variable, for this device it can go to 0xFF)  g\_ui8TrfBuffer[ui8Offset++] = (uint8\_t) ((ui16BlockNumber >> 8) & 0xFF); // Block # (variable, for this device it can go to 0x07)  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Read Single Block command  Trf797xIrqWaitTimeout(3,10); // 3 millisecond TX timeout, 10 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  // Response received  ui8Status = STATUS\_SUCCESS;  UartSendCString("NFC-V Block ");  UartPutByte((ui16BlockNumber >> 8) & 0xFF); // Output block number  UartPutByte(ui16BlockNumber & 0xFF);  UartSendCString(" Data: ");  UartPutChar('[');  ui8RxLength = Trf797xGetRxBytesReceived();  **if** (ui8ReqFlag & BIT6) // Handle case for Option Flag causing one extra byte to be transmitted.  {  ui8Offset = 2;  }  **else**  {  ui8Offset = 1;  }  // Output received data to UART  **for** (ui8LoopCount = ui8Offset; ui8LoopCount < ui8RxLength; ui8LoopCount++)  {  UartPutByte(g\_ui8TrfBuffer[ui8LoopCount]); // Send out data read from tag to host  }  UartPutChar(']');  UartPutCrlf();  }  **else**  {  // Received an error from the tag  ui8Status = STATUS\_FAIL;  // Indicates when an error occurs or block addresses are unreachable - useful for debugging  UartSendCString("NFC-V Block ");  UartPutByte(((ui16BlockNumber >> 8) & 0xFF)); // Output block number  UartPutByte((ui16BlockNumber & 0xFF));  UartSendCString(" Error");  UartPutCrlf();  UartSendCString("ISO15693 Error Code: ");  UartPutByte(g\_ui8TrfBuffer[1]); // Output ISO15693 error code  UartPutCrlf();  }  }  **else**  {  // Did not receive a proper response from tag  ui8Status = STATUS\_FAIL;  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui8Status;  }  //===============================================================  //  // Iso15693\_WriteSingleBlock - Issue the Write Single Block  // command for ISO15693 tags.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  // \param ui8BlockNumber is the block number to write data to.  // \param ui8BlockSize is the tag block size.  // \param pui8BlockData is the data to be written.  //  // Function issues an addressed Write Single Block with the  // specified request flags as well as the Address flag. The  // write single block command will write the provided data  // packet to an ISO15693 tag. This command supports writing to  // tags with more than 4 bytes of data per block.  //  // \return ui8Status returns either STATUS\_SUCCESS or  // STATUS\_FAIL to indicate if the Write Single Block was  // successful or not.  //  //===============================================================  uint8\_t Iso15693\_WriteSingleBlock(uint8\_t ui8ReqFlag, uint8\_t ui8BlockNumber, uint8\_t ui8BlockSize, uint8\_t \* pui8BlockData)  {  uint8\_t ui8Offset = 0;  uint8\_t ui8Status = STATUS\_FAIL;  uint8\_t ui8LoopCount = 0;  **if** (Trf797xGetIsoControlValue() != 0x02)  {  // Trf797x has not been properly configured for ISO15693  Trf797xWriteIsoControl(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  }  **if** (Trf797xCheckRfField() == false)  {  // RF field is not enabled, VICC will not receive the command  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  // The VCD should wait at least 2mSec after activating the  // magnetic field before sending the first request, to  // ensure that the VICC is ready to receive it. (ISO15693-3)  McuDelayMillisecond(6);  }  ui8ReqFlag = ui8ReqFlag | 0x20; // Mandatory use of addressed writes  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = (((0x0B+ui8BlockSize) & 0xF0) >> 0x04); // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = ((0x0B+ui8BlockSize) << 0x04); // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = ui8ReqFlag; // ISO15693 flags  g\_ui8TrfBuffer[ui8Offset++] = 0x21; // Write Single Block command code  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[0]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[1]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[2]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[3]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[4]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[5]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[6]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = g\_pui8Iso15693UId[7]; // Tag UID  g\_ui8TrfBuffer[ui8Offset++] = ui8BlockNumber; // Block # (variable, for HF-I Plus device can go to 0x3F, Pro and Standard handled with "error" response flags)  **for** (ui8LoopCount = 0; ui8LoopCount < ui8BlockSize; ui8LoopCount++)  {  g\_ui8TrfBuffer[ui8Offset++] = pui8BlockData[ui8LoopCount]; // Data to write to tag  }  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the Get System Information command  // Check if the option flag is set  **if** (ui8ReqFlag & 0x40)  {  Trf797xIrqWaitTimeoutTxOnly(10); // 10 millisecond TX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == TX\_COMPLETE) // If transmit is complete  {  McuDelayMillisecond(5);  Trf797xTransmitNextSlot(); // Send out End of Frame marker  Trf797xIrqWaitTimeoutRxOnly(30); // 30 millisecond RX timeout  }  **else** // Otherwise return an error  {  **return** ui8Status = STATUS\_FAIL;  }  }  **else**  {  Trf797xIrqWaitTimeout(10,30); // 10 millisecond TX timeout, 30 millisecond RX timeout  }  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_sTrfStatus == RX\_COMPLETE) // If data has been received  {  **if** (g\_ui8TrfBuffer[0] == 0x00) // Confirm "no error" in response flags byte  {  // Response received  ui8Status = STATUS\_SUCCESS;  }  **else** // An error has been sent back in the response byte  {  ui8Status = STATUS\_FAIL;  }  }  **else**  {  // No response  ui8Status = STATUS\_FAIL;  }  // Clear any IRQs  Trf797xResetIrqStatus();  **return** ui8Status;  }  //===============================================================  //  // Iso15693\_Get\_Uid - Fetches the ISO15693 Tag UID.  //  // This function allows for higher layers to fetch the tag UID of  // an ISO15693 tag. In the current implementation, the UID  // stored is from the most recent tag which responded to an  // inventory command or the last tag to provide it's UID during  // the ISO15693 anticollision procedure.  //  // \return g\_pui8Iso15693UId returns the currently stored UID.  //  //===============================================================  uint8\_t \* Iso15693\_Get\_Uid(**void**)  {  **return** g\_pui8Iso15693UId;  }  //===============================================================  //  // Iso15693\_Get\_TagCount - Fetches the ISO15693 Tag UID.  //  // This function allows for higher layers to fetch the tag  // detected colunter in order to know how many ISO15693 tags  // have been detected since the last counter reset.  //  // \return g\_ui8TagDetectedCount returns the count of ISO15693  // tags detected.  //  //===============================================================  uint8\_t Iso15693\_Get\_TagCount(**void**)  {  **return** g\_ui8TagDetectedCount;  }  //===============================================================  //  // Iso15693\_Reset\_TagCount - Fetches the ISO15693 Tag UID.  //  // This function allows for higher layers to reset the tag  // detected counter.  //  // \return None  //  //===============================================================  **void** Iso15693\_Reset\_TagCount(**void**)  {  g\_ui8TagDetectedCount = 0;  }  //===============================================================  //  // Nfc\_Iso15693\_ReadTag - Read all blocks of a ISO15693 tag.  //  // \param ui8ReqFlag are the request flags for ISO15693 commands.  //  // This function issues Get System Information command to  // determine how many blocks of data is stored within the  // ISO15693 tag.  // Afterwards, all blocks are read out using a Read Single block,  // unless an error occurs during the read process at which point  // the function will stop reading data and return.  //  // \return None.  //  //===============================================================  **void** Nfc\_Iso15693\_ReadTag(uint8\_t ui8ReqFlag)  {  uint16\_t ui16ReadBlocks = 0x00;  uint16\_t ui16LoopCount = 0x00;  ui16ReadBlocks = Iso15693\_GetSystemInfo(ui8ReqFlag); // Get Tag Information with Request Flag = 0x02  **if** (ui16ReadBlocks != 0x00)  {  // Read all available blocks on the ISO15693 Tag  **for** (ui16LoopCount = 0; ui16LoopCount < ui16ReadBlocks+1; ui16LoopCount++)  {  **if** (Iso15693\_ReadSingleBlock(ui8ReqFlag, ui16LoopCount) == STATUS\_FAIL) // Keep reading blocks unless a No Response is received  {  // No Response - stop reading  **break**;  }  }  }  }  **void** ISO15693FindTag(**void**)  {  uint8\_t ui8TagFound = STATUS\_FAIL;  uint8\_t ui8AddressedFlag = 0x00;  Trf797xTurnRfOn(); // Ensure TRF797x is outputting an RF Field  Trf797xWriteInitiatorSetup(0x02); // Configure the TRF797x for ISO15693 @ High Bit Rate, One Subcarrier, 1 out of 4  IRQ\_CLR; // PORT2 interrupt flag clear  IRQ\_ON;  // The VCD should wait at least 1 ms after it activated the  // powering field before sending the first request, to  // ensure that the VICCs are ready to receive it. (ISO15693-3)  osi\_Sleep(10);  //McuDelayMillisecond(20);  Iso15693\_Reset\_TagCount();  ui8TagFound = Iso15693\_SingleSlotInventory(); // Send a single slot inventory request to try and detect a single ISO15693 Tag  // Inventory failed - search with full anticollision routine  **if** (ui8TagFound == STATUS\_FAIL)  {  McuDelayMillisecond(5); // Delay before issuing the anticollision commmand  ui8TagFound = Iso15693\_Anticollision(0x06, 0x00, 0x00); // Send 16 Slot Inventory request with no mask length and no AFI  ui8AddressedFlag = 0x20; // Collision occurred, send addressed commands  }  **if** (ui8TagFound == STATUS\_SUCCESS)  {  **if** (Iso15693\_Get\_TagCount() > 1)  {  UartPutCrlf();  UartSendCString("Multiple ISO15693 Tags Found.");  UartPutCrlf();  UartSendCString("Number of ISO15693 Tags Detected: ");  UartPutChar('0' + Iso15693\_Get\_TagCount());  UartPutCrlf();  UartSendCString("Please place only 1 tag in the RF Field at a time to read tag data.");  UartPutCrlf();  }  **else**  {  Nfc\_Iso15693\_ReadTag(0x02 | ui8AddressedFlag); // Read an ISO15693 tag  }  }  **else**  {  }  Trf797xTurnRfOff(); // Turn off RF field once done reading the tag(s)  } |

|  |
| --- |
| **spi\_for\_trf.h** |
| /\*  \* {spi\_for\_trf.h}  \*  \* {Header File}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_SPI\_TRF\_H\_  **#define** \_SPI\_TRF\_H\_  **#include** "types.h"  **#include** "hw\_types.h"  //================================================================  //===============================================================  **extern** **void** **SpiDirectCommand**(u08\_t \*pbuf);  **extern** **void** **SpiDirectMode**(**void**);  **extern** **void** **SpiRawWrite**(u08\_t \*pbuf, u08\_t length);  **extern** **void** **SpiReadCont**(u08\_t \*pbuf, u08\_t length);  **extern** **void** **SpiReadSingle**(u08\_t \*pbuf, u08\_t number);  **extern** **void** **SpiWriteCont**(u08\_t \*pbuf, u08\_t length);  **extern** **void** **SpiWriteSingle**(u08\_t \*pbuf, u08\_t length);  //===============================================================  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPIConfigSetExpClk() as ulMode parameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_MODE\_MASTER 0x00000000  **#define** SPI\_MODE\_SLAVE 0x00000004  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPIConfigSetExpClk() as ulSubMode parameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_SUB\_MODE\_0 0x00000000  **#define** SPI\_SUB\_MODE\_1 0x00000001  **#define** SPI\_SUB\_MODE\_2 0x00000002  **#define** SPI\_SUB\_MODE\_3 0x00000003  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPIConfigSetExpClk() as ulConfigFlags parameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_SW\_CTRL\_CS 0x01000000  **#define** SPI\_HW\_CTRL\_CS 0x00000000  **#define** SPI\_3PIN\_MODE 0x02000000  **#define** SPI\_4PIN\_MODE 0x00000000  **#define** SPI\_TURBO\_ON 0x00080000  **#define** SPI\_TURBO\_OFF 0x00000000  **#define** SPI\_CS\_ACTIVEHIGH 0x00000000  **#define** SPI\_CS\_ACTIVELOW 0x00000040  **#define** SPI\_WL\_8 0x00000380  **#define** SPI\_WL\_16 0x00000780  **#define** SPI\_WL\_32 0x00000F80  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPIFIFOEnable() and SPIFIFODisable()  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_TX\_FIFO 0x08000000  **#define** SPI\_RX\_FIFO 0x10000000  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPIDMAEnable() and SPIDMADisable()  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_RX\_DMA 0x00008000  **#define** SPI\_TX\_DMA 0x00004000  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPIIntEnable(), SPIIntDiasble(),  // SPIIntClear() or returned from SPIStatus()  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_INT\_DMATX 0x20000000  **#define** SPI\_INT\_DMARX 0x10000000  **#define** SPI\_INT\_EOW 0x00020000  **#define** SPI\_INT\_WKS 0x00010000  **#define** SPI\_INT\_RX\_OVRFLOW 0x00000008  **#define** SPI\_INT\_RX\_FULL 0x00000004  **#define** SPI\_INT\_TX\_UDRFLOW 0x00000002  **#define** SPI\_INT\_TX\_EMPTY 0x00000001  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Values that can be passed to SPITransfer()  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** SPI\_CS\_ENABLE 0x00000001  **#define** SPI\_CS\_DISABLE 0x00000002  **#endif** |

|  |
| --- |
| **spi\_for\_trf.c** |
| /\*  \* {spi\_for\_trf.c}  \*  \* {SPI Interface Functions}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#include** "hw\_types.h"  **#include** "spi.h"  **#include** "spi\_for\_trf.h"  **#include** "gpio\_if.h"  **#include** "trf797x.h"  **#include** "hw\_memmap.h"  **#include** "hw\_common\_reg.h"  **#include** "hw\_ints.h"  **#include** "pin.h"  **#include** "rom.h"  **#include** "rom\_map.h"  **#include** "utils.h"  **#include** "prcm.h"  **#include** "interrupt.h"  **#include** "pinmux.h"  **#include** "uart.h"  **#include** "uart\_if.h"  **#include** "types.h"  **#include** "trf7970BoosterPack.h"  **#define** ASSERT\_CS() (SPI\_CS\_ON)  **#define** DEASSERT\_CS() (SPI\_CS\_OFF)  **typedef** **unsigned** **int** Fd\_t;  //===============================================================  u08\_t temp = 0;  **extern** u08\_t direct\_mode;  //===============================================================  **void** SpiStartCondition(**void**);  **void** SpiStopCondition(**void**);  **void** SpiUsciSet(**void**);  **void** SpiUsciDisable(**void**);  //===============================================================  // NAME: void SpiDirectCommand (u08\_t \*pbuf)  //  // BRIEF: Is used in SPI mode to transmit a Direct Command to  // reader chip.  //  // INPUTS:  // Parameters:  // u08\_t \*pbuf Direct Command  //  // OUTPUTS:  //  // PROCESS: [1] transmit Direct Command  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiDirectCommand(u08\_t \*pbuf)  {  **unsigned** **char** ucDummy;  // set Address/Command Word Bit Distribution to command  \*pbuf = (0x80 | \*pbuf); // command  \*pbuf = (0x9f & \*pbuf); // command code  SPI\_CS\_OFF;  SPITransfer(GSPI\_BASE, pbuf, &ucDummy, 1, SPI\_CS\_ENABLE);  SPI\_CS\_ON;  }  //===============================================================  // NAME: void SpiDirectMode (void)  //  // BRIEF: Is used in SPI mode to start Direct Mode.  //  // INPUTS:  //  // OUTPUTS:  //  // PROCESS: [1] start Direct Mode  //  // NOTE: No stop condition  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiDirectMode(**void**)  {  u08\_t command [2];    command[0] = TRF797x\_STATUS\_CONTROL;  command[1] = TRF797x\_STATUS\_CONTROL;  // read byte to command[1]  SPI\_CS\_OFF;  SPITransfer(GSPI\_BASE, 0, &command[1], 1, SPI\_CS\_ENABLE);  command[1] |= 0x60; // RF on and BIT 6 in Chip Status Control Register set  // write command[0] byte  SPITransfer(GSPI\_BASE, command, 0, 1, SPI\_CS\_ENABLE);  SpiWriteSingle(command, 2);  SPI\_CS\_ON;  }  //===============================================================  // NAME: void SpiRawWrite (u08\_t \*pbuf, u08\_t length)  //  // BRIEF: Is used in SPI mode to write direct to the reader chip.  //  // INPUTS:  // Parameters:  // u08\_t \*pbuf raw data  // u08\_t length number of data bytes  //  // OUTPUTS:  //  // PROCESS: [1] send raw data to reader chip  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiRawWrite(u08\_t \*pbuf, u08\_t length)  {  SPI\_CS\_OFF;  SPITransfer(GSPI\_BASE, pbuf, 0, length, SPI\_CS\_ENABLE);  SPI\_CS\_ON;  }  //===============================================================  // NAME: void SpiReadCont (u08\_t \*pbuf, u08\_t length)  //  // BRIEF: Is used in SPI mode to read a specified number of  // reader chip registers from a specified address upwards.  //  // INPUTS:  // Parameters:  // u08\_t \*pbuf address of first register  // u08\_t length number of registers  //  // OUTPUTS:  //  // PROCESS: [1] read registers  // [2] write contents to \*pbuf  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiReadCont(u08\_t \*pbuf, u08\_t length)  {    // Address/Command Word Bit Distribution  \*pbuf = (0x60 | \*pbuf); // address, read, continuous  \*pbuf = (0x7f &\*pbuf); // register address  // write command address/word  SPI\_CS\_OFF;  SPITransfer(GSPI\_BASE, pbuf, 0, 1, SPI\_CS\_ENABLE);  // read length bytes into pbuf[length]  SPITransfer(GSPI\_BASE, 0, pbuf, length, SPI\_CS\_ENABLE);  SPI\_CS\_ON;    }  //===============================================================  // NAME: void SpiReadSingle (u08\_t \*pbuf, u08\_t number)  //  // BRIEF: Is used in SPI mode to read specified reader chip  // registers.  //  // INPUTS:  // Parameters:  // u08\_t \*pbuf addresses of the registers  // u08\_t number number of the registers  //  // OUTPUTS:  //  // PROCESS: [1] read registers  // [2] write contents to \*pbuf  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiReadSingle(u08\_t \*pbuf, u08\_t number)  {  SPI\_CS\_OFF;  **while**(number > 0)  {  // Address/Command Word Bit Distribution  \*pbuf = (0x40 | \*pbuf); // address, read, single  \*pbuf = (0x5f & \*pbuf); // register address    // write address/command byte into pbuf  SPITransfer(GSPI\_BASE, pbuf, 0, 1, SPI\_CS\_ENABLE);  // read single byte into pbuf  SPITransfer(GSPI\_BASE, 0, pbuf, 1, SPI\_CS\_ENABLE);    pbuf++;  number--;    }  SPI\_CS\_ON;  }  //===============================================================  // NAME: void SpiWriteCont (u08\_t \*pbuf, u08\_t length)  //  // BRIEF: Is used in SPI mode to write to a specific number of  // reader chip registers from a specific address upwards.  //  // INPUTS:  // u08\_t \*pbuf address of first register followed by the  // contents to write  // u08\_t length number of registers + 1  //  // OUTPUTS:  //  // PROCESS: [1] write to the registers  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiWriteCont(u08\_t \*pbuf, u08\_t length)  {  SPI\_CS\_OFF;  // Address/Command Word Bit Distribution  \*pbuf = (0x20 | \*pbuf); // address, write, continuous  \*pbuf = (0x3f &\*pbuf); // register address  // write command address/word + number of registers  SPITransfer(GSPI\_BASE, pbuf, 0, length, SPI\_CS\_ENABLE);  SPI\_CS\_ON;  }  //===============================================================  // NAME: void SpiWriteSingle (u08\_t \*pbuf, u08\_t length)  //  // BRIEF: Is used in SPI mode to write to a specified reader chip  // registers.  //  // INPUTS:  // u08\_t \*pbuf addresses of the registers followed by the  // contends to write  // u08\_t length number of registers \* 2  //  // OUTPUTS:  //  // PROCESS: [1] write to the registers  //  // CHANGE:  // DATE WHO DETAIL  // 24Nov2010 RP Original Code  // 07Dec2010 RP integrated wait while busy loops  //===============================================================  **void**  SpiWriteSingle(u08\_t \*pbuf, u08\_t length)  {    SPI\_CS\_OFF;  **while**(length > 0)  {  // Address/Command Word Bit Distribution  // address, write, single (fist 3 bits = 0)  \*pbuf = (0x1f &\*pbuf); // register address  SPITransfer(GSPI\_BASE, pbuf, 0, 2, SPI\_CS\_ENABLE);  pbuf=pbuf + 2;  length=length - 2;  }  SPI\_CS\_ON;  } |

|  |
| --- |
| **trf7970BoosterPack.h** |
| /\*  \* {trf7970BoosterPack.h}  \*  \* {Header File}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_Trf7970BOOSTERPACK\_H\_  **#define** \_Trf7970BOOSTERPACK\_H\_  //================================================================  **#include** <stdint.h>  **#include** "types.h"  **#include** "gpio\_if.h"  **#define** \_BOARD\_H  /\* This macro is for use by other macros to form a fully valid C statement. \*/  **#define** st(x) **do** { x } **while** (\_\_LINE\_\_ == -1)  **#define** COUNT\_1ms 80000  **#define** IRQ\_CLR GPIOIntClear(GPIOA1\_BASE, GPIO\_PIN\_4); //GPIO 7  **#define** IRQ\_ON GPIOIntEnable(GPIOA1\_BASE,GPIO\_PIN\_4);  **#define** IRQ\_OFF GPIOIntDisable(GPIOA1\_BASE,GPIO\_PIN\_4);  **#define** START\_COUNTER A2CounterEnable();  **#define** RESET\_COUNTER A2CounterDisable();  **#define** STOP\_COUNTER A2CounterStop();  **#define** TRF\_OFF GPIO\_IF\_Set(11,0);  **#define** TRF\_ON GPIO\_IF\_Set(11,1);  **#define** SPI\_CS\_ON MAP\_SPICSEnable(GSPI\_BASE);  **#define** SPI\_CS\_OFF MAP\_SPICSDisable(GSPI\_BASE);  **void** Delay(**unsigned** **long** interval);  **void** McuDelayMillisecond(**unsigned** **int** ui\_delay\_in\_msec);  //===============================================================  **#define** TRIGGER 0 // if TRIGGER 1 trigger-point at LED 5  //=====MCU constants=============================================  // IRQ on P2.0  // rising edge interrupt  **#define** IRQ\_PIN BIT7  **#define** IRQ\_PORT HWREG(GPIOA1\_BASE + GPIO\_O\_GPIO\_RIS)  // Hard code to only SPI mode  **#define** SPIMODE 1  // CSn  //-----Counter-timer constants-----------------------------------  //---------------------------------------------------------------  // Set timer capture register value based on clock source  //---------------------------------------------------------------  // 25MHz clock  //===============================================================  **#endif** |

|  |
| --- |
| **trf7970BoosterPack.c** |
| /\*  \* {trf7970BoosterPack.c}  \*  \* {MSP430G2553 specific functions}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#include** "trf797x.h"  **#include** "trf7970BoosterPack.h"  **#include** "utils.h"  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! Delay  //!  //! \param interval is time in milli second  //!  //! \return none  //!  //! \brief  //  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** Delay(**unsigned** **long** interval)  {  **while**(interval > 0)  {  UtilsDelay(80000);  interval--;  }  }  **void** McuDelayMillisecond(**unsigned** **int** ui\_delay\_in\_msec){  UtilsDelay(80000\*ui\_delay\_in\_msec);  } |

|  |
| --- |
| **trf797x.h** |
| /\*  \* File Name: trf797x.h  \*  \* Description: Headers and Defines for TRF797x Driver Functions  \*  \* Copyright (C) 2016 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_TRF797x\_H\_  **#define** \_TRF797x\_H\_  //===============================================================  **#include** "spi\_for\_trf.h"  **#include** "uart.h"  **#include** "uart\_if.h"  **#include** "types.h"  **#define** ENABLE\_STANDALONE  **#define** NFC\_FIFO\_SIZE 120  //===============================================================  //==== TRF797x definitions ======================================  //---- Direct commands ------------------------------------------  **#define** IDLE 0x00  **#define** SOFT\_INIT 0x03  **#define** INITIAL\_RF\_COLLISION 0x04  **#define** RESPONSE\_RF\_COLLISION\_N 0x05  **#define** RESPONSE\_RF\_COLLISION\_0 0x06  **#define** RESET 0x0F  **#define** TRANSMIT\_NO\_CRC 0x10  **#define** TRANSMIT\_CRC 0x11  **#define** DELAY\_TRANSMIT\_NO\_CRC 0x12  **#define** DELAY\_TRANSMIT\_CRC 0x13  **#define** TRANSMIT\_NEXT\_SLOT 0x14  **#define** CLOSE\_SLOT\_SEQUENCE 0x15  **#define** STOP\_DECODERS 0x16  **#define** RUN\_DECODERS 0x17  **#define** CHECK\_INTERNAL\_RF 0x18  **#define** CHECK\_EXTERNAL\_RF 0x19  **#define** ADJUST\_GAIN 0x1A  //---- Registers ------------------------------------------------  **#define** TRF797x\_STATUS\_CONTROL 0x00  **#define** TRF797x\_ISO\_CONTROL 0x01  **#define** TRF797x\_ISO\_14443\_TX\_OPTIONS 0x02  **#define** TRF797x\_ISO\_14443\_BITRATE\_OPTIONS 0x03  **#define** TRF797x\_TX\_TIMER\_EPC\_HIGH 0x04  **#define** TRF797x\_TX\_TIMER\_EPC\_LOW 0x05  **#define** TRF797x\_TX\_PULSE\_LENGTH\_CONTROL 0x06  **#define** TRF797x\_RX\_NO\_RESPONSE\_WAIT\_TIME 0x07  **#define** TRF797x\_RX\_WAIT\_TIME 0x08  **#define** TRF797x\_MODULATOR\_CONTROL 0x09  **#define** TRF797x\_RX\_SPECIAL\_SETTINGS 0x0A  **#define** TRF797x\_REGULATOR\_CONTROL 0x0B  **#define** TRF797x\_IRQ\_STATUS 0x0C // IRQ Status Register  **#define** TRF797x\_IRQ\_MASK 0x0D // Collision Position and Interrupt Mask Register  **#define** TRF797x\_COLLISION\_POSITION 0x0E  **#define** TRF797x\_RSSI\_LEVELS 0x0F  **#define** TRF797x\_SPECIAL\_FUNCTION\_1 0x10  **#define** TRF797x\_SPECIAL\_FUNCTION\_2 0x11  **#define** TRF797x\_FIFO\_IRQ\_LEVELS 0x14  **#define** TRF797x\_NFC\_LOW\_DETECTION 0x16  **#define** TRF797x\_NFC\_TARGET\_LEVEL 0x18  **#define** TRF797x\_NFC\_TARGET\_PROTOCOL 0x19  **#define** TRF797x\_TEST\_SETTINGS\_1 0x1A  **#define** TRF797x\_TEST\_SETTINGS\_2 0x1B  **#define** TRF797x\_FIFO\_STATUS 0x1C  **#define** TX\_LENGTH\_BYTE\_1 0x1D  **#define** TX\_LENGTH\_BYTE\_2 0x1E  **#define** FIFO 0x1F  //---- IRQ STATUS -----------------------------------------------  **#define** TRF797x\_IRQ\_STATUS\_IDLE 0x00  **#define** TRF797x\_IRQ\_STATUS\_NO\_RESPONSE 0x01  **#define** TRF797x\_IRQ\_STATUS\_COLLISION\_ERROR 0x02  **#define** TRF797x\_IRQ\_STATUS\_FRAMING\_ERROR 0x04  **#define** TRF797x\_IRQ\_STATUS\_PARITY\_ERROR 0x08  **#define** TRF797x\_IRQ\_STATUS\_CRC\_ERROR 0x10  **#define** TRF797x\_IRQ\_STATUS\_FIFO\_HIGH\_OR\_LOW 0x20  **#define** TRF797x\_IRQ\_STATUS\_RX\_COMPLETE 0x40  **#define** TRF797x\_IRQ\_STATUS\_TX\_COMPLETE 0x80  //==== End of TRF797x definitions ===============================  //===============================================================  **typedef** **enum**  {  *TRF\_IDLE*,  *RX\_COMPLETE*,  *TX\_COMPLETE*,  *TX\_ERROR*,  *RX\_WAIT*,  *RX\_WAIT\_EXTENSION*,  *TX\_WAIT*,  *PROTOCOL\_ERROR*,  *NO\_RESPONSE\_RECEIVED*,  *NO\_RESPONSE\_RECEIVED\_15693*  }tTRF797x\_Status;  //===============================================================  **extern** **void** **Trf797xCommunicationSetup**(**void**);  **extern** **void** **Trf797xDirectCommand**(u08\_t \*ui8Value);  **extern** **void** **Trf797xDisableSlotCounter**(**void**);  **extern** **void** **Trf797xEnableSlotCounter**(**void**);  **extern** **void** **Trf797xInitialSettings**(**void**);  **extern** **void** **Trf797xRawWrite**(u08\_t \* pui8Payload, u08\_t ui8Length);  **extern** **void** **Trf797xReConfig**(**void**);  **extern** **void** **Trf797xReadCont**(u08\_t \* pui8Payload, u08\_t ui8Length);  **extern** u08\_t **Trf797xReadIsoControl**(**void**);  **extern** **void** **Trf797xReadIrqStatus**(u08\_t \* pui8Value);  **extern** **void** **Trf797xReadSingle**(u08\_t \*pbuf, u08\_t number);  **extern** u08\_t **Trf797xReadStatusControl**(**void**);  **extern** **void** **Trf797xReset**(**void**);  **extern** **void** **Trf797xResetIrqStatus**(**void**);  **extern** **void** **Trf797xRunDecoders**(**void**);  **extern** **void** **Trf797xStopDecoders**(**void**);  **extern** **void** **Trf797xTransmitNextSlot**(**void**);  **extern** **void** **Trf797xTurnRfOff**(**void**);  **extern** **void** **Trf797xTurnRfOn**(**void**);  **extern** **void** **Trf797xWriteCont**(u08\_t \* pui8Payload, u08\_t ui8Length);  **extern** **void** **Trf797xWriteIsoControl**(u08\_t ui8IsoControl);  **extern** **void** **Trf797xWriteRegister**(u08\_t ui8TrfRegister, u08\_t ui8Value);  **extern** **void** **Trf797xWriteInitiatorSetup**(u08\_t ui8IsoControl);  **extern** **void** **Trf797xWriteSingle**(u08\_t \*pbuf, u08\_t length);  **extern** **void** **Trf797xIrqWaitTimeout**(**long** ui8TxTimeout, **long** ui8RxTimeout);  **extern** **void** **Trf797xIrqWaitTimeoutTxOnly**(u08\_t ui8TxTimeout);  **extern** **void** **Trf797xIrqWaitTimeoutRxOnly**(u08\_t ui8RxTimeout);  **extern** **void** **Trf797xIrqWaitTimeoutFeliCa**(**void**);  **extern** tTRF797x\_Status **Trf797xGetTrfStatus**(**void**);  **extern** **void** **Trf797xSetTrfStatus**(tTRF797x\_Status sTrfStatus);  **extern** u08\_t **Trf797xGetCollisionPosition**(**void**);  **extern** **void** **Trf797xSetCollisionPosition**(u08\_t ui8ColPos);  **extern** u08\_t **Trf797xGetRxBytesReceived**(**void**);  **extern** u08\_t **Trf797xGetIsoControlValue**(**void**);  **extern** u08\_t **Trf797xReadRssiLevels**(**void**);  **extern** bool **Trf797xCheckRfField**(**void**);  **extern** **void** **A2CNTIntHandler** (**void**);  **extern** **void** **Trf797xIRQ**(**void**);  **extern** **void** **StatusPrint**(tTRF797x\_Status sTrfStatus);  **extern** u08\_t g\_ui8FifoRxLength;  //===============================================================  **#endif** |

|  |
| --- |
| **trf797x.c** |
| /\*  \* File Name: trf797x.c  \*  \* Description: TRF797x Driver Functions  \*  \* Copyright (C) 2016 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#include** <string.h>  **#include** <stdio.h>  **#include** "hw\_types.h"  **#include** "hw\_memmap.h"  **#include** "trf797x.h"  **#include** "trf7970BoosterPack.h"  **#include** "timer\_if.h"  **#include** "gpio.h"  **#include** "osi.h"  //===============================================================  u08\_t g\_ui8TrfBuffer[NFC\_FIFO\_SIZE];  **static** u08\_t g\_ui8CollisionPosition = 0;  **static** u08\_t g\_ui8FifoOffset = 0;  u08\_t g\_ui8FifoRxLength = 0;  **static** bool g\_bRfFieldOn = false;  **static** u08\_t g\_ui8IsoControlValue = 0x00;  **static** tTRF797x\_Status g\_sTrf797xStatus = TRF\_IDLE;  **static** **volatile** u08\_t g\_ui8IrqFlag = 0x00;  **static** **volatile** u08\_t g\_ui8TimeoutFlag = 0x00;  u08\_t command[2];  //===============================================================  //  // Trf797xCommunicationSetup - Initialize Communication to TRF797x  //  // Call the appropriate serial communication initialize based on  // hardware setup  //  // \return None.  //  //===============================================================  **void**  Trf797xCommunicationSetup(**void**)  {  A2CounterInit(A2CNTIntHandler);  RESET\_COUNTER;  STOP\_COUNTER;  command[0] = SOFT\_INIT;  Trf797xDirectCommand(command);  command[0] = IDLE;  Trf797xDirectCommand(command);  command[0] = TRF797x\_MODULATOR\_CONTROL;  command[1] = 0x01; // ASK 100%, no SYS\_CLK output  Trf797xWriteSingle(command, 2);  command[0] = TRF797x\_NFC\_TARGET\_LEVEL;  command[1] = 0x00;  Trf797xWriteSingle(command, 2);  }  //===============================================================  //  // Trf797xDirectCommand - Send a Direct Command to TRF797x.  //  // \param ui8Value is the direct command to be issued  //  // Function is used to transmit a Direct Command to TRF797x.  //  // \return None.  //  //===============================================================  **void**  Trf797xDirectCommand(u08\_t \*ui8Value)  {  SpiDirectCommand(ui8Value);  }  //===============================================================  //  // Trf797xDisableSlotCounter - Disables interrupts from 15693  // slot counter function.  //  // Turns off the firing of the IRQ interrupt when a 15693 slot  // hits the No Response timeout  //  // \return None.  //  //===============================================================  **void**  Trf797xDisableSlotCounter(**void**)  {  u08\_t pui8ModControl[2];  pui8ModControl[0] = TRF797x\_IRQ\_MASK; // next slot counter  pui8ModControl[1] = TRF797x\_IRQ\_MASK;  Trf797xReadSingle(&pui8ModControl[1], 1);  pui8ModControl[1] &= 0xFE; // clear BIT0 in register 0x01  Trf797xWriteSingle(pui8ModControl, 2);  }  //===============================================================  //  // Trf797xEnableSlotCounter - Enables interrupts from 15693  // slot counter function.  //  // Turns on the firing of the IRQ interrupt when a 15693 slot  // hits the No Response timeout  //  // \return None.  //  //===============================================================  **void**  Trf797xEnableSlotCounter(**void**)  {  u08\_t pui8ModControl[2];  pui8ModControl[0] = TRF797x\_IRQ\_MASK; // next slot counter  pui8ModControl[1] = TRF797x\_IRQ\_MASK;  Trf797xReadSingle(&pui8ModControl[1],1);  pui8ModControl[1] |= BIT0; // set BIT0 in register 0x01  Trf797xWriteSingle(pui8ModControl,2);  }  //===============================================================  //  // Trf797xEnableSlotCounter - Initialize TRF797x  //  // Handles setting up the clock register and writes register 0x18  // as per TRF7970A Errata  //  // \return None.  //  //===============================================================  **void**  Trf797xInitialSettings(**void**)  {  command[0] = SOFT\_INIT;  Trf797xDirectCommand(command);  command[0] = IDLE;  Trf797xDirectCommand(command);  command[0] = TRF797x\_MODULATOR\_CONTROL;  command[1] = 0x01; // ASK 100%, no SYS\_CLK output  Trf797xWriteSingle(command,2);  command[0] = TRF797x\_NFC\_TARGET\_LEVEL;  command[1] = 0x00;  Trf797xWriteSingle(command,2);  }  //===============================================================  //  // Trf797xISR - Services TRF797x IRQ interrupts  //  // \param pui8\_IrqStatus is the received IRQ Status flags  //  // The Interrupt Service Routine determines how the IRQ should  // be handled. The Trf797x IRQ status register is read to  // determine the cause of the IRQ. Conditions are checked and  // appropriate actions taken.  //  // \return None.  //  //===============================================================  **void**  Trf797xISR(u08\_t \* pui8\_IrqStatus)  {  u08\_t ui8DummyRead;  u08\_t ui8Length;  **if**(\*pui8\_IrqStatus == (TRF797x\_IRQ\_STATUS\_TX\_COMPLETE | TRF797x\_IRQ\_STATUS\_FIFO\_HIGH\_OR\_LOW)) // BIT5 and BIT7  { // TX active and 32 bytes left in FIFO  g\_sTrf797xStatus = TX\_COMPLETE;  }  **else** **if**(\*pui8\_IrqStatus == TRF797x\_IRQ\_STATUS\_TX\_COMPLETE)  { // TX complete  g\_sTrf797xStatus = TX\_COMPLETE;  Trf797xReset(); // reset the FIFO after TX  }  **else** **if**((\*pui8\_IrqStatus & BIT1) == TRF797x\_IRQ\_STATUS\_COLLISION\_ERROR)  { // Collision error  g\_sTrf797xStatus = PROTOCOL\_ERROR;  **if** ((g\_ui8IsoControlValue == 0x08) || (g\_ui8IsoControlValue == 0x88))  {  g\_ui8CollisionPosition = TRF797x\_COLLISION\_POSITION;  Trf797xReadSingle(&g\_ui8CollisionPosition, 1);  **if** (g\_ui8CollisionPosition > 0x20)  {  ui8Length = g\_ui8CollisionPosition - 0x20; // number of valid bytes in FIFO  **if**((ui8Length & 0x0F) != 0x00)  {  ui8Length = ui8Length + 0x10; // add 1 byte if broken byte recieved  }  ui8Length = ui8Length >> 4;  **if**(ui8Length != 0x00)  {  g\_ui8TrfBuffer[g\_ui8FifoOffset] = FIFO; // write the recieved bytes to the correct place of the buffer  Trf797xReadCont(&g\_ui8TrfBuffer[g\_ui8FifoOffset], ui8Length);  g\_ui8FifoOffset = g\_ui8FifoOffset + ui8Length;  }  }  **else**  {  g\_ui8FifoRxLength = TRF797x\_FIFO\_STATUS;  Trf797xReadSingle(&g\_ui8FifoRxLength,1); // determine the number of bytes left in FIFO  g\_ui8FifoRxLength = 0x7F & g\_ui8FifoRxLength;  g\_ui8TrfBuffer[g\_ui8FifoOffset] = FIFO; // write the recieved bytes to the correct place of the buffer  Trf797xReadCont(&g\_ui8TrfBuffer[g\_ui8FifoOffset], g\_ui8FifoRxLength);  g\_ui8FifoOffset = g\_ui8FifoOffset + g\_ui8FifoRxLength;  }  }  Trf797xStopDecoders();  Trf797xReset(); // reset the FIFO after TX  Trf797xResetIrqStatus();  IRQ\_CLR;  }  **else** **if**(\*pui8\_IrqStatus == TRF797x\_IRQ\_STATUS\_RX\_COMPLETE || \*pui8\_IrqStatus == 0xC0)  {  //10-20-2016 Fix  //TRF7970A returns C0 (TX\_IRQ = 1, RX\_IRQ = 1) during selection command  //Really strange, but reading from fifo right afterwards does work for some reason...  // RX flag means that EOF has been recieved  // and the number of unread bytes is in FIFOstatus regiter  g\_ui8FifoRxLength = TRF797x\_FIFO\_STATUS;  Trf797xReadSingle(&g\_ui8FifoRxLength, 1); // determine the number of bytes left in FIFO  g\_ui8FifoRxLength = 0x7F & g\_ui8FifoRxLength;  g\_ui8TrfBuffer[g\_ui8FifoOffset] = FIFO; // write the recieved bytes to the correct place of the buffer  Trf797xReadCont(&g\_ui8TrfBuffer[g\_ui8FifoOffset], g\_ui8FifoRxLength);  g\_ui8FifoOffset = g\_ui8FifoOffset + g\_ui8FifoRxLength;  Trf797xReset(); // reset the FIFO after last byte has been read out  **if** (g\_sTrf797xStatus == RX\_WAIT\_EXTENSION)  {  g\_ui8FifoRxLength = g\_ui8FifoOffset;  }  g\_sTrf797xStatus = RX\_COMPLETE;  }  **else** **if**(\*pui8\_IrqStatus == (TRF797x\_IRQ\_STATUS\_RX\_COMPLETE | TRF797x\_IRQ\_STATUS\_FIFO\_HIGH\_OR\_LOW))  { // RX active and 96 bytes allready in FIFO  g\_sTrf797xStatus = RX\_WAIT;  // Read FIFO Status to determine how many bytes are in the FIFO  g\_ui8FifoRxLength = TRF797x\_FIFO\_STATUS;  Trf797xReadSingle(&g\_ui8FifoRxLength, 1);  g\_ui8FifoRxLength = 0x7F & g\_ui8FifoRxLength;  **if** (NFC\_FIFO\_SIZE > (g\_ui8FifoOffset+g\_ui8FifoRxLength))  {  // Read from the FIFO to empty it  g\_ui8TrfBuffer[g\_ui8FifoOffset] = FIFO;  Trf797xReadCont(&g\_ui8TrfBuffer[g\_ui8FifoOffset], g\_ui8FifoRxLength); // read all received bytes from FIFO  g\_ui8FifoOffset = g\_ui8FifoOffset + g\_ui8FifoRxLength; // Adjust buffer index  }  **else**  {  g\_sTrf797xStatus = PROTOCOL\_ERROR;  **return**;  }  // Read FIFO Status again to determine if more bytes have been received  g\_ui8FifoRxLength = TRF797x\_FIFO\_STATUS;  Trf797xReadSingle(&g\_ui8FifoRxLength, 1); // determine the number of bytes left in FIFO  g\_ui8FifoRxLength = 0x7F & g\_ui8FifoRxLength;  **if** (g\_ui8FifoRxLength > 0)  {  g\_sTrf797xStatus = RX\_WAIT\_EXTENSION;  }  **else**  {  g\_ui8FifoRxLength = g\_ui8FifoOffset;  g\_sTrf797xStatus = RX\_COMPLETE;  **return**;  }  }  **else** **if** (\*pui8\_IrqStatus == (TRF797x\_IRQ\_STATUS\_RX\_COMPLETE | TRF797x\_IRQ\_STATUS\_NO\_RESPONSE))  {  // RX has begun but as not completed, space exists in FIFO still, just wait longer to receive full reply.  g\_sTrf797xStatus = RX\_WAIT\_EXTENSION;  }  **else** **if**((\*pui8\_IrqStatus & BIT4) == TRF797x\_IRQ\_STATUS\_CRC\_ERROR) // CRC error  {  **if**((\*pui8\_IrqStatus & BIT5) == TRF797x\_IRQ\_STATUS\_FIFO\_HIGH\_OR\_LOW)  {  g\_sTrf797xStatus = RX\_WAIT;  }  **if**((\*pui8\_IrqStatus & BIT6) == TRF797x\_IRQ\_STATUS\_RX\_COMPLETE) // 4 Bit receive  {  ui8DummyRead = FIFO; // write the recieved bytes to the correct place of the buffer  Trf797xReadCont(&ui8DummyRead, 1);  Trf797xReset();  g\_sTrf797xStatus = PROTOCOL\_ERROR;  }  **else**  {  g\_sTrf797xStatus = PROTOCOL\_ERROR;  }  }  **else** **if**((\*pui8\_IrqStatus & BIT2) == TRF797x\_IRQ\_STATUS\_FRAMING\_ERROR) // byte framing error  {  **if**((\*pui8\_IrqStatus & BIT5) == TRF797x\_IRQ\_STATUS\_FIFO\_HIGH\_OR\_LOW)  {  g\_sTrf797xStatus = RX\_WAIT;  }  **else**  {  g\_sTrf797xStatus = PROTOCOL\_ERROR;  }  }  **else** **if**(\*pui8\_IrqStatus == TRF797x\_IRQ\_STATUS\_IDLE)  { // No response interrupt  //g\_sTrf797xStatus = NO\_RESPONSE\_RECEIVED;  }  **else** **if**(\*pui8\_IrqStatus == TRF797x\_IRQ\_STATUS\_NO\_RESPONSE)  { // No response interrupt  g\_sTrf797xStatus = NO\_RESPONSE\_RECEIVED\_15693;  g\_ui8FifoOffset = 0;  }  **else**  { // Interrupt register not properly set  g\_sTrf797xStatus = PROTOCOL\_ERROR;  Trf797xStopDecoders(); // reset the FIFO after TX  Trf797xReset();  Trf797xResetIrqStatus();  IRQ\_CLR;  }  } // Interrupt Service Routine  //===============================================================  //  // Trf797xIRQ - Interrupt handler for IRQ interrupts  //  // Handles receiving IRQ's, getting IRQ status, and maintaining  // timers/global variables  //  // \return None.  //  //===============================================================  **void**  Trf797xIRQ(**void**) // interrupt handler  {  u08\_t ui8IrqStatus, ui8IsoControl;  RESET\_COUNTER; // stop timer mode  STOP\_COUNTER;  g\_ui8IrqFlag = 0x01;  **do**  {  IRQ\_CLR; // PORT2 interrupt flag clear  // IRQ status register has to be read  Trf797xReadIrqStatus(&ui8IrqStatus);  //UartSendCString("IRQ Status: ");  //UartPutByteHex(ui8IrqStatus);  //UartPutCrlf();  **if**(ui8IrqStatus == 0xA0) // TX active and only 3 bytes left in FIFO  {  g\_sTrf797xStatus = TX\_WAIT;  //StatusPrint(g\_sTrf797xStatus);  **break**;  }  **else**  {  ui8IsoControl = TRF797x\_ISO\_CONTROL;  Trf797xReadSingle(&ui8IsoControl, 1);  **if**((ui8IsoControl & BIT5) != BIT5) // RFID mode  {  Trf797xISR(&ui8IrqStatus);  //StatusPrint(g\_sTrf797xStatus);  }  **else** // NFC mode  {  // Do Nothing  }  }  } **while**(GPIOIntStatus(GPIOA1\_BASE,1) & GPIO\_PIN\_4);  }  //===============================================================  //  // Trf797xRawWrite - Write data to TRF797x  //  // \param pui8Payload is the buffer with data packet contents  // \param ui8Length is the size of the data packet  //  // Function is used to write data directly to the TRF797x.  //  // \return None.  //  //===============================================================  **void**  Trf797xRawWrite(u08\_t \* pui8Payload, u08\_t ui8Length)  {  u08\_t ui8TxBytesRemaining;  u08\_t ui8TxIndex;  u08\_t ui8FifoTxLength;  u08\_t ui8TxBytesAvailable;  **if** (127 > ui8Length)  {  SpiRawWrite(pui8Payload, ui8Length);  }  **else**  {  ui8TxBytesRemaining = ui8Length;  ui8TxIndex = 0;  ui8TxBytesAvailable = 127;  **while**(ui8TxBytesRemaining > 0)  {  **if** (ui8TxBytesRemaining > 127)  {  SpiRawWrite(&pui8Payload[ui8TxIndex], ui8TxBytesAvailable);  ui8TxBytesRemaining = ui8TxBytesRemaining - ui8TxBytesAvailable;  ui8TxIndex = ui8TxIndex + ui8TxBytesAvailable;  }  **else**  {  SpiRawWrite(&pui8Payload[ui8TxIndex], ui8TxBytesRemaining);  }  Trf797xIrqWaitTimeoutTxOnly(35);  g\_sTrf797xStatus = Trf797xGetTrfStatus();  **if** ((g\_sTrf797xStatus == TX\_COMPLETE) || (g\_sTrf797xStatus == TX\_WAIT))  {  ui8FifoTxLength = TRF797x\_FIFO\_STATUS;  Trf797xReadSingle(&ui8FifoTxLength, 1); // determine the number of bytes left in FIFO  ui8TxBytesAvailable = 127-ui8FifoTxLength;  }  **else**  {  // Error occurred, break  g\_sTrf797xStatus = TX\_ERROR;  **break**;  }  }  }  }  //===============================================================  //  // Trf797xReadCont - Read out multiple TRF797x registers  //  // \param pui8Payload is the address of the first register as  // well as the pointer for buffer where the results will be  // \param ui8Length is the number of registers to read  //  // Function used to read a specified number of TRF797x registers  // from a specified address.  //  // \return None.  //  //===============================================================  **void**  Trf797xReadCont(u08\_t \* pui8Payload, u08\_t ui8Length)  {  SpiReadCont(pui8Payload, ui8Length);  }  //===============================================================  //  // Trf797xReadIsoControl - Read the ISO Control Register  //  // Function used to read the ISO Control Register of the TRF797x  //  // \return pui8Value returns the value of the ISO Control  // Register  //  //===============================================================  u08\_t  Trf797xReadIsoControl(**void**)  {  u08\_t pui8Value[1];  \*pui8Value = TRF797x\_ISO\_CONTROL;  SpiReadSingle(pui8Value, 1);  g\_ui8IsoControlValue = pui8Value[0]; // Update the ISO Control Register variable  **return** pui8Value[0];  }  //===============================================================  //  // Trf797xReadIrqStatus - Read out the IRQ Status Register  //  // \param pui8Value is the pointer to the buffer where the  // result will be  //  // Function used to read the IRQ Status register of the TRF797x  // and store the result into the location pointed to by the input  //  // \return None.  //  //===============================================================  **void**  Trf797xReadIrqStatus(u08\_t \* pui8Value)  {  \*pui8Value = TRF797x\_IRQ\_STATUS;  SpiReadSingle(pui8Value, 1);  /\*  \*pui8Value = TRF797x\_IRQ\_STATUS;  \*(pui8Value + 1) = TRF797x\_IRQ\_MASK;  Trf797xReadCont(pui8Value, 2); // read second reg. as dummy read  \*/  }  //===============================================================  //  // Trf797xReadSingle - Read out a single TRF797x registers  //  // \param pui8Value is the address of the register to read as  // well as pointer for the buffer where the result will be  //  // Function used to read a specific TRF797x register  //  // \return None.  //  //===============================================================  **void**  Trf797xReadSingle(u08\_t \*pbuf, u08\_t number)  {  SpiReadSingle(pbuf, number);  }  //===============================================================  //  // Trf797xReadStatusControl - Read the Chip Status Control  // Register  //  // Function used to read the Chip Status Control Register of the  // TRF797x  //  // \return pui8Value returns the value of the Chip Status Control  // Register  //  //===============================================================  u08\_t  Trf797xReadStatusControl(**void**)  {  u08\_t pui8Value[1];  \*pui8Value = TRF797x\_STATUS\_CONTROL;  SpiReadSingle(pui8Value, 1);  **if** ((pui8Value[0] & BIT5) == BIT5) // Check for RF field bit and update variable  {  g\_bRfFieldOn = true;  }  **else**  {  g\_bRfFieldOn = false;  }  **return** pui8Value[0];  }  //===============================================================  //  // Trf797xReset - Resets TRF797x  //  // Function used to reset the TRF797x  //  // \return None.  //  //===============================================================  **void**  Trf797xReset(**void**)  {  command[0] = RESET;  Trf797xDirectCommand(command);  }  //===============================================================  //  // Trf797xReset - Resets the IRQ Status Register of the TRF797x  //  // Function used to reset the TRF797x IRQ Status Register  //  // \return None.  //  //===============================================================  **void**  Trf797xResetIrqStatus(**void**)  {  u08\_t puiIrqStatus[2];  puiIrqStatus[0] = TRF797x\_IRQ\_STATUS;  puiIrqStatus[1] = TRF797x\_IRQ\_MASK;  Trf797xReadCont(puiIrqStatus, 2); // read second reg. as dummy read  }  //===============================================================  //  // Trf797xRunDecoders - Direct command to enable TRF797x receivers  //  // Issue direct command 0x17 - Enable Receivers to the TRF797x  //  // \return None.  //  //===============================================================  **void**  Trf797xRunDecoders(**void**)  {  command[0] = RUN\_DECODERS;  Trf797xDirectCommand(command);  }  //===============================================================  //  // Trf797xStopDecoders - Direct command to disable TRF797x  // receivers  //  // Issue direct command 0x16 - Block Receivers to the TRF797x  //  // \return None.  //  //===============================================================  **void**  Trf797xStopDecoders(**void**)  {  command[0] = STOP\_DECODERS;  Trf797xDirectCommand(command);  }  //===============================================================  //  // Trf797xTransmitNextSlot - Direct command to transmit next slot  // for ISO15693 End of Frame  //  // Issue direct command 0x14 - End of Frame/Transmit Next Time Slot  // (ISO15693) to the TRF797x  //  // \return None.  //  //===============================================================  **void**  Trf797xTransmitNextSlot(**void**)  {  command[0] = TRANSMIT\_NEXT\_SLOT;  Trf797xDirectCommand(command);  }  //===============================================================  //  // Trf797xTurnRfOff - Turn off the transmission of the TRF797x  // RF Field  //  // Function used stop the TRF797x transmitting an RF field  //  // \return None.  //  //===============================================================  **void**  Trf797xTurnRfOff(**void**)  {  u08\_t pui8Command[2];  pui8Command[0] = TRF797x\_STATUS\_CONTROL;  pui8Command[1] = 0x00; // 3.3VDC, full power out  // pui8Command[1] = 0x10; // 3.3VDC, half power out  Trf797xWriteSingle(pui8Command, 2);  g\_bRfFieldOn = false; // Update RF Field variable  }  //===============================================================  //  // Trf797xTurnRfOn - Turns on the transmission of the TRF797x  // RF Field  //  // Function used make the TRF797x transmit an RF field  //  // \return None.  //  //===============================================================  **void**  Trf797xTurnRfOn(**void**)  {  u08\_t pui8Command[2];  pui8Command[0] = TRF797x\_STATUS\_CONTROL;  pui8Command[1] = 0x20; // 3.3VDC, full power out  // pui8Command[1] = 0x30; // 3.3VDC, half power out  Trf797xWriteSingle(pui8Command, 2);  g\_bRfFieldOn = true; // Update RF Field variable  }  //===============================================================  //  // Trf797xWriteCont - Write to consecutive TRF797x registers  //  // \param pui8Payload is the address of the first register  // followed by the contents to write for each register  // \param ui8Length is the number of registers to write + 1  // Minimum value of ui8Length allowed = 2 (a write to 1 register)  //  // Function used to write to a specific number of TRF797x  // registers from a specific address.  //  // \return None.  //  //===============================================================  **void**  Trf797xWriteCont(u08\_t \* pui8Payload, u08\_t ui8Length)  {  **if** (ui8Length > 1) // Cannot specify a length of 1  {  **if** (\*pui8Payload == 0x00) // If the write starts at the Chip Status Control Register  {  **if** (((\*pui8Payload+1) & BIT5) == BIT5) // Check for RF field bit and update variable  {  g\_bRfFieldOn = true;  }  **else**  {  g\_bRfFieldOn = false;  }  **if** (ui8Length > 2) // Check if the write length includes the ISO Control Register being written to (0x01)  {  g\_ui8IsoControlValue = (\*pui8Payload+2); // If so, update the Iso Control Register variable  }  }  **else** **if** (\*pui8Payload == 0x01) // If the write starts at the ISO Control Register  {  g\_ui8IsoControlValue = \*pui8Payload+1; // Update the ISO Control Register variable  }  // Call continuous write function  SpiWriteCont(pui8Payload, ui8Length);  }  **else**  {  // Error, cannot specify a length of 1  **return**;  }  }  //===============================================================  //  // Trf797xWriteIsoControl - Write to TRF797x ISO Control Register  //  // \param ui8IsoControl is the value to write to the ISO control  // register of the TRF797x  //  // Function used to write a new value into the ISO Control  // register of the TRF797x.  //  // \return None.  //  //===============================================================  **void**  Trf797xWriteIsoControl(u08\_t ui8IsoControl)  {  u08\_t pui8Write[2];  **if**((ui8IsoControl & BIT5) == BIT5)  {  // Attempt to enable Card Emulation/Peer to Peer which is not supported by firmware  // Exit function to avoid issues with that  **return**;  }  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  pui8Write[1] &= ~BIT5;  Trf797xWriteSingle(pui8Write, 2);  g\_ui8IsoControlValue = ui8IsoControl; // Update the ISO Control Register variable  }  //===============================================================  //  // Trf797xWriteRegister - Write single to a TRF797x Register  //  // \param ui8TrfRegister is the register address for the write  // \param ui8Value is the value to write to the specified  // register  //  // Function used to write a new value into a single TRF797x  // register.  //  // \return None.  //  //===============================================================  **void**  Trf797xWriteRegister(u08\_t ui8TrfRegister, u08\_t ui8Value)  {  u08\_t pui8Write[2];  **if** (ui8TrfRegister == TRF797x\_ISO\_CONTROL)  {  // Attempt to enable Card Emulation/Peer to Peer which is not supported by firmware  // Exit function to avoid issues with that  **if** ((ui8Value & BIT5) == BIT5)  {  **return**;  }  g\_ui8IsoControlValue = ui8Value; // Update the ISO Control Register variable  }  **if** (ui8TrfRegister == TRF797x\_STATUS\_CONTROL)  {  **if** ((ui8Value & BIT5) == BIT5) // Check for RF field bit and update variable  {  g\_bRfFieldOn = true;  }  **else**  {  g\_bRfFieldOn = false;  }  }  pui8Write[0] = ui8TrfRegister;  pui8Write[1] = ui8Value;  Trf797xWriteSingle(pui8Write, 2);  }  //===============================================================  //  // Trf797xWriteInitiatorSetup - Write the initial settings for  // a set of TRF797x registers based on which protocol is to be  // enabled.  //  // \param ui8IsoControl is the value to write to the ISO Control  // register of the TRF797x  //  // Function used to write to a series of TRF797x registers based  // on which technology will be enabled in the ISO control register  // This function currently only enables 1 technology at a time  //  // \return None.  //  //===============================================================  **void**  Trf797xWriteInitiatorSetup(u08\_t ui8IsoControl)  {  u08\_t pui8Write[2];  g\_ui8IsoControlValue = ui8IsoControl; // Update the ISO Control Register variable  **if** (ui8IsoControl == 0x88) // ISO14443A  {  // Register 0x00  pui8Write[0] = TRF797x\_STATUS\_CONTROL;  pui8Write[1] = 0x20;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x01  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x09 - System Clock Output, Modulation Scheme  pui8Write[0] = TRF797x\_MODULATOR\_CONTROL;  pui8Write[1] = 0x01; // Sys Clock Output = 13.56MHz, OOK = 100%  Trf797xWriteSingle(pui8Write, 2);  }  **if** (ui8IsoControl == 0x0C) // ISO14443B  {  // Register 0x01  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x09 - System Clock Output, Modulation Scheme  pui8Write[0] = TRF797x\_MODULATOR\_CONTROL;  pui8Write[1] = 0x00; // Sys Clock Output = 13.56MHz, ASK 10%  Trf797xWriteSingle(pui8Write, 2);  }  **if** (ui8IsoControl == 0x02) // ISO15693  {  pui8Write[0] = TRF797x\_STATUS\_CONTROL;  pui8Write[1] = 0x20;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x01  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  Trf797xWriteSingle(pui8Write, 2);  // Resgister 0x07 - No Response Wait Time  pui8Write[0] = TRF797x\_RX\_NO\_RESPONSE\_WAIT\_TIME;  pui8Write[1] = 0x15;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x09 - System Clock Output, Modulation Scheme  pui8Write[0] = TRF797x\_MODULATOR\_CONTROL;  pui8Write[1] = 0x01; // Sys Clock Output = 13.56MHz, OOK = 100%  Trf797xWriteSingle(pui8Write, 2);  /\* write[0] = 0x20; //Continuous Write, starting with register 0x00  write[1] = 0x20; //Value for Chip Status Control Register 0x00, 0x20 = +3.3VDC, full power, etc.  write[2] = 0x02; //Value for ISO Control Register 0x01, 0x02 = high tag data rate, etc.  write[3] = 0x00; //0x02  write[4] = 0x00;  write[5] = 0xC1;  write[6] = 0xBB;  write[7] = 0x00;  write[8] = 0x30;  write[9] = 0x1F;  write[10] = 0x01;  write[11] = 0x40;  write[12] = 0x03;\*/  /\* // Register 0x00  pui8Write[0] = TRF797x\_STATUS\_CONTROL;  pui8Write[1] = 0x20;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x01  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x02  pui8Write[0] = TRF797x\_ISO\_14443\_TX\_OPTIONS;  pui8Write[1] = 0x00;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x03  pui8Write[0] = TRF797x\_ISO\_14443\_BITRATE\_OPTIONS;  pui8Write[1] = 0x00;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x04  pui8Write[0] = TRF797x\_TX\_TIMER\_EPC\_HIGH;  pui8Write[1] = 0xC1;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x05  pui8Write[0] = TRF797x\_TX\_TIMER\_EPC\_LOW;  pui8Write[1] = 0xBB;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x06  pui8Write[0] = TRF797x\_TX\_PULSE\_LENGTH\_CONTROL;  pui8Write[1] = 0x00;  Trf797xWriteSingle(pui8Write, 2);  // Resgister 0x07 - No Response Wait Time  pui8Write[0] = TRF797x\_RX\_NO\_RESPONSE\_WAIT\_TIME;  pui8Write[1] = 0x30;  Trf797xWriteSingle(pui8Write, 2);  // Resgister 0x08 -  pui8Write[0] = TRF797x\_RX\_WAIT\_TIME;  pui8Write[1] = 0x1F;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x09 - System Clock Output, Modulation Scheme  pui8Write[0] = TRF797x\_MODULATOR\_CONTROL;  pui8Write[1] = 0x01; // Sys Clock Output = 13.56MHz, OOK = 100%  Trf797xWriteSingle(pui8Write, 2);  // Register 0x0A  pui8Write[0] = TRF797x\_RX\_SPECIAL\_SETTINGS;  pui8Write[1] = 0x40;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x0B  pui8Write[0] = TRF797x\_REGULATOR\_CONTROL;  pui8Write[1] = 0x03;  Trf797xWriteSingle(pui8Write, 2);  \*/  /\* write[0] = 0x20; //Continuous Write, starting with register 0x00  write[1] = 0x20; //Value for Chip Status Control Register 0x00, 0x20 = +3.3VDC, full power, etc.  write[2] = 0x02; //Value for ISO Control Register 0x01, 0x02 = high tag data rate, etc.  write[3] = 0x00; //0x02  write[4] = 0x00;  write[5] = 0xC1;  write[6] = 0xBB;  write[7] = 0x00;  write[8] = 0x30;  write[9] = 0x1F;  write[10] = 0x01;  write[11] = 0x40;  write[12] = 0x03;\*/  //Trf797xWriteCont(write, 13); //writes registers 0x00:0x0B  }  **if** (ui8IsoControl == 0x1A) // FeliCa  {  // Register 0x01  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x09 - System Clock Output, Modulation Scheme  pui8Write[0] = TRF797x\_MODULATOR\_CONTROL;  pui8Write[1] = 0x00; // Sys Clock Output = 13.56MHz, ASK 10%  Trf797xWriteSingle(pui8Write, 2);  }  **if** (ui8IsoControl == 0x86) // PicoPass  {  // Register 0x01  pui8Write[0] = TRF797x\_ISO\_CONTROL;  pui8Write[1] = ui8IsoControl;  Trf797xWriteSingle(pui8Write, 2);  // Register 0x09 - System Clock Output, Modulation Scheme  pui8Write[0] = TRF797x\_MODULATOR\_CONTROL;  pui8Write[1] = 0x00; // Sys Clock Output = 13.56MHz, ASK 10%  Trf797xWriteSingle(pui8Write, 2);  }  // Register 0x14 - Adjustable FIFO Level  pui8Write[0] = TRF797x\_FIFO\_IRQ\_LEVELS;  **if** (ui8IsoControl == 0x86)  {  // TX water level set to 32 bytes for PicoPass  pui8Write[1] = 0x0F;  }  **else**  {  pui8Write[1] = 0x0C;  }  Trf797xWriteSingle(pui8Write, 2);  }  //===============================================================  //  // Trf797xWriteSingle - Write single to a TRF797x Register  //  // \param pui8Value is a pointer to a buffer which has the  // Register address for the write followed by the data to be  // written to that register  //  // Function used to write a new value into a single TRF797x  // register.  //  // \return None.  //  //===============================================================  **void**  Trf797xWriteSingle(u08\_t \*pbuf, u08\_t length)  {  SpiWriteSingle(pbuf, length);  }  //===============================================================  //  // Trf797xIrqWaitTimeout - Timeout sequence for both TX and RX  //  // \param ui8TxTimeout is the TX timeout in milliseconds  // \param ui8RxTimeout is the RX timeout in milliseconds  //  // Function is used to ensure data is transmitted correct as well  // as determine if data has been received prior to the RX timeout  // When the RX timeout occurs before data is received, then mark  // the TRF797x status as a No Response Received status.  //  // \return None.  //  //===============================================================  **void** Trf797xIrqWaitTimeout(**long** ui8TxTimeout, **long** ui8RxTimeout)  {  g\_ui8FifoOffset = 0; // Reset the FIFO Offset prior to receiving data  g\_sTrf797xStatus = RX\_WAIT;  g\_ui8TimeoutFlag = 0x00;  **while**((g\_sTrf797xStatus != TX\_COMPLETE) && (g\_sTrf797xStatus != TX\_ERROR))  { // Wait for end of TX  // Clear the IRQ Flag  g\_ui8IrqFlag = 0x00;  // Setup for the Timer  // Calculate the timeout value for the timer  A2CounterLoad(COUNT\_1ms \* ui8TxTimeout);  // Start the Timeout  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  RESET\_COUNTER;  STOP\_COUNTER;  **if** (g\_sTrf797xStatus != TX\_COMPLETE)  {  **if** (g\_sTrf797xStatus == TX\_WAIT) // Wait longer since we received an 0xA0  {  UartSendCString("Received TX\_WAIT\n\r");  Trf797xIrqWaitTimeoutTxOnly(ui8TxTimeout); // Wait longer for transmission to complete  }  **else** // Failed to send packet properly - Exit TX Timeout  {  g\_sTrf797xStatus = TX\_ERROR; // Set status to error  }  }  }  //StatusPrint(g\_sTrf797xStatus);  **if** (g\_sTrf797xStatus != TX\_ERROR)  {  g\_sTrf797xStatus = RX\_WAIT;  g\_ui8TimeoutFlag = 0x00;  **while**(g\_sTrf797xStatus == RX\_WAIT || g\_sTrf797xStatus == TX\_COMPLETE) // Wait for end of RX or timeout  {  // Clear the IRQ Flag  g\_ui8IrqFlag = 0x00;  // Setup for the Timer  // Calculate the timeout value for the timer  A2CounterLoad(COUNT\_1ms \* ui8RxTimeout);  //A2CounterLoad(COUNT\_1ms \* 10000);  // Start the Timeout  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  //StatusPrint(g\_sTrf797xStatus);  **while** (g\_sTrf797xStatus == RX\_WAIT\_EXTENSION || g\_sTrf797xStatus == TX\_WAIT)  {  RESET\_COUNTER;  STOP\_COUNTER;  g\_ui8IrqFlag = 0x00;  **if** ((g\_ui8IsoControlValue & 0x1F) > 0x07)  {  A2CounterLoad(COUNT\_1ms \* 7);  }  **else**  {  A2CounterLoad(COUNT\_1ms \* 50);  }  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  }  RESET\_COUNTER;  STOP\_COUNTER;  **if** (g\_sTrf797xStatus == RX\_WAIT)  {  // Exit the while loop  g\_sTrf797xStatus = NO\_RESPONSE\_RECEIVED;  }  }  }  }  //===============================================================  //  // Trf797xIrqWaitTimeoutTxOnly - Timeout sequence for just TX  //  // \param ui8TxTimeout is the TX timeout in milliseconds  //  // Function is used to handle delays for transmit only, which  // is helpful when a response is not expected (such as with sleep  // or halt commands)  // This function will not delay to receive any responses  //  // \return None.  //  //===============================================================  **void** Trf797xIrqWaitTimeoutTxOnly(u08\_t ui8TxTimeout)  {  g\_sTrf797xStatus = RX\_WAIT;  g\_ui8TimeoutFlag = 0x00;  **while**((g\_sTrf797xStatus != TX\_COMPLETE) && (g\_sTrf797xStatus != NO\_RESPONSE\_RECEIVED) && (g\_sTrf797xStatus != TX\_ERROR))  { // Wait for end of TX  // Clear the IRQ Flag  g\_ui8IrqFlag = 0x00;  // Setup for the Timer  // Calculate the timeout value for the timer  A2CounterLoad(COUNT\_1ms \* ui8TxTimeout);  // Start the Timeout  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  RESET\_COUNTER;  STOP\_COUNTER;  **if** (g\_sTrf797xStatus != TX\_COMPLETE)  {  **if** (g\_sTrf797xStatus == TX\_WAIT) // Wait longer since we received an 0xA0  {  Trf797xIrqWaitTimeoutTxOnly(ui8TxTimeout); // Wait longer for transmission to complete  }  **else** // Failed to send packet properly - Exit TX Timeout  {  g\_sTrf797xStatus = TX\_ERROR; // Set status to error  }  }  }  }  //===============================================================  //  // Trf797xIrqWaitTimeoutRxOnly - Timeout sequence for just RX  //  // \param ui8RxTimeout is the RX timeout in milliseconds  //  // Function is used to handle delays for receive only, such as  // when waiting for a response without sending a data package  // during anticollision slot procedures.  //  // \return None.  //  //===============================================================  **void** Trf797xIrqWaitTimeoutRxOnly(u08\_t ui8RxTimeout)  {  g\_ui8FifoOffset = 0; // Reset the FIFO Offset prior to receiving data  g\_sTrf797xStatus = RX\_WAIT;  g\_ui8TimeoutFlag = 0x00;  **while**(g\_sTrf797xStatus == RX\_WAIT) // Wait for end of RX or timeout  {  // Clear the IRQ Flag  g\_ui8IrqFlag = 0x00;  // Setup for the Timer  // Calculate the timeout value for the timer  A2CounterLoad(COUNT\_1ms \* ui8RxTimeout);  // Start the Timeout  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  **while** (g\_sTrf797xStatus == RX\_WAIT\_EXTENSION)  {  RESET\_COUNTER;  STOP\_COUNTER;  g\_ui8IrqFlag = 0x00;  **if** ((g\_ui8IsoControlValue & 0x1F) > 0x07)  {  A2CounterLoad(COUNT\_1ms \* 7);  }  **else**  {  A2CounterLoad(COUNT\_1ms \* 50);  }  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  }  RESET\_COUNTER;  STOP\_COUNTER;  **if** (g\_sTrf797xStatus == RX\_WAIT)  {  // Exit the while loop  g\_sTrf797xStatus = NO\_RESPONSE\_RECEIVED;  }  }  }  //===============================================================  //  // Trf797xIrqWaitTimeoutFeliCa - Timeout sequence for just RX  // specifically for FeliCa slot timings  //  // Function is used to handle receive delays for FeliCa  // anticollision only.  //  // \return None.  //  //===============================================================  **void** Trf797xIrqWaitTimeoutFeliCa(**void**)  {  g\_ui8FifoOffset = 0; // Reset the FIFO Offset prior to receiving data  g\_sTrf797xStatus = RX\_WAIT;  g\_ui8TimeoutFlag = 0x00;  **while**(g\_sTrf797xStatus == RX\_WAIT) // Wait for end of RX or timeout  {  // Clear the IRQ Flag  g\_ui8IrqFlag = 0x00;  // Setup for the Timer  // Calculate the timeout value for the timer  A2CounterLoad(2);  // Start the Timeout  START\_COUNTER;  **while**((g\_ui8IrqFlag == 0x00) && (g\_ui8TimeoutFlag == 0x00)) // Wait for an interrupt  {  // Do Nothing  }  RESET\_COUNTER;  STOP\_COUNTER;  **if** (g\_sTrf797xStatus == RX\_WAIT)  {  // Exit the while loop  g\_sTrf797xStatus = NO\_RESPONSE\_RECEIVED;  }  }  }  //===============================================================  //  // Trf797xGetTrfStatus - Returns current TRF797x driver status  //  // Function is used to pass the current TRF797x driver status  // up to higher layers of the firmware  //  // \return g\_sTrf797xStatus returns the current TRF797x drive  // status  //  //===============================================================  tTRF797x\_Status Trf797xGetTrfStatus(**void**)  {  **return** g\_sTrf797xStatus;  }  //===============================================================  //  // Trf797xSetTrfStatus - Set the TRF797x driver status  //  // \param sTrfStatus is the new TRF797x driver status  //  // Function is used to allow higher layers of the firmware to  // set the TRF797x driver status without an IRQ event. Use with  // caution.  //  // \return None.  //  //===============================================================  **void** Trf797xSetTrfStatus(tTRF797x\_Status sTrfStatus)  {  g\_sTrf797xStatus = sTrfStatus;  }  //===============================================================  //  // Trf797xGetCollisionPosition - Return the current Collision  // Position value  //  // Function is used to pass the current Collision Position value  // up to higher layers of the firmware. Used for ISO14443 Type A  // anti-collision process.  //  // \return g\_ui8CollisionPosition returns the current Collision  // Position value  //  //===============================================================  u08\_t Trf797xGetCollisionPosition(**void**)  {  **return** g\_ui8CollisionPosition;  }  //===============================================================  //  // Trf797xSetCollisionPosition - Set the Collision Position  // variable  //  // \param ui8ColPos is the new Collision Position value  //  // Function is used to allow higher layers of the firmware to  // set the Collision Position variable. Used for ISO14443 Type A  // anti-collision process  //  // \return None.  //  //===============================================================  **void** Trf797xSetCollisionPosition(u08\_t ui8ColPos)  {  g\_ui8CollisionPosition = ui8ColPos;  }  //===============================================================  //  // Trf797xGetRxBytesReceived - Returns the Number of RX Bytes  // received by the TRF797x FIFO  //  // Function is used to pass the number of RX bytes received  // during the last packet reception up to higher layers of the  // firmware.  // This can be used by the application layer to check for the  // length of bytes received and ensure packets were correctly  // received.  //  // \return g\_ui8FifoRxLength returns the current FIFO RX Length  //  //===============================================================  u08\_t Trf797xGetRxBytesReceived(**void**)  {  **return** g\_ui8FifoRxLength;  }  //===============================================================  //  // Trf797xGetIsoControlValue - Fetch the latest Iso Control  // Register value  //  // Function is used to pass the current Iso Control Register  // value up to higher layers of the firmware.  //  // The Iso Control Register value is updated whenever a read or  // write to the Iso Control Register occurs.  //  // \return g\_ui8IsoControlValue returns the current ISO Control  // Register value  //  //===============================================================  u08\_t Trf797xGetIsoControlValue(**void**)  {  **return** g\_ui8IsoControlValue;  }  //===============================================================  //  // Trf797xReadRssiLevels - Reads the RSSI Levels register and then  // returns the value of the latest RSSI measurement  //  // Function is used to determine the latest RSSI Level reading.  // The RSSI Level register is updated when RF data is received  // by the TRF797x. For most accurate reading, this function should  // be called directly following a reception of RF data.  //  // \return pui8Read returns the latest RSSI Register value  //  //===============================================================  u08\_t Trf797xReadRssiLevels(**void**)  {  u08\_t pui8Read[1];  pui8Read[0] = TRF797x\_RSSI\_LEVELS; // read RSSI levels  Trf797xReadSingle(pui8Read, 1);  **return** pui8Read[0];  }  //===============================================================  //  // Trf797xCheckRfField - Fetch the RfFieldOn variable  //  // Function is used to pass the current RfFieldOn value up to  // higher layers of the firmware  //  // \return g\_bRfFieldOn retuns the current RF field status  //  //===============================================================  bool Trf797xCheckRfField(**void**)  {  **return** g\_bRfFieldOn;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //  //! This function gets triggered when A2CNT interrupt occures  //!  //! \param none  //!  //! \return None  //!  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **void** A2CNTIntHandler (**void**){  u08\_t ui8IrqStatus;  ui8IrqStatus = 0x00;  STOP\_COUNTER;  RESET\_COUNTER;  g\_ui8TimeoutFlag = 0x01;  Trf797xReadIrqStatus(&ui8IrqStatus);  ui8IrqStatus = ui8IrqStatus & 0xF7; // Set the parity flag to 0  **if**(ui8IrqStatus == TRF797x\_IRQ\_STATUS\_TX\_COMPLETE)  {  g\_sTrf797xStatus = TX\_COMPLETE;  }  **else** **if**(ui8IrqStatus == TRF797x\_IRQ\_STATUS\_IDLE)  {  g\_sTrf797xStatus = NO\_RESPONSE\_RECEIVED;  }  **else**  {  g\_sTrf797xStatus = RX\_WAIT;  }  }  **void** StatusPrint(tTRF797x\_Status sTrfStatus) {  **switch** (sTrfStatus) {  **case** TRF\_IDLE:  UartSendCString("Status: TRF\_IDLE\n\r");  **break**;  **case** TX\_COMPLETE:  UartSendCString("Status: TX\_COMPLETE\n\r");  **break**;  **case** RX\_COMPLETE:  UartSendCString("Status: RX\_COMPLETE\n\r");  **break**;  **case** TX\_ERROR:  UartSendCString("Status: TX\_ERROR\n\r");  **break**;  **case** RX\_WAIT:  UartSendCString("Status: RX\_WAIT\n\r");  **break**;  **case** RX\_WAIT\_EXTENSION:  UartSendCString("Status: RX\_WAIT\_EXTENSION\n\r");  **break**;  **case** TX\_WAIT:  UartSendCString("Status: TX\_WAIT\n\r");  **break**;  **case** PROTOCOL\_ERROR:  UartSendCString("Status: PROTOCOL\_ERROR\n\r");  **break**;  **case** NO\_RESPONSE\_RECEIVED:  UartSendCString("Status: NO\_RESPONSE\_RECEIVED\n\r");  **break**;  **case** NO\_RESPONSE\_RECEIVED\_15693:  UartSendCString("Status: NO\_RESPONSE\_RECEIVED\_15693\n\r");  **break**;  **default**:  UartSendCString("Status: Unknown\n\r");  **break**;  }  } |

|  |
| --- |
| **type\_4\_ndef.h** |
| /\*  \* File Name: type\_4\_ndef.h  \*  \* Description: Headers for Type 4 NDEF Functions  \*  \* Copyright (C) 2016 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_NDEF\_H\_  **#define** \_NDEF\_H\_  //===============================================================  **#include** "trf797x.h"  //===============================================================  u08\_t NDEF\_ApplicationSelect(**void**);  u08\_t NDEF\_CapabilityContainerSelect(**void**);  u16\_t NDEF\_ReadBinary(u16\_t ui16FileOffset, u08\_t ui8ReadLength);  u08\_t NDEF\_FileSelect(u16\_t ui16FileID);  **void** NDEF\_UpdateBinaryLength(u16\_t ui16Nlen);  **void** Ndef\_ReadCC(**void**);  **void** Ndef\_ReadApplication(**void**);  **void** Ndef\_WriteText(**void**);  **void** Ndef\_WriteUri(**void**);  **void** Ndef\_SetBlockNumberBit(**bool** bValue);  **void** NDEF\_UpdateBinaryText(**void**);  **void** NDEF\_UpdateBinaryUri(**void**);  **extern** **char** g\_ndef\_content[600];  //===============================================================  **#endif** |

|  |
| --- |
| **type\_4\_ndef.c** |
| /\*  \* File Name: type\_4\_ndef.c  \*  \* Description: Type 4 NDEF Functions  \*  \* Copyright (C) 2016 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#include** <string.h>  **#include** "type\_4\_ndef.h"  **#include** "trf7970BoosterPack.h"  **#include** "trf797x.h"  //===============================================================  **extern** u08\_t g\_ui8TrfBuffer[NFC\_FIFO\_SIZE];  **static** **volatile** tTRF797x\_Status g\_sTrfStatus;  **static** **volatile** bool g\_bBlockNumberBit = 0;  **char** g\_ndef\_content[600]; // used for saving a content of TAG buffer  //===============================================================  //Echoes back WTX request from Android HCE  //Overwrites g\_sTrfStatus  **static** **void** SBlock\_WTXEcho(u08\_t wtxBuf, u08\_t didBuf)  {  u08\_t ui8Offset = 0;  //S-Block WTX request. Echo back  // Buffer setup for FIFO writing  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x20; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = wtxBuf; //RATS Command  g\_ui8TrfBuffer[ui8Offset++] = didBuf; //RATS Parameters: 128 byte max receive and CID = 0  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(20,50);  g\_sTrfStatus = Trf797xGetTrfStatus();  }  u08\_t NDEF\_ApplicationSelect(**void**)  {  u08\_t ui8SelectSuccess = STATUS\_FAIL;  u08\_t ui8Offset = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0xE0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xA4; // INS = Select (Application in this case)  g\_ui8TrfBuffer[ui8Offset++] = 0x04; // P1  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // P2  g\_ui8TrfBuffer[ui8Offset++] = 0x07; // Lc  g\_ui8TrfBuffer[ui8Offset++] = 0xD2; // Data = 0xD2760000850101 - per NFC Forum Type 4 Tag Operation  g\_ui8TrfBuffer[ui8Offset++] = 0x76;  g\_ui8TrfBuffer[ui8Offset++] = 0x00;  g\_ui8TrfBuffer[ui8Offset++] = 0x00;  g\_ui8TrfBuffer[ui8Offset++] = 0x85;  g\_ui8TrfBuffer[ui8Offset++] = 0x01;  g\_ui8TrfBuffer[ui8Offset++] = 0x01;  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Le  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(20,100); // 10 millisecond TX timeout, 20 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  // If data received, should return same PCB, SW1 = 0x90, SW2 = 0x00  **if** (g\_sTrfStatus == RX\_COMPLETE)  {  **if** (g\_ui8TrfBuffer[0] == 0xF2)  {  //Delay a bit before echoing  McuDelayMillisecond(50);  SBlock\_WTXEcho(g\_ui8TrfBuffer[0], g\_ui8TrfBuffer[1]);  }  **if**(g\_sTrfStatus == RX\_COMPLETE && (g\_ui8TrfBuffer[0] == 0x02 | g\_bBlockNumberBit) && (g\_ui8TrfBuffer[1] == 0x90) && (g\_ui8TrfBuffer[2] == 0x00))  {  //Copy received payload except the first two  **unsigned** **int** copyIdx;  //Clear buffer first  memset(g\_ndef\_content, 0, **sizeof** g\_ndef\_content);  **for** (copyIdx = 3; copyIdx < g\_ui8FifoRxLength; copyIdx++) {  g\_ndef\_content[copyIdx - 3] = g\_ui8TrfBuffer[copyIdx];  }  ui8SelectSuccess = STATUS\_SUCCESS;  }  }  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  **return** ui8SelectSuccess;  }  u08\_t NDEF\_CapabilityContainerSelect(**void**)  {  u08\_t ui8SelectSuccess = STATUS\_FAIL;  u08\_t ui8Offset = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x80; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xA4; // INS = Read CC  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // P1  g\_ui8TrfBuffer[ui8Offset++] = 0x0C; // P2  g\_ui8TrfBuffer[ui8Offset++] = 0x02; // Lc  g\_ui8TrfBuffer[ui8Offset++] = 0xE1; // Data = 0xE103 - per NFC Forum Type 4 Tag Operation  g\_ui8TrfBuffer[ui8Offset++] = 0x03;  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(200,200); // 10 millisecond TX timeout, 20 millisecond RX timeout  g\_sTrfStatus = Trf797xGetTrfStatus();  // If data received, should return same PCB, SW1 = 0x90, SW2 = 0x00  **if** (g\_sTrfStatus == RX\_COMPLETE)  {  **if** (g\_ui8TrfBuffer[0] == 0xF2)  {  SBlock\_WTXEcho(g\_ui8TrfBuffer[0], g\_ui8TrfBuffer[1]);  }  **if**((g\_ui8TrfBuffer[0] == 0x02 | g\_bBlockNumberBit) && (g\_ui8TrfBuffer[1] == 0x90) && (g\_ui8TrfBuffer[2] == 0x00))  {  ui8SelectSuccess = STATUS\_SUCCESS;  }  }  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  McuDelayMillisecond(1); // Short delay before sending next command  **return** ui8SelectSuccess;  }  u16\_t NDEF\_ReadBinary(u16\_t ui16FileOffset, u08\_t ui8ReadLength)  {  u08\_t ui8Offset = 0;  u16\_t ui16Nlen = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x60; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xB0; // INS = Read Binary  g\_ui8TrfBuffer[ui8Offset++] = ((ui16FileOffset >> 8) & 0xFF); // File Offset where to start reading data  g\_ui8TrfBuffer[ui8Offset++] = (ui16FileOffset & 0x00FF); // File Offset where to start reading data  g\_ui8TrfBuffer[ui8Offset++] = ui8ReadLength; // Read Length  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(20,100); // 20 millisecond TX timeout, 100 millisecond RX timeout  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  g\_sTrfStatus = Trf797xGetTrfStatus();  **if** (g\_ui8TrfBuffer[0] == 0xF2)  {  SBlock\_WTXEcho(g\_ui8TrfBuffer[0], g\_ui8TrfBuffer[1]);  }  **if**(g\_sTrfStatus == RX\_COMPLETE)  {  ui16Nlen = (g\_ui8TrfBuffer[1] << 8) + g\_ui8TrfBuffer[2];  }  McuDelayMillisecond(1); // Short delay before sending next command  **return** ui16Nlen;  }  u08\_t NDEF\_FileSelect(u16\_t ui16FileID)  {  u08\_t ui8SelectSuccess = STATUS\_FAIL;  u08\_t ui8Offset = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x80; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xA4; // INS = Select (File in this case)  g\_ui8TrfBuffer[ui8Offset++] = 0x00;  g\_ui8TrfBuffer[ui8Offset++] = 0x0C;  g\_ui8TrfBuffer[ui8Offset++] = 0x02;  g\_ui8TrfBuffer[ui8Offset++] = ((ui16FileID >> 8) & 0xFF);  g\_ui8TrfBuffer[ui8Offset++] = (ui16FileID & 0x00FF);  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(15,50); // 15 millisecond TX timeout, 50 millisecond RX timeout  // If data received, should return same PCB, SW1 = 0x90, SW2 = 0x00  **if** (g\_sTrfStatus == RX\_COMPLETE)  {  **if**((g\_ui8TrfBuffer[0] == 0x02 | g\_bBlockNumberBit) && (g\_ui8TrfBuffer[1] == 0x90) && (g\_ui8TrfBuffer[2] == 0x00))  {  ui8SelectSuccess = STATUS\_SUCCESS;  }  }  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  McuDelayMillisecond(1); // Short delay before sending next command  **return** ui8SelectSuccess;  }  **void** NDEF\_UpdateBinaryLength(u16\_t ui16Nlen)  {  u08\_t ui8Offset = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x80; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xD6; // INS = Update Binary  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Offset, P1  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Offset, P2  g\_ui8TrfBuffer[ui8Offset++] = 0x02; // Length, Lc  g\_ui8TrfBuffer[ui8Offset++] = ((ui16Nlen >> 8) & 0xFF); // MSByte NLEN being set to 0  g\_ui8TrfBuffer[ui8Offset++] = (ui16Nlen & 0xFF); // LSByte NLEN being set to 0  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(10,20); // 10 millisecond TX timeout, 20 millisecond RX timeout  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  }  **void** Ndef\_ReadCC(**void**)  {  NDEF\_CapabilityContainerSelect(); // Selects the Capability Container  NDEF\_ReadBinary(0, 15); // Read the contents of the capability container  }  **void** Ndef\_ReadApplication(**void**)  {  u16\_t ui16NdefLength = 0;  u08\_t ui8NdefReadLength = 0;  **#ifdef** ENABLE\_HOST  u08\_t ui8NdefMessageOffset = 0;  u08\_t ui8LoopCount = 0;  **#endif**  NDEF\_FileSelect(0xE104); // Selects NDEF Application  ui16NdefLength = NDEF\_ReadBinary(0, 2); // Reads NDEF Application for length of message  **if** (ui16NdefLength > NFC\_FIFO\_SIZE)  {  ui8NdefReadLength = NFC\_FIFO\_SIZE;  **#ifdef** ENABLE\_HOST  UartSendCString("NDEF Message Size Exceeds Internal Buffer, Displaying Partial Message Contents Only.");  UartPutCrlf();  **#endif**  }  **else**  {  ui8NdefReadLength = ui16NdefLength;  }  NDEF\_ReadBinary(2, ui8NdefReadLength); // Reads NDEF Application for the NDEF content  **#ifdef** ENABLE\_HOST  UartSendCString("NDEF Message: ");  UartPutChar('[');  **if** (g\_ui8TrfBuffer[4] == 0x54)  {  ui8NdefMessageOffset = 8;  }  **else** **if** (g\_ui8TrfBuffer[4] == 0x55)  {  ui8NdefMessageOffset = 6;  **if** (g\_ui8TrfBuffer[5] == 0x01)  {  UartSendCString("http://wwww.");  }  **else** **if** (g\_ui8TrfBuffer[5] == 0x02)  {  UartSendCString("https://wwww.");  }  **else** **if** (g\_ui8TrfBuffer[5] == 0x03)  {  UartSendCString("http://");  }  **else** **if** (g\_ui8TrfBuffer[5] == 0x04)  {  UartSendCString("https://");  }  }  **else**  {  ui8NdefMessageOffset = 0x00;  }  **if** (ui16NdefLength > (NFC\_FIFO\_SIZE-ui8NdefMessageOffset))  {  ui8NdefReadLength = NFC\_FIFO\_SIZE-ui8NdefMessageOffset;  }  **else**  {  ui8NdefReadLength = ui16NdefLength-ui8NdefMessageOffset+1;  }  **for** (ui8LoopCount = 0; ui8LoopCount < ui8NdefReadLength; ui8LoopCount++)  {  UartPutChar(g\_ui8TrfBuffer[ui8NdefMessageOffset++]);  }  UartPutChar(']');  UartPutCrlf();  UartPutCrlf();  **#endif**  }  **void** Ndef\_WriteText(**void**)  {  **#ifdef** ENABLE\_HOST  UartSendCString("Now Writing New NDEF Message.");  UartPutCrlf();  UartPutCrlf();  **#endif**  NDEF\_UpdateBinaryText();  }  **void** Ndef\_WriteUri(**void**)  {  **#ifdef** ENABLE\_HOST  UartSendCString("Now Writing New NDEF Message.");  UartPutCrlf();  UartPutCrlf();  **#endif**  NDEF\_UpdateBinaryUri();  }  **void** Ndef\_SetBlockNumberBit(bool bValue)  {  g\_bBlockNumberBit = bValue;  }  **void** NDEF\_UpdateBinaryText(**void**)  {  u08\_t ui8Offset = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0xF0; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xD6; // INS = update Binary  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Offset, P1  g\_ui8TrfBuffer[ui8Offset++] = 0x02; // Offset, P2  g\_ui8TrfBuffer[ui8Offset++] = 0x19; // Lc, length being written (all bytes)  g\_ui8TrfBuffer[ui8Offset++] = 0xD1; // MB = 1, ME = 1, Short Record, TNF = NFC Forum Well Known Type  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // Length of Record Type  g\_ui8TrfBuffer[ui8Offset++] = 0x15; // Length of Text being written (21 bytes, hardcoded for now)  g\_ui8TrfBuffer[ui8Offset++] = 0x54; // Text  g\_ui8TrfBuffer[ui8Offset++] = 0x02; // Language Length  g\_ui8TrfBuffer[ui8Offset++] = 0x65; // 'e' - For English  g\_ui8TrfBuffer[ui8Offset++] = 0x6E; // 'n' - For English  g\_ui8TrfBuffer[ui8Offset++] = 0x4E; // N  g\_ui8TrfBuffer[ui8Offset++] = 0x46; // F  g\_ui8TrfBuffer[ui8Offset++] = 0x43; // C  g\_ui8TrfBuffer[ui8Offset++] = 0x20; //  g\_ui8TrfBuffer[ui8Offset++] = 0x50; // P  g\_ui8TrfBuffer[ui8Offset++] = 0x6F; // o  g\_ui8TrfBuffer[ui8Offset++] = 0x77; // w  g\_ui8TrfBuffer[ui8Offset++] = 0x65; // e  g\_ui8TrfBuffer[ui8Offset++] = 0x72; // r  g\_ui8TrfBuffer[ui8Offset++] = 0x65; // e  g\_ui8TrfBuffer[ui8Offset++] = 0x64; // d  g\_ui8TrfBuffer[ui8Offset++] = 0x20; //  g\_ui8TrfBuffer[ui8Offset++] = 0x42; // B  g\_ui8TrfBuffer[ui8Offset++] = 0x79; // y  g\_ui8TrfBuffer[ui8Offset++] = 0x20; //  g\_ui8TrfBuffer[ui8Offset++] = 0x54; // T  g\_ui8TrfBuffer[ui8Offset++] = 0x49; // I  g\_ui8TrfBuffer[ui8Offset++] = 0x21; // !  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(10,20); // 10 millisecond TX timeout, 20 millisecond RX timeout  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  McuDelayMillisecond(5);  NDEF\_UpdateBinaryLength(0x19);  }  **void** NDEF\_UpdateBinaryUri(**void**)  {  u08\_t ui8Offset = 0;  g\_ui8TrfBuffer[ui8Offset++] = 0x8F; // Reset FIFO  g\_ui8TrfBuffer[ui8Offset++] = 0x91; // Send with CRC  g\_ui8TrfBuffer[ui8Offset++] = 0x3D; // Write Continuous  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // Length of packet in bytes - upper and middle nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x50; // Length of packet in bytes - lower and broken nibbles of transmit byte length  g\_ui8TrfBuffer[ui8Offset++] = 0x02 | g\_bBlockNumberBit; // I-Block PCB: Read Block 0 or Block 1, with CID = 0, NAD = 0, no chaining  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // CLA  g\_ui8TrfBuffer[ui8Offset++] = 0xD6; // INS = update Binary  g\_ui8TrfBuffer[ui8Offset++] = 0x00; // Offset, P1  g\_ui8TrfBuffer[ui8Offset++] = 0x02; // Offset, P2  g\_ui8TrfBuffer[ui8Offset++] = 0x0F; // Lc, length being written (all bytes)  g\_ui8TrfBuffer[ui8Offset++] = 0xD1; // MB = 1, ME = 1, Short Record, TNF = NFC Forum Well Known Type  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // Length of Record Type  g\_ui8TrfBuffer[ui8Offset++] = 0x0B; // Length of URI being written (11 bytes, hardcoded for now)  g\_ui8TrfBuffer[ui8Offset++] = 0x55; // URI  g\_ui8TrfBuffer[ui8Offset++] = 0x01; // URI Identifier: http://wwww.  g\_ui8TrfBuffer[ui8Offset++] = 0x74; // t  g\_ui8TrfBuffer[ui8Offset++] = 0x69; // i  g\_ui8TrfBuffer[ui8Offset++] = 0x2E; // .  g\_ui8TrfBuffer[ui8Offset++] = 0x63; // c  g\_ui8TrfBuffer[ui8Offset++] = 0x6F; // o  g\_ui8TrfBuffer[ui8Offset++] = 0x6D; // m  g\_ui8TrfBuffer[ui8Offset++] = 0x2F; // /  g\_ui8TrfBuffer[ui8Offset++] = 0x6E; // n  g\_ui8TrfBuffer[ui8Offset++] = 0x66; // f  g\_ui8TrfBuffer[ui8Offset++] = 0x63; // c  Trf797xRawWrite(&g\_ui8TrfBuffer[0], ui8Offset); // Issue the NDEF Command  Trf797xIrqWaitTimeout(10,20); // 10 millisecond TX timeout, 20 millisecond RX timeout  g\_bBlockNumberBit = !g\_bBlockNumberBit; // Toggle the PCB Block Number  McuDelayMillisecond(5);  NDEF\_UpdateBinaryLength(0x0F);  } |

|  |
| --- |
| **types.h** |
| /\*  \* {types.h}  \*  \* {Header File}  \*  \* Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  **#ifndef** \_TYPEDEF\_H\_  **#define** \_TYPEDEF\_H\_  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* STANDARD BITS  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **#define** BIT0 (0x0001)  **#define** BIT1 (0x0002)  **#define** BIT2 (0x0004)  **#define** BIT3 (0x0008)  **#define** BIT4 (0x0010)  **#define** BIT5 (0x0020)  **#define** BIT6 (0x0040)  **#define** BIT7 (0x0080)  **#define** BIT8 (0x0100)  **#define** BIT9 (0x0200)  **#define** BITA (0x0400)  **#define** BITB (0x0800)  **#define** BITC (0x1000)  **#define** BITD (0x2000)  **#define** BITE (0x4000)  **#define** BITF (0x8000)  **#define** **false** 0  **#define** **true** 1  //#define bool \_Bool  **typedef** **unsigned** **char** bool;  //================================================================  **typedef** **unsigned** **char** u08\_t;  **typedef** **unsigned** **short** u16\_t;  **typedef** **unsigned** **long** u32\_t;  **typedef** **signed** **char** s08\_t;  **typedef** **signed** **short** s16\_t;  **typedef** **signed** **long** s32\_t;  **#ifndef** STATUS\_FAIL  **#define** STATUS\_FAIL 0x00  **#endif**  **#ifndef** STATUS\_SUCCESS  **#define** STATUS\_SUCCESS 0x01  **#endif**  //===============================================================  **#endif** |

|  |
| --- |
| **cc3200v1p32.cmd** |
| --retain=g\_pfnVectors  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // The following command line options are set as part of the CCS project.  // If you are building using the command line, or for some reason want to  // define them here, you can uncomment and modify these lines as needed.  // If you are using CCS for building, it is probably better to make any such  // modifications in your CCS project and leave this file alone.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // The starting address of the application. Normally the interrupt vectors  // must be located at the beginning of the application.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  **#define** RAM\_BASE 0x20004000  /\* System memory map \*/  **MEMORY**  {  /\* Application uses internal RAM for program and data \*/  SRAM\_CODE (RWX) : origin = 0x20004000, length = 0x1C000  SRAM\_DATA (RWX) : origin = 0x20030000, length = 0x10000  }  /\* Section allocation in memory \*/  **SECTIONS**  {  .intvecs: > RAM\_BASE  .init\_array : > SRAM\_CODE  .vtable : > SRAM\_CODE  **.text** : > SRAM\_CODE  **.const** : > SRAM\_CODE  **.cinit** : > SRAM\_CODE  **.pinit** : > SRAM\_CODE  **.data** : > SRAM\_DATA  **.bss** : > SRAM\_DATA  **.sysmem** : > SRAM\_DATA  **.stack** : > SRAM\_DATA(HIGH)  } |

## **9.2 Web Server (C# language)**

|  |
| --- |
| **program.cs** |
| using Nancy.Hosting.Self;  using smart\_doorlock\_web\_server.Database;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server  {  class Program  {  private const string WEB\_SERVER\_PORT = "8203";  private const string TERMINATE\_STRING = "/Terminate";  private const string DEFAULT\_CONFIG\_FILE\_PATH = "config.cfg";  static void Main(string[] args)  {  ConfigHandler configHandler = new ConfigHandler(DEFAULT\_CONFIG\_FILE\_PATH);  if (!configHandler.IsConfigFileValid())  Console.WriteLine("Error loading default config file: {0}", DEFAULT\_CONFIG\_FILE\_PATH);  else  {  Console.WriteLine("Loading default config file: {0}", DEFAULT\_CONFIG\_FILE\_PATH);  configHandler.LoadConfig();  SDDatabase.Instance.LoadConnectionInfo(configHandler.DatabaseServer,  configHandler.DatabasePort,  configHandler.DatabaseUserName,  configHandler.DatabasePassword,  configHandler.DatabaseName);  }  MqttBroker mqttBroker = MqttBroker.Instance;  #if TRACE  //MqttUtility.Trace.TraceLevel = MqttUtility.TraceLevel.Verbose | MqttUtility.TraceLevel.Frame;  //MqttUtility.Trace.TraceListener = (f, a) => System.Diagnostics.Trace.WriteLine(System.String.Format(f, a));  #endif  mqttBroker.Start();    var uri = new Uri("http://localhost:" + WEB\_SERVER\_PORT + "/");  var config = new HostConfiguration  {  UrlReservations = { CreateAutomatically = true },  AllowChunkedEncoding = false  };  var host = new NancyHost(config, uri);  try  {  host.Start();  Console.Write("Smart Doorlock Web Server\n" +  "\t\"" + uri + "\"\n" +  "To quit, input \"" + TERMINATE\_STRING + "\".\n\n");  do Console.Write("> "); while (Console.ReadLine() != TERMINATE\_STRING);  }  catch (Exception e)  {  Console.WriteLine("Unhandled exception has been occured!\n"  + e.Message);  Console.ReadKey(true);  }  finally  {  host.Stop();  mqttBroker.Stop();  }  }  }  } |

|  |
| --- |
| **SDCommandHandler.cs** |
| using smart\_doorlock\_web\_server.Database;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server  {  public static class SDCommandHandler  {  public static void HandleCommand(string payload)  {  string[] splitPayload = payload.Split('|');  if (splitPayload.Length < 2)  return;  string cmd = splitPayload[0];  if (cmd.Equals("LOG",StringComparison.InvariantCultureIgnoreCase))  {  string type = splitPayload[1];  string data = splitPayload[2];  SDAccessLogDTO accessLog = new SDAccessLogDTO();  accessLog.AccessDateTime = DateTime.Now;  accessLog.Data = data;  accessLog.Type = type;  SDAccessLogDAL.InsertAccess(accessLog);  }  }  }  } |

|  |
| --- |
| **WebServerMainModule.cs** |
| using Nancy;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server  {  public class WebServerMainModule : NancyModule  {  MainPageSL mainPageSl = MainPageSL.Instance;  LogSL logSl = LogSL.Instance;  MqttBroker mqttBroker = MqttBroker.Instance;  public WebServerMainModule()  {  Get["/"] = \_ =>  {  return View["Views/About.sshtml"];  };  Get["/Control"] = \_ =>  {  return View["Views/Control.sshtml", mainPageSl.GetMainPageData()];  };  Get["/Log"] = \_ =>  {  return View["Views/Log.sshtml", logSl.GetAccessLog()];  };  Post["/Subscribe"] = x =>  {  string clientId = Request.Form.clientId;  string topic = Request.Form.topic;  if (mqttBroker.Subscribe(clientId, topic))  return "OK";  return "Fail";  };  Post["/Publish"] = x =>  {  string clientId = Request.Form.clientId;  string topic = Request.Form.topic;  string message = Request.Form.message;  mqttBroker.Publish(topic, message);    return "OK";  };  }  }  } |

|  |
| --- |
| **MqttBroker.cs** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using uPLibrary.Networking.M2Mqtt.Messages;  using uPLibrary.Networking.M2Mqtt.Exceptions;  using uPLibrary.Networking.M2Mqtt.Managers;  using uPLibrary.Networking.M2Mqtt.Communication;  using uPLibrary.Networking.M2Mqtt.Session;  using System.Security.Cryptography.X509Certificates;  using System.Net.Security;  using uPLibrary.Networking.M2Mqtt;  namespace smart\_doorlock\_web\_server  {  /// <summary>  /// MQTT broker business logic  /// </summary>  public class MqttBroker  {  internal const byte PROTOCOL\_VERSION\_V3\_1 = 0x03;  internal const byte PROTOCOL\_VERSION\_V3\_1\_1 = 0x04; // [v.3.1.1]  // max length for client id (removed in 3.1.1)  internal const int CLIENT\_ID\_MAX\_LENGTH = 23;  // MQTT broker settings  private MqttSettings settings;  // clients connected list  private MqttClientCollection clients;  // reference to publisher manager  private MqttPublisherManager publisherManager;  // reference to subscriber manager  private MqttSubscriberManager subscriberManager;  // reference to session manager  private MqttSessionManager sessionManager;  // reference to User Access Control manager  private MqttUacManager uacManager;  // MQTT communication layer  private IMqttCommunicationLayer commLayer;  private static readonly MqttBroker \_instance = new MqttBroker();  public static MqttBroker Instance  {  get {  return \_instance;  }  }  /// <summary>  /// User authentication method  /// </summary>  public MqttUserAuthenticationDelegate UserAuth  {  get { return this.uacManager.UserAuth; }  set { this.uacManager.UserAuth = value; }  }  /// <summary>  /// Constructor (TCP/IP communication layer on port 1883 and default settings)  /// </summary>  private MqttBroker()  : this(new MqttTcpCommunicationLayer(MqttSettings.MQTT\_BROKER\_DEFAULT\_PORT), MqttSettings.Instance)  {  }  /// <summary>  /// Constructor  /// </summary>  /// <param name="commLayer">Communication layer to use (TCP)</param>  /// <param name="settings">Broker settings</param>  private MqttBroker(IMqttCommunicationLayer commLayer, MqttSettings settings)  {  // MQTT broker settings  this.settings = settings;  // MQTT communication layer  this.commLayer = commLayer;  this.commLayer.ClientConnected += commLayer\_ClientConnected;  // create managers (publisher, subscriber, session and UAC)  this.subscriberManager = new MqttSubscriberManager();  this.sessionManager = new MqttSessionManager();  this.publisherManager = new MqttPublisherManager(this.subscriberManager, this.sessionManager);  this.uacManager = new MqttUacManager();  this.clients = new MqttClientCollection();  }  /// <summary>  /// Start broker  /// </summary>  public void Start()  {  this.commLayer.Start();  this.publisherManager.Start();  }  /// <summary>  /// Stop broker  /// </summary>  public void Stop()  {  this.commLayer.Stop();  this.publisherManager.Stop();  // close connection with all clients  foreach (MqttClient client in this.clients)  {  client.Close();  }  }  public void Publish(string topic, string message)  {  MqttMsgPublish publishMsg = new MqttMsgPublish(topic, Encoding.UTF8.GetBytes(message),false,1,false);  publisherManager.Publish(publishMsg);  }  public MqttClientCollection GetClientList()  {  return clients;  }  public bool Subscribe(string clientId, string topic)  {  MqttClient client = GetClient(clientId);  if (client == null)  return false;  byte[] qosLevel = new byte[] {MqttMsgBase.QOS\_LEVEL\_AT\_LEAST\_ONCE};  string[] topicParam = new string[] { topic };  client.Subscribe(topicParam,qosLevel);  return true;  }  /// <summary>  /// Close a client  /// </summary>  /// <param name="client">Client to close</param>  private void CloseClient(MqttClient client)  {  if (this.clients.Contains(client))  {  // if client is connected and it has a will message  if (!client.IsConnected && client.WillFlag)  {  // create the will PUBLISH message  MqttMsgPublish publish =  new MqttMsgPublish(client.WillTopic, Encoding.UTF8.GetBytes(client.WillMessage), false, client.WillQosLevel, false);  // publish message through publisher manager  this.publisherManager.Publish(publish);  }  // if not clean session  if (!client.CleanSession)  {  List<MqttSubscription> subscriptions = this.subscriberManager.GetSubscriptionsByClient(client.ClientId);  if ((subscriptions != null) && (subscriptions.Count > 0))  {  this.sessionManager.SaveSession(client.ClientId, client.Session, subscriptions);  // TODO : persist client session if broker close  }  }  // delete client from runtime subscription  this.subscriberManager.Unsubscribe(client);  // close the client  client.Close();  // remove client from the collection  this.clients.Remove(client);  }  }  void commLayer\_ClientConnected(object sender, MqttClientConnectedEventArgs e)  {  // register event handlers from client  e.Client.MqttMsgDisconnected += Client\_MqttMsgDisconnected;  e.Client.MqttMsgPublishReceived += Client\_MqttMsgPublishReceived;  e.Client.MqttMsgConnected += Client\_MqttMsgConnected;  e.Client.MqttMsgSubscribeReceived += Client\_MqttMsgSubscribeReceived;  e.Client.MqttMsgUnsubscribeReceived += Client\_MqttMsgUnsubscribeReceived;  e.Client.ConnectionClosed += Client\_ConnectionClosed;  // add client to the collection  this.clients.Add(e.Client);  // start client threads  e.Client.Open();  }  void Client\_MqttMsgPublishReceived(object sender, MqttMsgPublishEventArgs e)  {  MqttClient client = (MqttClient)sender;  // create PUBLISH message to publish  // [v3.1.1] DUP flag from an incoming PUBLISH message is not propagated to subscribers  // It should be set in the outgoing PUBLISH message based on transmission for each subscriber  MqttMsgPublish publish = new MqttMsgPublish(e.Topic, e.Message, false, e.QosLevel, e.Retain);  string payload = Encoding.ASCII.GetString(e.Message);  Console.WriteLine("Publish Received: [{0},{1}]",e.Topic,payload);  SDCommandHandler.HandleCommand(payload);  // publish message through publisher manager  this.publisherManager.Publish(publish);  }  void Client\_MqttMsgUnsubscribeReceived(object sender, MqttMsgUnsubscribeEventArgs e)  {  MqttClient client = (MqttClient)sender;  for (int i = 0; i < e.Topics.Length; i++)  {  // unsubscribe client for each topic requested  this.subscriberManager.Unsubscribe(e.Topics[i], client);  }  try  {  // send UNSUBACK message to the client  client.Unsuback(e.MessageId);  }  catch (MqttCommunicationException)  {  this.CloseClient(client);  }  }  void Client\_MqttMsgSubscribeReceived(object sender, MqttMsgSubscribeEventArgs e)  {  MqttClient client = (MqttClient)sender;  Console.WriteLine("Subscribe Received");  for (int i = 0; i < e.Topics.Length; i++)  {  // TODO : business logic to grant QoS levels based on some conditions ?  // now the broker granted the QoS levels requested by client  // subscribe client for each topic and QoS level requested  this.subscriberManager.Subscribe(e.Topics[i], e.QoSLevels[i], client);  Console.WriteLine("Topic: " + e.Topics[i]);  }  try  {  // send SUBACK message to the client  client.Suback(e.MessageId, e.QoSLevels);  for (int i = 0; i < e.Topics.Length; i++)  {  // publish retained message on the current subscription  this.publisherManager.PublishRetaind(e.Topics[i], client.ClientId);  }  }  catch (MqttCommunicationException)  {  this.CloseClient(client);  }  }  void Client\_MqttMsgConnected(object sender, MqttMsgConnectEventArgs e)  {  // [v3.1.1] session present flag  bool sessionPresent = false;  // [v3.1.1] generated client id for client who provides client id zero bytes length  string clientId = null;  MqttClient client = (MqttClient)sender;  // verify message to determine CONNACK message return code to the client  byte returnCode = this.MqttConnectVerify(e.Message);  // [v3.1.1] if client id is zero length, the broker assigns a unique identifier to it  clientId = (e.Message.ClientId.Length != 0) ? e.Message.ClientId : Guid.NewGuid().ToString();  // connection "could" be accepted  if (returnCode == MqttMsgConnack.CONN\_ACCEPTED)  {  // check if there is a client already connected with same client Id  MqttClient clientConnected = this.GetClient(clientId);  // force connection close to the existing client (MQTT protocol)  if (clientConnected != null)  {  this.CloseClient(clientConnected);  }  }  try  {  // connection accepted, load (if exists) client session  if (returnCode == MqttMsgConnack.CONN\_ACCEPTED)  {  // check if not clean session and try to recovery a session  if (!e.Message.CleanSession)  {  // create session for the client  MqttClientSession clientSession = new MqttClientSession(clientId);  // get session for the connected client  MqttBrokerSession session = this.sessionManager.GetSession(clientId);  // set inflight queue into the client session  if (session != null)  {  clientSession.InflightMessages = session.InflightMessages;  // [v3.1.1] session present flag  if (client.ProtocolVersion == MqttProtocolVersion.Version\_3\_1\_1)  sessionPresent = true;  }  // send CONNACK message to the client  client.Connack(e.Message, returnCode, clientId, sessionPresent);  // load/inject session to the client  client.LoadSession(clientSession);  if (session != null)  {  // set reference to connected client into the session  session.Client = client;  // there are saved subscriptions  if (session.Subscriptions != null)  {  // register all subscriptions for the connected client  foreach (MqttSubscription subscription in session.Subscriptions)  {  this.subscriberManager.Subscribe(subscription.Topic, subscription.QosLevel, client);  // publish retained message on the current subscription  this.publisherManager.PublishRetaind(subscription.Topic, clientId);  }  }  // there are saved outgoing messages  if (session.OutgoingMessages.Count > 0)  {  // publish outgoing messages for the session  this.publisherManager.PublishSession(session.ClientId);  }  }  }  // requested clean session  else  {  // send CONNACK message to the client  client.Connack(e.Message, returnCode, clientId, sessionPresent);  this.sessionManager.ClearSession(clientId);  }  }  else  {  // send CONNACK message to the client  client.Connack(e.Message, returnCode, clientId, sessionPresent);  }  }  catch (MqttCommunicationException)  {  this.CloseClient(client);  }  }  void Client\_MqttMsgDisconnected(object sender, EventArgs e)  {  MqttClient client = (MqttClient)sender;  // close the client  this.CloseClient(client);  }  void Client\_ConnectionClosed(object sender, EventArgs e)  {  MqttClient client = (MqttClient)sender;  // close the client  this.CloseClient(client);  }  /// <summary>  /// Check CONNECT message to accept or not the connection request  /// </summary>  /// <param name="connect">CONNECT message received from client</param>  /// <returns>Return code for CONNACK message</returns>  private byte MqttConnectVerify(MqttMsgConnect connect)  {  byte returnCode = MqttMsgConnack.CONN\_ACCEPTED;  // unacceptable protocol version  if ((connect.ProtocolVersion != PROTOCOL\_VERSION\_V3\_1) &&  (connect.ProtocolVersion != PROTOCOL\_VERSION\_V3\_1\_1))  returnCode = MqttMsgConnack.CONN\_REFUSED\_PROT\_VERS;  else  {  // client id length exceeded (only for old MQTT 3.1)  if ((connect.ProtocolVersion == PROTOCOL\_VERSION\_V3\_1) &&  (connect.ClientId.Length > CLIENT\_ID\_MAX\_LENGTH))  returnCode = MqttMsgConnack.CONN\_REFUSED\_IDENT\_REJECTED;  else  {  // [v.3.1.1] client id zero length is allowed but clean session must be true  if ((connect.ClientId.Length == 0) && (!connect.CleanSession))  returnCode = MqttMsgConnack.CONN\_REFUSED\_IDENT\_REJECTED;  else  {  // check user authentication  if (!this.uacManager.UserAuthentication(connect.Username, connect.Password))  returnCode = MqttMsgConnack.CONN\_REFUSED\_USERNAME\_PASSWORD;  // server unavailable and not authorized ?  else  {  // TODO : other checks on CONNECT message  }  }  }  }  return returnCode;  }  /// <summary>  /// Return reference to a client with a specified Id is already connected  /// </summary>  /// <param name="clientId">Client Id to verify</param>  /// <returns>Reference to client</returns>  private MqttClient GetClient(string clientId)  {  var query = from c in this.clients  where c.ClientId == clientId  select c;  return query.FirstOrDefault();  }  }  } |

|  |
| --- |
| **ConfigHandler.cs** |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server  {  public class ConfigHandler  {  public string DatabaseServer { get; private set; }  public uint DatabasePort { get; private set; }  public string DatabaseUserName { get; private set; }  public string DatabasePassword { get; private set; }  public string DatabaseName { get; private set; }  private string \_configFilePath;  private readonly List<string> \_fileContent = new List<string>();  public ConfigHandler(string configFilePath)  {  \_configFilePath = configFilePath;  if (File.Exists(configFilePath))  {  \_fileContent = new List<string>(File.ReadAllLines(configFilePath));  }  }  public bool IsConfigFileValid()  {  return \_fileContent.Any();  }  public void LoadConfig()  {  uint parsedInt;  DatabaseServer = GetConfigString("databaseserver");  if (uint.TryParse(GetConfigString("databaseport"), out parsedInt))  {  DatabasePort = parsedInt;  }  DatabaseUserName = GetConfigString("databaseusername");  DatabasePassword = GetConfigString("databasepassword");  DatabaseName = GetConfigString("databasename");  }  private string GetConfigString(string key)  {  foreach (string line in \_fileContent)  {  if (String.IsNullOrEmpty(line))  continue;  if (line[0] == '#') //Config Comment  continue;  string[] configValueArray = line.Split('=');  if (configValueArray.Length != 2)  continue;  if (configValueArray[0] == key)  return configValueArray[1];  }  return String.Empty;  }  }  } |

|  |
| --- |
| **LogData.cs** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server  {  public class LogData  {  public DateTime AccessDateTime { get; set; }  public string Type { get; set; }  public string AccessID { get; set; }  }  } |

|  |
| --- |
| **LogSL.cs** |
| using smart\_doorlock\_web\_server.Database;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server  {  public class LogSL  {  private static readonly LogSL \_instance = new LogSL();  public static LogSL Instance  {  get  {  return \_instance;  }  }  private LogSL() { }  public List<LogData> GetAccessLog()  {  List<LogData> dataList = new List<LogData>();  List<SDAccessLogDTO> accessLogDTO = SDAccessLogDAL.GetAccessLog().ToList();  foreach (SDAccessLogDTO accessLog in accessLogDTO)  {  LogData logData = new LogData();  logData.AccessDateTime = accessLog.AccessDateTime;  logData.Type = accessLog.Type;  logData.AccessID = accessLog.Data;  dataList.Add(logData);  }  return dataList;  }  }  } |

|  |
| --- |
| **SDAccessLogDAL.cs** |
| using MySql.Data.MySqlClient;  using System;  using System.Collections.Generic;  using System.Data;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server.Database  {  public static class SDAccessLogDAL  {  public static void InsertAccess(SDAccessLogDTO data)  {  try  {  using (SDDatabase.Instance.Connection)  {  SDDatabase.Instance.Connection.Open();  using (MySqlCommand cmd = new MySqlCommand("INSERT INTO access\_log(AccessDateTime, Type, Data) VALUES(@acdt,@type,@data)", SDDatabase.Instance.Connection))  {  cmd.Parameters.AddWithValue("@acdt", data.AccessDateTime);  cmd.Parameters.AddWithValue("@type", data.Type);  cmd.Parameters.AddWithValue("@data", data.Data);  cmd.ExecuteNonQuery();  }  }  }  catch (Exception ex)  {  Console.WriteLine("InsertAccess Exception: " + ex);  }  }  public static IEnumerable<SDAccessLogDTO> GetAccessLog()  {  List<SDAccessLogDTO> accessList = new List<SDAccessLogDTO>();  try  {  using (SDDatabase.Instance.Connection)  {  SDDatabase.Instance.Connection.Open();  using (MySqlCommand cmd = new MySqlCommand("SELECT LogID, AccessDateTime, Type, Data FROM access\_log", SDDatabase.Instance.Connection))  {  cmd.ExecuteNonQuery();  using (MySqlDataAdapter adap = new MySqlDataAdapter())  {  DataTable dt = new DataTable();  adap.SelectCommand = cmd;  adap.Fill(dt);  foreach (DataRow dr in dt.Rows)  {  SDAccessLogDTO logInfo = new SDAccessLogDTO();  logInfo.LogID = (int)dr["LogID"];  logInfo.AccessDateTime = (DateTime)dr["AccessDateTime"];  logInfo.Type = (string)dr["Type"];  logInfo.Data = (string)dr["Data"];  accessList.Add(logInfo);  }  }  }  }  }  catch (Exception ex)  {  Console.WriteLine("GetAccessLog Exception: " + ex);  }    return accessList;  }  }  } |

|  |
| --- |
| **SDAccessLogDTO.cs** |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server.Database  {  public class SDAccessLogDTO  {  public int LogID { get; set; }  public DateTime AccessDateTime { get; set; }  public string Data { get; set; }  public string Type { get; set; }  }  } |

|  |
| --- |
| **SDDatabase.cs** |
| using MySql.Data.MySqlClient;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace smart\_doorlock\_web\_server.Database  {  public class SDDatabase  {  private MySqlConnection \_connection = null;  private static readonly SDDatabase \_instance = new SDDatabase();  public static SDDatabase Instance {  get { return \_instance; }  }  private SDDatabase()  {  }  public MySqlConnection Connection  {  get { return \_connection; }  }  public void LoadConnectionInfo(string databaseAddr, uint databasePort, string userName, string password, string databaseName)  {  if (String.IsNullOrEmpty(databaseAddr))  {  throw new ArgumentNullException("Database address has not been provided");  }  if (databasePort <= 0 || databasePort >= 65535)  {  throw new ArgumentOutOfRangeException("Invalid database port given: " + databasePort);  }  if (String.IsNullOrEmpty(databaseName) || String.IsNullOrEmpty(userName) || String.IsNullOrEmpty(password))  {  throw new ArgumentNullException("Database name, user name and/or password has not been provided");  }    string connString = string.Format("Server={0}; port={1}; database={2}; UID={3}; password={4}",  databaseAddr,databasePort,databaseName,userName,password);  \_connection = new MySqlConnection(connString);  }  }  } |

|  |
| --- |
| **about.sshtml** |
| @Master['MasterPage']  @Section['Content']  <link href="./Content/css/About.css" rel="stylesheet">  <div class="container">  <div id="aboutCarousel" class="carousel slide" data-ride="carousel">  <!-- Indicators -->  <ol class="carousel-indicators">  <li data-target="#aboutCarousel" data-slide-to="0" class="active"></li>  <li data-target="#aboutCarousel" data-slide-to="1"></li>  <li data-target="#aboutCarousel" data-slide-to="2"></li>  </ol>  <!-- Wrapper for slides -->  <div class="carousel-inner" role="listbox">  <div class="item active">  <img src="/Content/images/AboutCarousel1.png">  </div>  <div class="item">  <img src="/Content/images/AboutCarousel2.png">  </div>  <div class="item">  <img src="/Content/images/AboutCarousel3.png">  </div>  </div>  <!-- Left and right controls -->  <a class="left carousel-control" href="#aboutCarousel" role="button" data-slide="prev">  <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>  <span class="sr-only">Previous</span>  </a>  <a class="right carousel-control" href="#aboutCarousel" role="button" data-slide="next">  <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>  <span class="sr-only">Next</span>  </a>  </div>  <ul class="nav nav-pills nav-justified" role="tablist" id="myPill">  <li role="presentation" class="active"><a href="#about" aria-controls="home" role="tab" data-toggle="tab">About</a></li>  <li role="presentation"><a href="#spec" aria-controls="profile" role="tab" data-toggle="tab">Project Spec.</a></li>  <li role="presentation"><a href="#circuit" aria-controls="profile" role="tab" data-toggle="tab">Circuit</a></li>  <li role="presentation"><a href="#source" aria-controls="profile" role="tab" data-toggle="tab">Source Code</a></li>  <li role="presentation"><a href="#demo" aria-controls="profile" role="tab" data-toggle="tab">Demo</a></li>  <li role="presentation"><a href="#files" aria-controls="profile" role="tab" data-toggle="tab">Documentation</a></li>  </ul>  <div class="jumbotron">  <div class="tab-content">  <div role="tabpanel" class="tab-pane active" id="about">  <h2>What is Smart Doorlock?</h2>  <br/>  <p class="lead">Smart Doorlock is a specialized electronic doorlock that takes advantage of NFC & IOT technology to better secure your home.</p>  <p class="lead">Smart Doorlock allows the user to:</p>  <ul>  <li>Operate the doorlock using Android phones via NFC P2P functionality</li>  <li>Operate the doorlock using PC, tablets and any smart phones over the internet using IoT technology</li>  <li>Grant and revoke access permissions to each individual personnels</li>  <li>Manage access logs for security audits</li>  <li>and more...</li>  </ul>  <br/>  <br/>  <br/>  Brought to you by Sokwhan Huh. All copyrights reserved.</p>  </div>  <div role="tabpanel" class="tab-pane" id="spec">  <h2>Project Specifications</h2>  <br/>  <ul>  <li>Interface via NFC in P2P mode using Android Beam technology</li>  <li>Interface via IoT to a cloud server over MQTT Protocol</li>  <li>NFC communication range: 5 cm maximum (from smartphone to Smart Doorlock)</li>  <li>Wi-Fi communication range: 50 meters maximum (TX 14.5dBM, RX -74.0dBM at 54 OFDM)</li>  <li>AC Power (100V-240V @0.8A 50-60Hz) with DC battery backup</li>  <li>Minimum backup battery life of 24 hours</li>  <li>Supported Android versions: 4.0 (Ice Cream Sandwich) and above</li>  </ul>  <br/>  <h2>Block Diagram</h2>  <br/>  <img src="/Content/images/BlockDiagram.png">  <br/>  <h2>Hardware Components</h2>  <br/>  <ul>  <li>Microcontroller: Texas Instruments CC3200</li>  <li>NFC Transceiver IC: Texas Instruments TRF7970A</li>  <li>Keypad: Storm Interface’s Storm 720GFXI Series (7207-121W20)</li>  <li>LCD: Newhaven's NHD-0420D3Z-NSW-BBW-V3-ND </li>  <li>Power Supply: XP Power's ECL30UD03-S 5V/12V 30W Dual Output</li>  <li>Solenoid Motor: Lock-style Solenoid - 12VDC</li>  </ul>  <h2>Software</h2>  <br/>  <ul>  <li>Programming Language: C (Microcontroller), Java (Android Application), C# (Web Server)</li>  <li>Web Server Framework: NancyFX</li>  <li>Web Server Database: MySQL</li>  </ul>  </div>  <div role="tabpanel" class="tab-pane" id="circuit">  <h2>Circuit Diagram</h2>  <br/>  Coming soon...  </div>  <div role="tabpanel" class="tab-pane" id="source">  <h2>Source Codes</h2>  <br/>  <p class="lead">Smart Doorlock is an open source project. All source codes are available on bitbucket repository:</p>  <ul>  <li>Smart Doorlock (MCU): <a href="https://bitbucket.org/l46kok/smart-doorlock">https://bitbucket.org/l46kok/smart-doorlock</a></li>  <li>Smart Doorlock Web Server: <a href="https://bitbucket.org/l46kok/smart-doorlock-web-server">https://bitbucket.org/l46kok/smart-doorlock-web-server</a></li>  <li>Smart Doorlock Android Application: <a href="https://bitbucket.org/l46kok/smart-doorlock-android-app">https://bitbucket.org/l46kok/smart-doorlock-android-app</a></li>  </ul>  </div>  <div role="tabpanel" class="tab-pane" id="demo">  <h2>Demo Videos</h2>  <br/>  <p class="lead">NFC Operation</p>  <iframe width="560" height="315" src="https://www.youtube.com/embed/rgIzpaI3vP8" frameborder="0" allowfullscreen></iframe>  <iframe width="560" height="315" src="https://www.youtube.com/embed/iMTLewEwC10" frameborder="0" allowfullscreen></iframe>  <br/>  <p class="lead">IoT Operation</p>  <iframe width="560" height="315" src="https://www.youtube.com/embed/ahTHVgW4CEE" frameborder="0" allowfullscreen></iframe>  <iframe width="560" height="315" src="https://www.youtube.com/embed/Ru\_4vKUZAZM" frameborder="0" allowfullscreen></iframe>  </div>  <div role="tabpanel" class="tab-pane" id="files">  <h2>Documentation</h2>  <br/>  <p class="lead">Presentation</p>  <ul>  <li><a href="/Content/files/Smart Doorlock Presentation.pptx">PPT presentation</a></li>  </ul>  </div>  </div>  </div>  </div>  @EndSection |

|  |
| --- |
| **Control.sshtml** |
| @Master['MasterPage']  @Section['Content']  <link href="./Content/css/Control.css" rel="stylesheet">  <!-- Begin page content -->  <div class="container">  <div class="page-header">  <h1>Doorlock Status</h1>  </div>    <table id="myTable" class="tablesorter">  <thead>  <tr>  <th>Doorlock No.</th>  <th>Client Id</th>  <th>Protocol Ver.</th>  <th>Status</th>  </tr>  </thead>  <tbody class="centerText">  @Each.Model  <tr>  <td rowspan="3">@Current.DoorlockNo</td>  <td>@Current.ClientId</td>  <td>@Current.ProtocolVer</td>  <td>Active</td>  </tr>  <tr>  <td colspan="2" class="txtInput"><input type="text" id="inputSubscribeTopic@Current.ClientId" class="form-control" id="usr" value="/SmartDoorlock/Log" placeholder="Topic"></td>  <td><button id="btnSubscribe" data-cid="@Current.ClientId" type="button" class="btn btn-success btnStyle">Subscribe</button></td>  </tr>  <tr>  <td class="txtInput"><input id="inputPublishTopic@Current.ClientId" type="text" class="form-control" id="usr" placeholder="Topic" value="/SmartDoorlock/DoorControl"></td>  <td class="txtInput"><input id="inputPublishMessage@Current.ClientId" type="text" class="form-control" id="usr" placeholder="Message" value="Open"></td>  <td><button id="btnPublish" data-cid="@Current.ClientId" type="button" class="btn btn-info btnStyle">Publish</button></td>  </tr>  @EndEach    </tbody>  </table>    <h1>MQTT Log</h1>  <textarea id="mqttLog" readonly class="form-control" rows="15"></textarea>  </div>  <script language="JavaScript" src="/Content/scripts/jquery.tablesorter.js"></script>  <script language="JavaScript" src="/Content/scripts/Control.js"></script>  @EndSection |

|  |
| --- |
| **Log.sshtml** |
| @Master['MasterPage']  @Section['Content']  <link href="./Content/css/Log.css" rel="stylesheet">  <!-- Begin page content -->  <div class="container">  <div class="page-header">  <h1>Access Log</h1>  </div>    <table id="logTable" class="tablesorter">  <thead>  <tr>  <th>Access Date</th>  <th>Type</th>  <th>Access ID</th>  </tr>  </thead>  <tbody class="centerText">  @Each.Model  <tr>  <td>@Current.AccessDateTime</td>  <td><b>@Current.Type</b></td>  <td>@Current.AccessID</td>  </tr>  @EndEach  </tbody>  </table>  </div>  <script language="JavaScript" src="/Content/scripts/jquery.tablesorter.js"></script>  <script language="JavaScript" src="/Content/scripts/Log.js"></script>  @EndSection |

|  |
| --- |
| **MasterPage.sshtml** |
| <!DOCTYPE html>  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <title>Smart Doorlock</title>  <link href="/Content/css/MasterPage.css" rel="stylesheet">  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">  <script src="https://code.jquery.com/jquery-2.2.4.min.js" integrity="sha256-BbhdlvQf/xTY9gja0Dq3HiwQF8LaCRTXxZKRutelT44=" crossorigin="anonymous"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>  </head>  <body>  <nav class="navbar navbar-default navbar-fixed-top">  <div class="container">  <div class="navbar-header">  <a class="navbar-brand" href="/">Smart Doorlock</a>  </div>  <div id="navbar" class="navbar-collapse collapse">  <ul class="nav navbar-nav">  <!--<li class="active"><a href="/">Home</a></li>-->  <li><a href="/">Home</a></li>  <li><a href="/Control">Control</a></li>  <li><a href="/Log">Log</a></li>  </ul>  </div>  </div>  </nav>  <!-- Wrap all page content here -->  <div id="wrap">  @Section['Content']  </div>  <div id="footer">  <div class="container">  <p class="text-muted">(C) 2016 Sokwhan Huh.</p>  </div>  </div>  </body>  </html> |

|  |
| --- |
| **About.css** |
| @media (min-width: 1000px) {  .container{  max-width: 920px;  max-height: 400px;  }  }  .centerText {  text-align: center;  vertical-align: middle !important;  } |

|  |
| --- |
| **Control.css** |
| @media (min-width: 1200px) {  .container{  max-width: 1000px;  }  }  /\* tables \*/  table.tablesorter {  font-family:inherit;  border: 1px solid #CDD2D2;  margin:10px 0pt 15px;  width: 100%;  text-align: center;  }  table.tablesorter thead tr th, table.tablesorter tfoot tr th {  background-color: #e6EEEE;  border: 1px solid #CDD2D2;  padding: 4px;  white-space:nowrap;  width:1%;  }  table.tablesorter thead tr .header {  background-image: url(tblColArrow.gif);  background-repeat: no-repeat;  background-position: center right;  cursor: pointer;  }  table.tablesorter tbody td {  color: #3D3D3D;  padding: 4px;  background-color: #FFF;  vertical-align: middle;  border: 1px solid #CDD2D2;  white-space:nowrap;  width:0;  }  table.tablesorter thead tr .headerSortUp {  background-image: url(tblColAsc.gif);  }  table.tablesorter thead tr .headerSortDown {  background-image: url(tblColDesc.gif);  }  table.tablesorter thead tr .headerSortDown, table.tablesorter thead tr .headerSortUp {  background-color: #8dbdd8;  }  table.tablesorter .winrateheader {  min-width: 180px;  }  .txtInput {  min-width: 180px;  }  .btnStyle {  width: 100px;  }  .leftText {  text-align: left !important;  } |

|  |
| --- |
| **Log.css** |
| @media (min-width: 1200px) {  .container{  max-width: 1000px;  }  }  /\* tables \*/  table.tablesorter {  font-family:inherit;  border: 1px solid #CDD2D2;  margin:10px 0pt 15px;  width: 100%;  text-align: center;  }  table.tablesorter thead tr th, table.tablesorter tfoot tr th {  background-color: #e6EEEE;  border: 1px solid #CDD2D2;  padding: 4px;  white-space:nowrap;  width:1%;  }  table.tablesorter thead tr .header {  background-image: url(tblColArrow.gif);  background-repeat: no-repeat;  background-position: center right;  cursor: pointer;  }  table.tablesorter tbody td {  color: #3D3D3D;  padding: 4px;  background-color: #FFF;  vertical-align: middle;  border: 1px solid #CDD2D2;  white-space:nowrap;  width:0;  }  table.tablesorter thead tr .headerSortUp {  background-image: url(tblColAsc.gif);  }  table.tablesorter thead tr .headerSortDown {  background-image: url(tblColDesc.gif);  }  table.tablesorter thead tr .headerSortDown, table.tablesorter thead tr .headerSortUp {  background-color: #8dbdd8;  }  table.tablesorter .winrateheader {  min-width: 180px;  }  .txtInput {  min-width: 180px;  }  .btnStyle {  width: 100px;  }  .leftText {  text-align: left !important;  } |

|  |
| --- |
| **MasterPage.css** |
| body {  margin-bottom: 60px;  }  body > .container {  padding: 60px 15px 0;  }  .container .text-muted {  margin: 20px 0;  }  /\* Sticky footer styles  -------------------------------------------------- \*/  html,  body {  height: 100%;  overflow: hidden;  /\* The html and body elements cannot have any padding or margin. \*/  }  /\* Wrapper for page content to push down footer \*/  #wrap {  height: calc(100% - 120px);  overflow-y: scroll;  padding: 0px 10%;  margin-top: 60px;  }  /\* Set the fixed height of the footer here \*/  #footer {  height: 60px;  background-color: #f5f5f5;  position: fixed;  bottom: 0;  width: 100%;  } |

|  |
| --- |
| **Control.js** |
| function appendToLog(txt) {  var log = $("#mqttLog");  log.val(log.val() + txt + "\n");  }  $(document).ready(function () {  $("#myTable").tablesorter();  $("#btnSubscribe").click(function () {  var btnElement = $(this);  var clientId = btnElement.attr("data-cid");  var txtElement = $("#inputSubscribeTopic" + clientId);  var data = { "clientId": clientId, "topic": txtElement.val() }  $.ajax({  type: "POST",  url: "/Subscribe",  data: data,  success: function (data) { alert(data); },  dataType: "json"  });  });  $("#btnPublish").click(function () {  var btnElement = $(this);  var clientId = btnElement.attr("data-cid");  var txtTopic = $("#inputPublishTopic" + clientId);  var txtMessage = $("#inputPublishMessage" + clientId);  var data = { "clientId": clientId, "topic": txtTopic.val(), "message": txtMessage.val() }  $.ajax({  type: "POST",  url: "/Publish",  data: data,  dataType: "json"  });  appendToLog("Published to topic: " + txtTopic.val() + ", message: " + txtMessage.val())  });  }); |

|  |
| --- |
| **Log.js** |
| $(document).ready(function () {  $("#logTable").tablesorter();  }); |

|  |
| --- |
| **jquery.tablesorter.js** |
| /\*  \*  \* TableSorter 2.0  \* Version 2.0.5b  \* @requires jQuery v1.2.3  \*  \*/  /\*\*  \*  \* @description Create a sortable table with multi-column sorting capabilitys  \*  \* @example $('table').tablesorter();  \* @desc Create a simple tablesorter interface.  \*  \* @example $('table').tablesorter({ sortList:[[0,0],[1,0]] });  \* @desc Create a tablesorter interface and sort on the first and secound column column headers.  \*  \* @example $('table').tablesorter({ headers: { 0: { sorter: false}, 1: {sorter: false} } });  \*  \* @desc Create a tablesorter interface and disableing the first and second column headers.  \*  \*  \* @example $('table').tablesorter({ headers: { 0: {sorter:"integer"}, 1: {sorter:"currency"} } });  \*  \* @desc Create a tablesorter interface and set a column parser for the first  \* and second column.  \*  \*  \* @param Object  \* settings An object literal containing key/value pairs to provide  \* optional settings.  \*  \*  \* @option String cssHeader (optional) A string of the class name to be appended  \* to sortable tr elements in the thead of the table. Default value:  \* "header"  \*  \* @option String cssAsc (optional) A string of the class name to be appended to  \* sortable tr elements in the thead on a ascending sort. Default value:  \* "headerSortUp"  \*  \* @option String cssDesc (optional) A string of the class name to be appended  \* to sortable tr elements in the thead on a descending sort. Default  \* value: "headerSortDown"  \*  \* @option String sortInitialOrder (optional) A string of the inital sorting  \* order can be asc or desc. Default value: "asc"  \*  \* @option String sortMultisortKey (optional) A string of the multi-column sort  \* key. Default value: "shiftKey"  \*  \* @option String textExtraction (optional) A string of the text-extraction  \* method to use. For complex html structures inside td cell set this  \* option to "complex", on large tables the complex option can be slow.  \* Default value: "simple"  \*  \* @option Object headers (optional) An array containing the forces sorting  \* rules. This option let's you specify a default sorting rule. Default  \* value: null  \*  \* @option Array sortList (optional) An array containing the forces sorting  \* rules. This option let's you specify a default sorting rule. Default  \* value: null  \*  \* @option Array sortForce (optional) An array containing forced sorting rules.  \* This option let's you specify a default sorting rule, which is  \* prepended to user-selected rules. Default value: null  \*  \* @option Boolean sortLocaleCompare (optional) Boolean flag indicating whatever  \* to use String.localeCampare method or not. Default set to true.  \*  \*  \* @option Array sortAppend (optional) An array containing forced sorting rules.  \* This option let's you specify a default sorting rule, which is  \* appended to user-selected rules. Default value: null  \*  \* @option Boolean widthFixed (optional) Boolean flag indicating if tablesorter  \* should apply fixed widths to the table columns. This is usefull when  \* using the pager companion plugin. This options requires the dimension  \* jquery plugin. Default value: false  \*  \* @option Boolean cancelSelection (optional) Boolean flag indicating if  \* tablesorter should cancel selection of the table headers text.  \* Default value: true  \*  \* @option Boolean debug (optional) Boolean flag indicating if tablesorter  \* should display debuging information usefull for development.  \*  \* @type jQuery  \*  \* @name tablesorter  \*  \* @cat Plugins/Tablesorter  \*  \* @author Christian Bach/christian.bach@polyester.se  \*/  (function ($) {  $.extend({  tablesorter: new  function () {  var parsers = [],  widgets = [];  this.defaults = {  cssHeader: "header",  cssAsc: "headerSortUp",  cssDesc: "headerSortDown",  cssChildRow: "expand-child",  sortInitialOrder: "asc",  sortMultiSortKey: "shiftKey",  sortForce: null,  sortAppend: null,  sortLocaleCompare: true,  textExtraction: "simple",  parsers: {}, widgets: [],  widgetZebra: {  css: ["even", "odd"]  }, headers: {}, widthFixed: false,  cancelSelection: true,  sortList: [],  headerList: [],  dateFormat: "us",  decimal: '/\.|\,/g',  onRenderHeader: null,  selectorHeaders: 'thead th',  debug: false  };  /\* debuging utils \*/  function benchmark(s, d) {  log(s + "," + (new Date().getTime() - d.getTime()) + "ms");  }  this.benchmark = benchmark;  function log(s) {  if (typeof console != "undefined" && typeof console.debug != "undefined") {  console.log(s);  } else {  alert(s);  }  }  /\* parsers utils \*/  function buildParserCache(table, $headers) {  if (table.config.debug) {  var parsersDebug = "";  }  if (table.tBodies.length == 0) return; // In the case of empty tables  var rows = table.tBodies[0].rows;  if (rows[0]) {  var list = [],  cells = rows[0].cells,  l = cells.length;  for (var i = 0; i < l; i++) {  var p = false;  if ($.metadata && ($($headers[i]).metadata() && $($headers[i]).metadata().sorter)) {  p = getParserById($($headers[i]).metadata().sorter);  } else if ((table.config.headers[i] && table.config.headers[i].sorter)) {  p = getParserById(table.config.headers[i].sorter);  }  if (!p) {  p = detectParserForColumn(table, rows, -1, i);  }  if (table.config.debug) {  parsersDebug += "column:" + i + " parser:" + p.id + "\n";  }  list.push(p);  }  }  if (table.config.debug) {  log(parsersDebug);  }  return list;  };  function detectParserForColumn(table, rows, rowIndex, cellIndex) {  var l = parsers.length,  node = false,  nodeValue = false,  keepLooking = true;  while (nodeValue == '' && keepLooking) {  rowIndex++;  if (rows[rowIndex]) {  node = getNodeFromRowAndCellIndex(rows, rowIndex, cellIndex);  nodeValue = trimAndGetNodeText(table.config, node);  if (table.config.debug) {  log('Checking if value was empty on row:' + rowIndex);  }  } else {  keepLooking = false;  }  }  for (var i = 1; i < l; i++) {  if (parsers[i].is(nodeValue, table, node)) {  return parsers[i];  }  }  // 0 is always the generic parser (text)  return parsers[0];  }  function getNodeFromRowAndCellIndex(rows, rowIndex, cellIndex) {  return rows[rowIndex].cells[cellIndex];  }  function trimAndGetNodeText(config, node) {  return $.trim(getElementText(config, node));  }  function getParserById(name) {  var l = parsers.length;  for (var i = 0; i < l; i++) {  if (parsers[i].id.toLowerCase() == name.toLowerCase()) {  return parsers[i];  }  }  return false;  }  /\* utils \*/  function buildCache(table) {  if (table.config.debug) {  var cacheTime = new Date();  }  var totalRows = (table.tBodies[0] && table.tBodies[0].rows.length) || 0,  totalCells = (table.tBodies[0].rows[0] && table.tBodies[0].rows[0].cells.length) || 0,  parsers = table.config.parsers,  cache = {  row: [],  normalized: []  };  for (var i = 0; i < totalRows; ++i) {  /\*\* Add the table data to main data array \*/  var c = $(table.tBodies[0].rows[i]),  cols = [];  // if this is a child row, add it to the last row's children and  // continue to the next row  if (c.hasClass(table.config.cssChildRow)) {  cache.row[cache.row.length - 1] = cache.row[cache.row.length - 1].add(c);  // go to the next for loop  continue;  }  cache.row.push(c);  for (var j = 0; j < totalCells; ++j) {  cols.push(parsers[j].format(getElementText(table.config, c[0].cells[j]), table, c[0].cells[j]));  }  cols.push(cache.normalized.length); // add position for rowCache  cache.normalized.push(cols);  cols = null;  };  if (table.config.debug) {  benchmark("Building cache for " + totalRows + " rows:", cacheTime);  }  return cache;  };  function getElementText(config, node) {  var text = "";  if (!node) return "";  if (!config.supportsTextContent) config.supportsTextContent = node.textContent || false;  if (config.textExtraction == "simple") {  if (config.supportsTextContent) {  text = node.textContent;  } else {  if (node.childNodes[0] && node.childNodes[0].hasChildNodes()) {  text = node.childNodes[0].innerHTML;  } else {  text = node.innerHTML;  }  }  } else {  if (typeof(config.textExtraction) == "function") {  text = config.textExtraction(node);  } else {  text = $(node).text();  }  }  return text;  }  function appendToTable(table, cache) {  if (table.config.debug) {  var appendTime = new Date()  }  var c = cache,  r = c.row,  n = c.normalized,  totalRows = n.length,  checkCell = (n[0].length - 1),  tableBody = $(table.tBodies[0]),  rows = [];  for (var i = 0; i < totalRows; i++) {  var pos = n[i][checkCell];  rows.push(r[pos]);  if (!table.config.appender) {  //var o = ;  var l = r[pos].length;  for (var j = 0; j < l; j++) {  tableBody[0].appendChild(r[pos][j]);  }  //  }  }  if (table.config.appender) {  table.config.appender(table, rows);  }  rows = null;  if (table.config.debug) {  benchmark("Rebuilt table:", appendTime);  }  // apply table widgets  applyWidget(table);  // trigger sortend  setTimeout(function () {  $(table).trigger("sortEnd");  }, 0);  };  function buildHeaders(table) {  if (table.config.debug) {  var time = new Date();  }  var meta = ($.metadata) ? true : false;    var header\_index = computeTableHeaderCellIndexes(table);  $tableHeaders = $(table.config.selectorHeaders, table).each(function (index) {  this.column = header\_index[this.parentNode.rowIndex + "-" + this.cellIndex];  // this.column = index;  this.order = formatSortingOrder(table.config.sortInitialOrder);      this.count = this.order;  if (checkHeaderMetadata(this) || checkHeaderOptions(table, index)) this.sortDisabled = true;  if (checkHeaderOptionsSortingLocked(table, index)) this.order = this.lockedOrder = checkHeaderOptionsSortingLocked(table, index);  if (!this.sortDisabled) {  var $th = $(this).addClass(table.config.cssHeader);  if (table.config.onRenderHeader) table.config.onRenderHeader.apply($th);  }  // add cell to headerList  table.config.headerList[index] = this;  });  if (table.config.debug) {  benchmark("Built headers:", time);  log($tableHeaders);  }  return $tableHeaders;  };  // from:  // http://www.javascripttoolbox.com/lib/table/examples.php  // http://www.javascripttoolbox.com/temp/table\_cellindex.html  function computeTableHeaderCellIndexes(t) {  var matrix = [];  var lookup = {};  var thead = t.getElementsByTagName('THEAD')[0];  var trs = thead.getElementsByTagName('TR');  for (var i = 0; i < trs.length; i++) {  var cells = trs[i].cells;  for (var j = 0; j < cells.length; j++) {  var c = cells[j];  var rowIndex = c.parentNode.rowIndex;  var cellId = rowIndex + "-" + c.cellIndex;  var rowSpan = c.rowSpan || 1;  var colSpan = c.colSpan || 1  var firstAvailCol;  if (typeof(matrix[rowIndex]) == "undefined") {  matrix[rowIndex] = [];  }  // Find first available column in the first row  for (var k = 0; k < matrix[rowIndex].length + 1; k++) {  if (typeof(matrix[rowIndex][k]) == "undefined") {  firstAvailCol = k;  break;  }  }  lookup[cellId] = firstAvailCol;  for (var k = rowIndex; k < rowIndex + rowSpan; k++) {  if (typeof(matrix[k]) == "undefined") {  matrix[k] = [];  }  var matrixrow = matrix[k];  for (var l = firstAvailCol; l < firstAvailCol + colSpan; l++) {  matrixrow[l] = "x";  }  }  }  }  return lookup;  }  function checkCellColSpan(table, rows, row) {  var arr = [],  r = table.tHead.rows,  c = r[row].cells;  for (var i = 0; i < c.length; i++) {  var cell = c[i];  if (cell.colSpan > 1) {  arr = arr.concat(checkCellColSpan(table, headerArr, row++));  } else {  if (table.tHead.length == 1 || (cell.rowSpan > 1 || !r[row + 1])) {  arr.push(cell);  }  // headerArr[row] = (i+row);  }  }  return arr;  };  function checkHeaderMetadata(cell) {  if (($.metadata) && ($(cell).metadata().sorter === false)) {  return true;  };  return false;  }  function checkHeaderOptions(table, i) {  if ((table.config.headers[i]) && (table.config.headers[i].sorter === false)) {  return true;  };  return false;  }    function checkHeaderOptionsSortingLocked(table, i) {  if ((table.config.headers[i]) && (table.config.headers[i].lockedOrder)) return table.config.headers[i].lockedOrder;  return false;  }    function applyWidget(table) {  var c = table.config.widgets;  var l = c.length;  for (var i = 0; i < l; i++) {  getWidgetById(c[i]).format(table);  }  }  function getWidgetById(name) {  var l = widgets.length;  for (var i = 0; i < l; i++) {  if (widgets[i].id.toLowerCase() == name.toLowerCase()) {  return widgets[i];  }  }  };  function formatSortingOrder(v) {  if (typeof(v) != "Number") {  return (v.toLowerCase() == "desc") ? 1 : 0;  } else {  return (v == 1) ? 1 : 0;  }  }  function isValueInArray(v, a) {  var l = a.length;  for (var i = 0; i < l; i++) {  if (a[i][0] == v) {  return true;  }  }  return false;  }  function setHeadersCss(table, $headers, list, css) {  // remove all header information  $headers.removeClass(css[0]).removeClass(css[1]);  var h = [];  $headers.each(function (offset) {  if (!this.sortDisabled) {  h[this.column] = $(this);  }  });  var l = list.length;  for (var i = 0; i < l; i++) {  h[list[i][0]].addClass(css[list[i][1]]);  }  }  function fixColumnWidth(table, $headers) {  var c = table.config;  if (c.widthFixed) {  var colgroup = $('<colgroup>');  $("tr:first td", table.tBodies[0]).each(function () {  colgroup.append($('<col>').css('width', $(this).width()));  });  $(table).prepend(colgroup);  };  }  function updateHeaderSortCount(table, sortList) {  var c = table.config,  l = sortList.length;  for (var i = 0; i < l; i++) {  var s = sortList[i],  o = c.headerList[s[0]];  o.count = s[1];  o.count++;  }  }  /\* sorting methods \*/  function multisort(table, sortList, cache) {  if (table.config.debug) {  var sortTime = new Date();  }  var dynamicExp = "var sortWrapper = function(a,b) {",  l = sortList.length;  // TODO: inline functions.  for (var i = 0; i < l; i++) {  var c = sortList[i][0];  var order = sortList[i][1];  // var s = (getCachedSortType(table.config.parsers,c) == "text") ?  // ((order == 0) ? "sortText" : "sortTextDesc") : ((order == 0) ?  // "sortNumeric" : "sortNumericDesc");  // var s = (table.config.parsers[c].type == "text") ? ((order == 0)  // ? makeSortText(c) : makeSortTextDesc(c)) : ((order == 0) ?  // makeSortNumeric(c) : makeSortNumericDesc(c));  var s = (table.config.parsers[c].type == "text") ? ((order == 0) ? makeSortFunction("text", "asc", c) : makeSortFunction("text", "desc", c)) : ((order == 0) ? makeSortFunction("numeric", "asc", c) : makeSortFunction("numeric", "desc", c));  var e = "e" + i;  dynamicExp += "var " + e + " = " + s; // + "(a[" + c + "],b[" + c  // + "]); ";  dynamicExp += "if(" + e + ") { return " + e + "; } ";  dynamicExp += "else { ";  }  // if value is the same keep orignal order  var orgOrderCol = cache.normalized[0].length - 1;  dynamicExp += "return a[" + orgOrderCol + "]-b[" + orgOrderCol + "];";  for (var i = 0; i < l; i++) {  dynamicExp += "}; ";  }  dynamicExp += "return 0; ";  dynamicExp += "}; ";  if (table.config.debug) {  benchmark("Evaling expression:" + dynamicExp, new Date());  }  eval(dynamicExp);  cache.normalized.sort(sortWrapper);  if (table.config.debug) {  benchmark("Sorting on " + sortList.toString() + " and dir " + order + " time:", sortTime);  }  return cache;  };  function makeSortFunction(type, direction, index) {  var a = "a[" + index + "]",  b = "b[" + index + "]";  if (type == 'text' && direction == 'asc') {  return "(" + a + " == " + b + " ? 0 : (" + a + " === null ? Number.POSITIVE\_INFINITY : (" + b + " === null ? Number.NEGATIVE\_INFINITY : (" + a + " < " + b + ") ? -1 : 1 )));";  } else if (type == 'text' && direction == 'desc') {  return "(" + a + " == " + b + " ? 0 : (" + a + " === null ? Number.POSITIVE\_INFINITY : (" + b + " === null ? Number.NEGATIVE\_INFINITY : (" + b + " < " + a + ") ? -1 : 1 )));";  } else if (type == 'numeric' && direction == 'asc') {  return "(" + a + " === null && " + b + " === null) ? 0 :(" + a + " === null ? Number.POSITIVE\_INFINITY : (" + b + " === null ? Number.NEGATIVE\_INFINITY : " + a + " - " + b + "));";  } else if (type == 'numeric' && direction == 'desc') {  return "(" + a + " === null && " + b + " === null) ? 0 :(" + a + " === null ? Number.POSITIVE\_INFINITY : (" + b + " === null ? Number.NEGATIVE\_INFINITY : " + b + " - " + a + "));";  }  };  function makeSortText(i) {  return "((a[" + i + "] < b[" + i + "]) ? -1 : ((a[" + i + "] > b[" + i + "]) ? 1 : 0));";  };  function makeSortTextDesc(i) {  return "((b[" + i + "] < a[" + i + "]) ? -1 : ((b[" + i + "] > a[" + i + "]) ? 1 : 0));";  };  function makeSortNumeric(i) {  return "a[" + i + "]-b[" + i + "];";  };  function makeSortNumericDesc(i) {  return "b[" + i + "]-a[" + i + "];";  };  function sortText(a, b) {  if (table.config.sortLocaleCompare) return a.localeCompare(b);  return ((a < b) ? -1 : ((a > b) ? 1 : 0));  };  function sortTextDesc(a, b) {  if (table.config.sortLocaleCompare) return b.localeCompare(a);  return ((b < a) ? -1 : ((b > a) ? 1 : 0));  };  function sortNumeric(a, b) {  return a - b;  };  function sortNumericDesc(a, b) {  return b - a;  };  function getCachedSortType(parsers, i) {  return parsers[i].type;  }; /\* public methods \*/  this.construct = function (settings) {  return this.each(function () {  // if no thead or tbody quit.  if (!this.tHead || !this.tBodies) return;  // declare  var $this, $document, $headers, cache, config, shiftDown = 0,  sortOrder;  // new blank config object  this.config = {};  // merge and extend.  config = $.extend(this.config, $.tablesorter.defaults, settings);  // store common expression for speed  $this = $(this);  // save the settings where they read  $.data(this, "tablesorter", config);  // build headers  $headers = buildHeaders(this);  // try to auto detect column type, and store in tables config  this.config.parsers = buildParserCache(this, $headers);  // build the cache for the tbody cells  cache = buildCache(this);  // get the css class names, could be done else where.  var sortCSS = [config.cssDesc, config.cssAsc];  // fixate columns if the users supplies the fixedWidth option  fixColumnWidth(this);  // apply event handling to headers  // this is to big, perhaps break it out?  $headers.click(  function (e) {  var totalRows = ($this[0].tBodies[0] && $this[0].tBodies[0].rows.length) || 0;  if (!this.sortDisabled && totalRows > 0) {  // Only call sortStart if sorting is  // enabled.  $this.trigger("sortStart");  // store exp, for speed  var $cell = $(this);  // get current column index  var i = this.column;  // get current column sort order  this.order = this.count++ % 2;  // always sort on the locked order.  if(this.lockedOrder) this.order = this.lockedOrder;    // user only whants to sort on one  // column  if (!e[config.sortMultiSortKey]) {  // flush the sort list  config.sortList = [];  if (config.sortForce != null) {  var a = config.sortForce;  for (var j = 0; j < a.length; j++) {  if (a[j][0] != i) {  config.sortList.push(a[j]);  }  }  }  // add column to sort list  config.sortList.push([i, this.order]);  // multi column sorting  } else {  // the user has clicked on an all  // ready sortet column.  if (isValueInArray(i, config.sortList)) {  // revers the sorting direction  // for all tables.  for (var j = 0; j < config.sortList.length; j++) {  var s = config.sortList[j],  o = config.headerList[s[0]];  if (s[0] == i) {  o.count = s[1];  o.count++;  s[1] = o.count % 2;  }  }  } else {  // add column to sort list array  config.sortList.push([i, this.order]);  }  };  setTimeout(function () {  // set css for headers  setHeadersCss($this[0], $headers, config.sortList, sortCSS);  appendToTable(  $this[0], multisort(  $this[0], config.sortList, cache)  );  }, 1);  // stop normal event by returning false  return false;  }  // cancel selection  }).mousedown(function () {  if (config.cancelSelection) {  this.onselectstart = function () {  return false  };  return false;  }  });  // apply easy methods that trigger binded events  $this.bind("update", function () {  var me = this;  setTimeout(function () {  // rebuild parsers.  me.config.parsers = buildParserCache(  me, $headers);  // rebuild the cache map  cache = buildCache(me);  }, 1);  }).bind("updateCell", function (e, cell) {  var config = this.config;  // get position from the dom.  var pos = [(cell.parentNode.rowIndex - 1), cell.cellIndex];  // update cache  cache.normalized[pos[0]][pos[1]] = config.parsers[pos[1]].format(  getElementText(config, cell), cell);  }).bind("sorton", function (e, list) {  $(this).trigger("sortStart");  config.sortList = list;  // update and store the sortlist  var sortList = config.sortList;  // update header count index  updateHeaderSortCount(this, sortList);  // set css for headers  setHeadersCss(this, $headers, sortList, sortCSS);  // sort the table and append it to the dom  appendToTable(this, multisort(this, sortList, cache));  }).bind("appendCache", function () {  appendToTable(this, cache);  }).bind("applyWidgetId", function (e, id) {  getWidgetById(id).format(this);  }).bind("applyWidgets", function () {  // apply widgets  applyWidget(this);  });  if ($.metadata && ($(this).metadata() && $(this).metadata().sortlist)) {  config.sortList = $(this).metadata().sortlist;  }  // if user has supplied a sort list to constructor.  if (config.sortList.length > 0) {  $this.trigger("sorton", [config.sortList]);  }  // apply widgets  applyWidget(this);  });  };  this.addParser = function (parser) {  var l = parsers.length,  a = true;  for (var i = 0; i < l; i++) {  if (parsers[i].id.toLowerCase() == parser.id.toLowerCase()) {  a = false;  }  }  if (a) {  parsers.push(parser);  };  };  this.addWidget = function (widget) {  widgets.push(widget);  };  this.formatFloat = function (s) {  var i = parseFloat(s);  return (isNaN(i)) ? 0 : i;  };  this.formatInt = function (s) {  var i = parseInt(s);  return (isNaN(i)) ? 0 : i;  };  this.isDigit = function (s, config) {  // replace all an wanted chars and match.  return /^[-+]?\d\*$/.test($.trim(s.replace(/[,.']/g, '')));  };  this.clearTableBody = function (table) {  if ($.browser.msie) {  function empty() {  while (this.firstChild)  this.removeChild(this.firstChild);  }  empty.apply(table.tBodies[0]);  } else {  table.tBodies[0].innerHTML = "";  }  };  }  });  // extend plugin scope  $.fn.extend({  tablesorter: $.tablesorter.construct  });  // make shortcut  var ts = $.tablesorter;  // add default parsers  ts.addParser({  id: "text",  is: function (s) {  return true;  }, format: function (s) {  return $.trim(s.toLocaleLowerCase());  }, type: "text"  });  ts.addParser({  id: "digit",  is: function (s, table) {  var c = table.config;  return $.tablesorter.isDigit(s, c);  }, format: function (s) {  return $.tablesorter.formatFloat(s);  }, type: "numeric"  });  ts.addParser({  id: "currency",  is: function (s) {  return /^[£$€?.]/.test(s);  }, format: function (s) {  return $.tablesorter.formatFloat(s.replace(new RegExp(/[£$€]/g), ""));  }, type: "numeric"  });  ts.addParser({  id: "ipAddress",  is: function (s) {  return /^\d{2,3}[\.]\d{2,3}[\.]\d{2,3}[\.]\d{2,3}$/.test(s);  }, format: function (s) {  var a = s.split("."),  r = "",  l = a.length;  for (var i = 0; i < l; i++) {  var item = a[i];  if (item.length == 2) {  r += "0" + item;  } else {  r += item;  }  }  return $.tablesorter.formatFloat(r);  }, type: "numeric"  });  ts.addParser({  id: "url",  is: function (s) {  return /^(https?|ftp|file):\/\/$/.test(s);  }, format: function (s) {  return jQuery.trim(s.replace(new RegExp(/(https?|ftp|file):\/\//), ''));  }, type: "text"  });  ts.addParser({  id: "isoDate",  is: function (s) {  return /^\d{4}[\/-]\d{1,2}[\/-]\d{1,2}$/.test(s);  }, format: function (s) {  return $.tablesorter.formatFloat((s != "") ? new Date(s.replace(  new RegExp(/-/g), "/")).getTime() : "0");  }, type: "numeric"  });  ts.addParser({  id: "percent",  is: function (s) {  return /\%$/.test($.trim(s));  }, format: function (s) {  return $.tablesorter.formatFloat(s.replace(new RegExp(/%/g), ""));  }, type: "numeric"  });  ts.addParser({  id: "usLongDate",  is: function (s) {  return s.match(new RegExp(/^[A-Za-z]{3,10}\.? [0-9]{1,2}, ([0-9]{4}|'?[0-9]{2}) (([0-2]?[0-9]:[0-5][0-9])|([0-1]?[0-9]:[0-5][0-9]\s(AM|PM)))$/));  }, format: function (s) {  return $.tablesorter.formatFloat(new Date(s).getTime());  }, type: "numeric"  });  ts.addParser({  id: "shortDate",  is: function (s) {  return /\d{1,2}[\/\-]\d{1,2}[\/\-]\d{2,4}/.test(s);  }, format: function (s, table) {  var c = table.config;  s = s.replace(/\-/g, "/");  if (c.dateFormat == "us") {  // reformat the string in ISO format  s = s.replace(/(\d{1,2})[\/\-](\d{1,2})[\/\-](\d{4})/, "$3/$1/$2");  } else if (c.dateFormat == "uk") {  // reformat the string in ISO format  s = s.replace(/(\d{1,2})[\/\-](\d{1,2})[\/\-](\d{4})/, "$3/$2/$1");  } else if (c.dateFormat == "dd/mm/yy" || c.dateFormat == "dd-mm-yy") {  s = s.replace(/(\d{1,2})[\/\-](\d{1,2})[\/\-](\d{2})/, "$1/$2/$3");  }  return $.tablesorter.formatFloat(new Date(s).getTime());  }, type: "numeric"  });  ts.addParser({  id: "time",  is: function (s) {  return /^(([0-2]?[0-9]:[0-5][0-9])|([0-1]?[0-9]:[0-5][0-9]\s(am|pm)))$/.test(s);  }, format: function (s) {  return $.tablesorter.formatFloat(new Date("2000/01/01 " + s).getTime());  }, type: "numeric"  });  ts.addParser({  id: "metadata",  is: function (s) {  return false;  }, format: function (s, table, cell) {  var c = table.config,  p = (!c.parserMetadataName) ? 'sortValue' : c.parserMetadataName;  return $(cell).metadata()[p];  }, type: "numeric"  });  // add default widgets  ts.addWidget({  id: "zebra",  format: function (table) {  if (table.config.debug) {  var time = new Date();  }  var $tr, row = -1,  odd;  // loop through the visible rows  $("tr:visible", table.tBodies[0]).each(function (i) {  $tr = $(this);  // style children rows the same way the parent  // row was styled  if (!$tr.hasClass(table.config.cssChildRow)) row++;  odd = (row % 2 == 0);  $tr.removeClass(  table.config.widgetZebra.css[odd ? 0 : 1]).addClass(  table.config.widgetZebra.css[odd ? 1 : 0])  });  if (table.config.debug) {  $.tablesorter.benchmark("Applying Zebra widget", time);  }  }  });  })(jQuery); |

## **9.3 Android (Java language)**

|  |
| --- |
| **build.gradle** |
| apply plugin: 'com.android.application'  android {  compileSdkVersion 24  buildToolsVersion "24.0.2"  defaultConfig {  applicationId "app.smartdoorlock.com.smartdoorlockandroidapp"  minSdkVersion 19  targetSdkVersion 19  versionCode 1  versionName "1.0"  }  buildTypes {  release {  minifyEnabled false  proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  }  }  }  dependencies {  compile fileTree(dir: 'libs', include: ['\*.jar'])  compile 'com.android.support:appcompat-v7:24.2.1'  compile 'com.android.support:design:24.2.1'  compile 'com.android.support:support-v4:24.2.1'  testCompile 'junit:junit:4.12'  } |

|  |
| --- |
| **MainActivity.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp;  import android.os.Bundle;  import android.support.v4.app.Fragment;  import android.support.v4.app.FragmentManager;  import android.support.design.widget.NavigationView;  import android.support.v4.view.GravityCompat;  import android.support.v4.widget.DrawerLayout;  import android.support.v7.app.ActionBarDrawerToggle;  import android.support.v7.app.AppCompatActivity;  import android.support.v7.widget.Toolbar;  import android.view.Menu;  import android.view.MenuItem;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper;  public class MainActivity extends AppCompatActivity  implements NavigationView.OnNavigationItemSelectedListener , IFragmentInteractionListener {  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  setSupportActionBar(toolbar);  DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer\_layout);  ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(  this, drawer, toolbar, R.string.navigation\_drawer\_open, R.string.navigation\_drawer\_close);  drawer.setDrawerListener(toggle);  toggle.syncState();  NavigationView navigationView = (NavigationView) findViewById(R.id.nav\_view);  navigationView.setNavigationItemSelectedListener(this);  SPHelper.putCommand(MainActivity.this,SPHelper.SAVED\_COMMAND,CommandEnum.NONE);  SPHelper.putCommand(MainActivity.this,SPHelper.CURRENT\_COMMAND,CommandEnum.NONE);  }  @Override  public void onBackPressed() {  DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer\_layout);  if (drawer.isDrawerOpen(GravityCompat.START)) {  drawer.closeDrawer(GravityCompat.START);  } else {  super.onBackPressed();  }  }  @Override  public boolean onCreateOptionsMenu(Menu menu) {  // Inflate the menu; this adds items to the action bar if it is present.  getMenuInflater().inflate(R.menu.main, menu);  return true;  }  @Override  public boolean onOptionsItemSelected(MenuItem item) {  // Handle action bar item clicks here. The action bar will  // automatically handle clicks on the Home/Up button, so long  // as you specify a parent activity in AndroidManifest.xml.  int id = item.getItemId();  //noinspection SimplifiableIfStatement  if (id == R.id.action\_settings) {  return true;  }  return super.onOptionsItemSelected(item);  }  @Override  public void onPause() {  CommandEnum val = SPHelper.getCommand(MainActivity.this,SPHelper.CURRENT\_COMMAND);  SPHelper.putCommand(MainActivity.this,SPHelper.SAVED\_COMMAND,val);  SPHelper.putCommand(MainActivity.this,SPHelper.CURRENT\_COMMAND,CommandEnum.NONE);  super.onPause();  }  @Override  public void onResume() {  CommandEnum val = SPHelper.getCommand(MainActivity.this,SPHelper.SAVED\_COMMAND);  SPHelper.putCommand(MainActivity.this,SPHelper.CURRENT\_COMMAND,val);  super.onResume();  }  @SuppressWarnings("StatementWithEmptyBody")  @Override  public boolean onNavigationItemSelected(MenuItem menuItem) {  // Create a new fragment and specify the fragment to show based on nav item clicked  Fragment fragment = null;  Class fragmentClass = null;  switch(menuItem.getItemId()) {  case R.id.nav\_nfc\_doorlock\_control:  fragmentClass = NFCDoorlockControlFragment.class;  break;  case R.id.nav\_nfc\_doorlock\_registration:  fragmentClass = NFCDoorlockRegistrationFragment.class;  break;  case R.id.nav\_nfc\_wifi\_config:  fragmentClass = NFCWifiSetupFragment.class;  break;  case R.id.nav\_config:  break;  default:  return true;  }  try {  fragment = (Fragment) fragmentClass.newInstance();  } catch (Exception e) {  e.printStackTrace();  }  // Insert the fragment by replacing any existing fragment  FragmentManager fragmentManager = getSupportFragmentManager();  fragmentManager.beginTransaction().replace(R.id.fl\_frag\_content, fragment).commit();  // Highlight the selected item has been done by NavigationView  menuItem.setChecked(true);  // Set action bar title  setTitle(menuItem.getTitle());  DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer\_layout);  drawer.closeDrawer(GravityCompat.START);  return true;  }  @Override  public void onFragmentInteraction(String string) {  }  } |

|  |
| --- |
| **NFCDoorlockControlFragment.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp;  import android.content.Context;  import android.net.Uri;  import android.nfc.NdefMessage;  import android.nfc.NdefRecord;  import android.nfc.NfcAdapter;  import android.nfc.NfcEvent;  import android.os.Bundle;  import android.support.v4.app.Fragment;  import android.text.TextUtils;  import android.view.LayoutInflater;  import android.view.View;  import android.view.ViewGroup;  import android.widget.TextView;  import android.widget.Toast;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper;  import static android.nfc.NdefRecord.createMime;  public class NFCDoorlockControlFragment extends Fragment {  private TextView tvInfo;  public NFCDoorlockControlFragment() {  // Required empty public constructor  }  public static NFCDoorlockControlFragment newInstance() {  NFCDoorlockControlFragment fragment = new NFCDoorlockControlFragment();  return fragment;  }  @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  }  @Override  public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {  // Inflate the layout for this fragment  View v = inflater.inflate(R.layout.fragment\_nfc\_doorlock\_control, container, false);  tvInfo = (TextView) v.findViewById(R.id.fragment\_nfc\_doorlock\_ctl\_tv\_info);  String phoneId = SPHelper.getString(getActivity(),SPHelper.KEY\_PHONE\_ID);  if (TextUtils.isEmpty(phoneId)) {  tvInfo.setText("Phone ID could not be found.\n\nPlease register phone to Smart Doorlock first");  SPHelper.putCommand(getActivity(),SPHelper.CURRENT\_COMMAND, CommandEnum.NONE);  }  else {  SPHelper.putCommand(getActivity(),SPHelper.CURRENT\_COMMAND, CommandEnum.DOORLOCK\_CONTROL);  }  return v;  }  @Override  public void onAttach(Context context) {  super.onAttach(context);  }  @Override  public void onDetach() {  super.onDetach();  }  } |

|  |
| --- |
| **NFCDoorlockHCE.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp;  import android.content.Intent;  import android.nfc.cardemulation.HostApduService;  import android.os.Bundle;  import android.text.TextUtils;  import android.util.Log;  import android.widget.Toast;  import java.text.SimpleDateFormat;  import java.util.Arrays;  import java.util.Date;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper;  import static app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum.DOORLOCK\_CONTROL;  import static app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum.DOORLOCK\_REGISTRATION;  import static app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum.WIFI\_SETUP;  import static app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper.KEY\_REGISTRATION\_DATE;  /\*\*  \* Created by shuh on 10/22/2016.  \*/  public class NFCDoorlockHCE extends HostApduService {  private static final String TAG = "CardService";  // AID for smart doorlock  private static final String SMART\_DOORLOCK\_AID = "D2760000850101";  // ISO-DEP command HEADER for selecting an AID.  // Format: [Class | Instruction | Parameter 1 | Parameter 2]  private static final String SELECT\_APDU\_HEADER = "00A40400";  // "OK" status word sent in response to SELECT AID command (0x9000)  private static final byte[] SELECT\_OK\_SW = HexStringToByteArray("9000");  // "UNKNOWN" status word sent in response to invalid APDU command (0x0000)  private static final byte[] UNKNOWN\_CMD\_SW = HexStringToByteArray("0000");  private static final byte[] SELECT\_APDU = BuildSelectApdu(SMART\_DOORLOCK\_AID);  public static final String BR\_REFRESH\_FILTER = "NFC\_REG\_REFRESH";  /\*\*  \* Called if the connection to the NFC card is lost, in order to let the application know the  \* cause for the disconnection (either a lost link, or another AID being selected by the  \* reader).  \*  \* @param reason Either DEACTIVATION\_LINK\_LOSS or DEACTIVATION\_DESELECTED  \*/  @Override  public void onDeactivated(int reason) { }  /\*\*  \* Processes APDU commands from TRF7970A  \*  \* Handles SELECT AID command.  \*  \* Response is appended to R\_APDU OK status (90 00) + Payload  \*/  // BEGIN\_INCLUDE(processCommandApdu)  @Override  public byte[] processCommandApdu(byte[] commandApdu, Bundle extras) {  Log.i(TAG, "Received APDU: " + ByteArrayToHexString(commandApdu));  if (!Arrays.equals(SELECT\_APDU, commandApdu)) {  return UNKNOWN\_CMD\_SW;  }  CommandEnum enumVal = SPHelper.getCommand(NFCDoorlockHCE.this,SPHelper.CURRENT\_COMMAND);  // If the APDU matches the SELECT AID command for this service,  // send the loyalty card account number, followed by a SELECT\_OK status trailer (0x9000).  String payload;  switch (enumVal) {  case DOORLOCK\_CONTROL:  String phoneId = SPHelper.getString(NFCDoorlockHCE.this,SPHelper.KEY\_PHONE\_ID);  payload = DOORLOCK\_CONTROL.toString() + "|" + phoneId;  return getAckPayload(payload);  case DOORLOCK\_REGISTRATION:  String newId = SPHelper.getString(NFCDoorlockHCE.this,SPHelper.KEY\_PHONE\_ID);  if (!TextUtils.isEmpty(newId)) {  SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");  payload = DOORLOCK\_REGISTRATION.toString() + "|" + newId + "|" + sdf.format(new Date());  Toast.makeText(NFCDoorlockHCE.this,"Successfully registered phone",Toast.LENGTH\_LONG).show();  sdf = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");  SPHelper.putString(NFCDoorlockHCE.this,KEY\_REGISTRATION\_DATE,sdf.format(new Date()));  //Broadcast message to refresh fragment  Intent intent = new Intent();  intent.setAction(BR\_REFRESH\_FILTER);  intent.putExtra(BR\_REFRESH\_FILTER, 1);  sendBroadcast(intent);  return getAckPayload(payload);  }  break;  case WIFI\_SETUP:  String ssid = SPHelper.getString(NFCDoorlockHCE.this,SPHelper.KEY\_WIFI\_SSID);  String password = SPHelper.getString(NFCDoorlockHCE.this,SPHelper.KEY\_WIFI\_PASSWORD);  String encryption = SPHelper.getString(NFCDoorlockHCE.this,SPHelper.KEY\_WIFI\_ENCRYPTION);  payload = WIFI\_SETUP.toString() + "|" + ssid + "|" + password + "|" + encryption;  return getAckPayload(payload);  default:  break;  }  return null;  }  private byte[] getAckPayload(String content) {  byte[] payload = content.getBytes();  Log.i(TAG, "NFC Responding with: " + content);  Log.i(TAG, "Byte response: " + ByteArrayToHexString(payload));  return ConcatArrays(SELECT\_OK\_SW, payload);  }  // END\_INCLUDE(processCommandApdu)  /\*\*  \* Build APDU for SELECT AID command. This command indicates which service a reader is  \* interested in communicating with. See ISO 7816-4.  \*  \* @param aid Application ID (AID) to select  \* @return APDU for SELECT AID command  \*/  public static byte[] BuildSelectApdu(String aid) {  // Format: [CLASS | INSTRUCTION | PARAMETER 1 | PARAMETER 2 | LENGTH | DATA | LE]  return HexStringToByteArray(SELECT\_APDU\_HEADER + String.format("%02X",  aid.length() / 2) + aid + "00");  }  /\*\*  \* Utility method to convert a byte array to a hexadecimal string.  \*  \* @param bytes Bytes to convert  \* @return String, containing hexadecimal representation.  \*/  public static String ByteArrayToHexString(byte[] bytes) {  final char[] hexArray = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};  char[] hexChars = new char[bytes.length \* 2]; // Each byte has two hex characters (nibbles)  int v;  for (int j = 0; j < bytes.length; j++) {  v = bytes[j] & 0xFF; // Cast bytes[j] to int, treating as unsigned value  hexChars[j \* 2] = hexArray[v >>> 4]; // Select hex character from upper nibble  hexChars[j \* 2 + 1] = hexArray[v & 0x0F]; // Select hex character from lower nibble  }  return new String(hexChars);  }  /\*\*  \* Utility method to convert a hexadecimal string to a byte string.  \*  \* <p>Behavior with input strings containing non-hexadecimal characters is undefined.  \*  \* @param s String containing hexadecimal characters to convert  \* @return Byte array generated from input  \* @throws java.lang.IllegalArgumentException if input length is incorrect  \*/  public static byte[] HexStringToByteArray(String s) throws IllegalArgumentException {  int len = s.length();  if (len % 2 == 1) {  throw new IllegalArgumentException("Hex string must have even number of characters");  }  byte[] data = new byte[len / 2]; // Allocate 1 byte per 2 hex characters  for (int i = 0; i < len; i += 2) {  // Convert each character into a integer (base-16), then bit-shift into place  data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4)  + Character.digit(s.charAt(i+1), 16));  }  return data;  }  /\*\*  \* Utility method to concatenate two byte arrays.  \* @param first First array  \* @param rest Any remaining arrays  \* @return Concatenated copy of input arrays  \*/  public static byte[] ConcatArrays(byte[] first, byte[]... rest) {  int totalLength = first.length;  for (byte[] array : rest) {  totalLength += array.length;  }  byte[] result = Arrays.copyOf(first, totalLength);  int offset = first.length;  for (byte[] array : rest) {  System.arraycopy(array, 0, result, offset, array.length);  offset += array.length;  }  return result;  }  } |

|  |
| --- |
| **NFCDoorlockRegistrationFragment.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp;  import android.content.BroadcastReceiver;  import android.content.Context;  import android.content.Intent;  import android.content.IntentFilter;  import android.net.Uri;  import android.os.Bundle;  import android.support.v4.app.Fragment;  import android.text.TextUtils;  import android.view.LayoutInflater;  import android.view.View;  import android.view.ViewGroup;  import android.widget.Button;  import android.widget.TextView;  import java.math.BigInteger;  import java.security.SecureRandom;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper;  public class NFCDoorlockRegistrationFragment extends Fragment {  private TextView tvPhoneId;  private TextView tvRegDate;  private Button btnGenId;  private SecureRandom random = new SecureRandom();  private RegisterReceiver regEventReceiver;  public NFCDoorlockRegistrationFragment() {  // Required empty public constructor  }  public static NFCDoorlockRegistrationFragment newInstance() {  NFCDoorlockRegistrationFragment fragment = new NFCDoorlockRegistrationFragment();  Bundle args = new Bundle();  fragment.setArguments(args);  return fragment;  }  @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  }  @Override  public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {  View v = inflater.inflate(R.layout.fragment\_nfc\_doorlock\_registration, container, false);  tvPhoneId = (TextView) v.findViewById(R.id.fragment\_nfc\_doorlock\_reg\_tv\_smartphone\_id);  tvRegDate = (TextView) v.findViewById(R.id.fragment\_nfc\_doorlock\_reg\_tv\_reg\_date);  SPHelper.putCommand(getActivity(),SPHelper.CURRENT\_COMMAND, CommandEnum.DOORLOCK\_REGISTRATION);  String phoneId = SPHelper.getString(getActivity(),SPHelper.KEY\_PHONE\_ID);  if (!TextUtils.isEmpty(phoneId)) {  tvPhoneId.setText(phoneId);  }  tvRegDate.setText(SPHelper.getString(getActivity(),SPHelper.KEY\_REGISTRATION\_DATE));  btnGenId = (Button) v.findViewById(R.id.fragment\_nfc\_doorlock\_btn\_gen\_id);  btnGenId.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  String randId = new BigInteger(130, random).toString(32);  tvPhoneId.setText(randId);  SPHelper.putString(getActivity(),SPHelper.KEY\_PHONE\_ID,randId);  }  });  return v;  }  @Override  public void onAttach(Context context) {  IntentFilter intentFilter = new IntentFilter();  intentFilter.addAction(NFCDoorlockHCE.BR\_REFRESH\_FILTER);  regEventReceiver = new RegisterReceiver();  getActivity().registerReceiver(regEventReceiver, intentFilter);  super.onAttach(context);  }  @Override  public void onDetach() {  getActivity().unregisterReceiver(regEventReceiver);  regEventReceiver = null;  super.onDetach();  }  private class RegisterReceiver extends BroadcastReceiver {  @Override  public void onReceive(Context arg0, Intent arg1) {  int refreshScreen = arg1.getIntExtra(NFCDoorlockHCE.BR\_REFRESH\_FILTER, 0);  if (refreshScreen == 1) {  tvRegDate.setText(SPHelper.getString(getActivity(),SPHelper.KEY\_REGISTRATION\_DATE));  }  }  }  } |

|  |
| --- |
| **NFCWifiSetupFragment.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp;  import android.Manifest;  import android.app.ProgressDialog;  import android.content.BroadcastReceiver;  import android.content.Context;  import android.content.Intent;  import android.content.IntentFilter;  import android.content.pm.PackageManager;  import android.net.wifi.ScanResult;  import android.net.wifi.WifiManager;  import android.os.Build;  import android.os.Bundle;  import android.os.Handler;  import android.support.v4.app.ActivityCompat;  import android.support.v4.app.Fragment;  import android.text.TextUtils;  import android.view.LayoutInflater;  import android.view.View;  import android.view.ViewGroup;  import android.widget.AdapterView;  import android.widget.Button;  import android.widget.ListView;  import android.widget.Switch;  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  import app.smartdoorlock.com.smartdoorlockandroidapp.Adapters.NFCWifiSetupListAdapter;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.WifiSignalEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Model.NFCWifiSetupModel;  import app.smartdoorlock.com.smartdoorlockandroidapp.Model.NFCWifiSetupModelComparator;  import app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper;  public class NFCWifiSetupFragment extends Fragment {  private Button btnRefresh;  private ListView lvWifi;  private NFCWifiSetupListAdapter wifiListAdapter;  private ArrayList<NFCWifiSetupModel> wifiModelList = new ArrayList<>();  private Switch swWifi;  private WifiManager wifiManager;  private static final int PERMISSIONS\_REQUEST\_CODE\_ACCESS\_COARSE\_LOCATION = 1001;  private ProgressDialog pDialog;  public NFCWifiSetupFragment() {  // Required empty public constructor  }  public static NFCWifiSetupFragment newInstance() {  NFCWifiSetupFragment fragment = new NFCWifiSetupFragment();  Bundle args = new Bundle();  fragment.setArguments(args);  return fragment;  }  @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  }  @Override  public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {  // Inflate the layout for this fragment  View v = inflater.inflate(R.layout.fragment\_nfc\_wifi\_setup, container, false);  btnRefresh = (Button)v.findViewById(R.id.fragment\_nfc\_wifi\_setup\_btn\_refresh);  btnRefresh.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  refreshWifiList();  }  });  swWifi = (Switch)v.findViewById(R.id.fragment\_nfc\_wifi\_setup\_sw\_wifi\_enable);  wifiListAdapter = new NFCWifiSetupListAdapter(getActivity(), wifiModelList);  lvWifi =(ListView)v.findViewById(R.id.fragment\_nfc\_wifi\_setup\_lv\_wifi);  lvWifi.setAdapter(wifiListAdapter);  lvWifi.setOnItemClickListener(new AdapterView.OnItemClickListener() {  @Override  public void onItemClick(AdapterView<?> parent, View view,  int position, long id) {  startSendActivity(wifiModelList.get(position));  }  });  wifiManager = (WifiManager)getActivity().getApplicationContext().getSystemService(Context.WIFI\_SERVICE);  registerWifiReceiver();  swWifi.setChecked(wifiManager.isWifiEnabled());  final Handler handler = new Handler();  handler.postDelayed(new Runnable() {  @Override  public void run() {  refreshWifiList();  }  }, 100);  SPHelper.putCommand(getActivity(),SPHelper.CURRENT\_COMMAND, CommandEnum.NONE);  return v;  }  private void requestPermissions() {  if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.M &&  getActivity().checkSelfPermission(Manifest.permission.ACCESS\_COARSE\_LOCATION) != PackageManager.PERMISSION\_GRANTED) {  ActivityCompat.requestPermissions(getActivity(),new String[]{  Manifest.permission.ACCESS\_NETWORK\_STATE,  Manifest.permission.ACCESS\_WIFI\_STATE,  Manifest.permission.CHANGE\_WIFI\_STATE,  Manifest.permission.ACCESS\_COARSE\_LOCATION,  Manifest.permission.INTERNET,  Manifest.permission.ACCESS\_FINE\_LOCATION},  PERMISSIONS\_REQUEST\_CODE\_ACCESS\_COARSE\_LOCATION );  }  }  private void startSendActivity(NFCWifiSetupModel wifiInfo) {  Intent intent = new Intent(getActivity(), NFCWifiSetupSendActivity.class);  Bundle b = new Bundle();  b.putParcelable("WifiInfo",wifiInfo);  intent.putExtras(b); //Put your id to your next Intent  startActivity(intent);  }  private NFCWifiSetupModel getWifiInfoFromSSID(String ssid) {  for (NFCWifiSetupModel wifiInfo : wifiModelList) {  if (wifiInfo.getSSID().equals(ssid))  return wifiInfo;  }  return null;  }  private void refreshWifiList() {  requestPermissions();  wifiManager.startScan();  pDialog = new ProgressDialog(getActivity());  pDialog.setMessage("Scanning for access points...");  pDialog.setTitle("Smart Doorlock");  pDialog.setIndeterminate(false);  pDialog.setCancelable(false);  pDialog.show();  }  private void showWifiList(List<ScanResult> scanList) {  wifiModelList.clear();  for (ScanResult result : scanList) {  //Check if same SSID has been added  NFCWifiSetupModel existingWifi = getWifiInfoFromSSID(result.SSID);  WifiSignalEnum signalStr = WifiSignalEnum.GetSignalFromLevel(result.level);  if (existingWifi == null) {  String securityType = result.capabilities.replace("[ESS]","");  boolean isEncrypted = !TextUtils.isEmpty(securityType);  NFCWifiSetupModel wifiInfo = new NFCWifiSetupModel(result.SSID,signalStr,isEncrypted,securityType);  wifiModelList.add(wifiInfo);  }  else {  if (signalStr.isStrongerThan(existingWifi.getSignal())) {  existingWifi.setSignal(signalStr);  }  }  }  Collections.sort(wifiModelList, new NFCWifiSetupModelComparator());  wifiListAdapter.notifyDataSetChanged();  pDialog.dismiss();  }  private void registerWifiReceiver(){  IntentFilter filter = new IntentFilter();  filter.addAction(WifiManager.SCAN\_RESULTS\_AVAILABLE\_ACTION);  getActivity().registerReceiver(new BroadcastReceiver(){  public void onReceive(Context context, Intent intent) {  StringBuilder sb = new StringBuilder();  List<ScanResult> scanList = wifiManager.getScanResults();  sb.append("\n Number Of Wifi connections :" + " " +scanList.size()+"\n\n");  showWifiList(scanList);  }  },filter);  }  @Override  public void onAttach(Context context) {  super.onAttach(context);  }  @Override  public void onDetach() {  super.onDetach();  }  @Override  public void onResume() {  super.onResume();  SPHelper.putCommand(getActivity(),SPHelper.CURRENT\_COMMAND, CommandEnum.NONE);  }  } |

|  |
| --- |
| **NFCWifiSetupSendActivity.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp;  import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.text.Editable;  import android.text.TextWatcher;  import android.widget.EditText;  import android.widget.TextView;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  import app.smartdoorlock.com.smartdoorlockandroidapp.Model.NFCWifiSetupModel;  import app.smartdoorlock.com.smartdoorlockandroidapp.Utility.SPHelper;  public class NFCWifiSetupSendActivity extends AppCompatActivity {  private NFCWifiSetupModel wifiInfo;  private TextView tvSSID;  private TextView tvEncryption;  private EditText etPassword;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_nfc\_wifi\_setup\_send);  Bundle b = getIntent().getExtras();  wifiInfo = b.getParcelable("WifiInfo");  tvSSID = (TextView) findViewById(R.id.activity\_nfc\_wifi\_setup\_send\_ssid);  tvSSID.setText(wifiInfo.getSSID());  tvEncryption = (TextView) findViewById(R.id.activity\_nfc\_wifi\_setup\_send\_encryption);  String encryption;  if (wifiInfo.isEncrypted()) {  encryption = wifiInfo.getEncryptionType();  tvEncryption.setText(encryption);  }  else {  encryption = "None";  tvEncryption.setText(encryption);  }  etPassword = (EditText) findViewById(R.id.activity\_nfc\_wifi\_setup\_send\_et\_password);  etPassword.addTextChangedListener(new TextWatcher() {  @Override  public void beforeTextChanged(CharSequence s, int start, int count, int after) {  }  @Override  public void onTextChanged(CharSequence s, int start, int before, int count) {  }  @Override  public void afterTextChanged(Editable s) {  SPHelper.putString(NFCWifiSetupSendActivity.this,SPHelper.KEY\_WIFI\_PASSWORD,s.toString());  }  });  SPHelper.putCommand(this,SPHelper.CURRENT\_COMMAND, CommandEnum.WIFI\_SETUP);  SPHelper.putString(this,SPHelper.KEY\_WIFI\_SSID,wifiInfo.getSSID());  SPHelper.putString(this,SPHelper.KEY\_WIFI\_ENCRYPTION,encryption);  }  } |

|  |
| --- |
| **SPHelper.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp.Utility;  import android.content.Context;  import android.content.SharedPreferences;  import android.preference.PreferenceManager;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.CommandEnum;  /\*\*  \* Created by shuh on 10/23/2016.  \*/  public class SPHelper {  private SPHelper() {}  public static final String CURRENT\_COMMAND = "CURRENT\_COMMAND";  public static final String SAVED\_COMMAND = "SAVED\_COMMAND";  public static final String KEY\_PHONE\_ID = "PHONE\_ID";  public static final String KEY\_REGISTRATION\_DATE = "REG\_DATE";  public static final String KEY\_WIFI\_SSID = "SSID";  public static final String KEY\_WIFI\_PASSWORD = "WIFI\_PASSWORD";  public static final String KEY\_WIFI\_ENCRYPTION = "ENCRYPTION";  public static void putCommand(Context ctx, String command, CommandEnum val) {  SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(ctx);  SharedPreferences.Editor editor = sp.edit();  editor.putString(command, val.toString());  editor.apply();  }  public static CommandEnum getCommand(Context ctx, String command) {  SharedPreferences sp = PreferenceManager.getDefaultSharedPreferences(ctx);  String enumStr = sp.getString(command, CommandEnum.NONE.toString());  return CommandEnum.toCommandEnum(enumStr);  }  public static void putString(Context ctx, String key, String content) {  SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(ctx);  SharedPreferences.Editor editor = preferences.edit();  editor.putString(key,content);  editor.apply();  }  public static String getString(Context ctx, String key) {  SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(ctx);  return preferences.getString(key, "");  }  } |

|  |
| --- |
| **NFCWifiSetupModelComparator.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp.Model;  import java.util.Comparator;  /\*\*  \* Created by shuh on 11/1/2016.  \*/  public class NFCWifiSetupModelComparator implements Comparator<NFCWifiSetupModel> {  @Override  public int compare(NFCWifiSetupModel o1, NFCWifiSetupModel o2) {  return o1.getSignal().isStrongerThan(o2.getSignal()) ? -1 : 1;  }  } |

|  |
| --- |
| **NFCWifiSetupModel.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp.Model;  import android.os.Parcel;  import android.os.Parcelable;  import app.smartdoorlock.com.smartdoorlockandroidapp.Enums.WifiSignalEnum;  /\*\*  \* Created by shuh on 11/1/2016.  \*/  public class NFCWifiSetupModel implements Parcelable {  private String ssid;  private WifiSignalEnum signal;  private boolean isEncrypted;  private String encryptionType;  public NFCWifiSetupModel(String ssid, WifiSignalEnum signal, boolean isEncrypted, String encryptionType) {  this.ssid = ssid;  this.signal = signal;  this.isEncrypted = isEncrypted;  this.encryptionType = encryptionType;  }  public String getSSID() {  return ssid;  }  public void setSsid(String ssid) {  this.ssid = ssid;  }  public WifiSignalEnum getSignal() {  return signal;  }  public void setSignal(WifiSignalEnum signal) {  this.signal = signal;  }  public String getEncryptionType() {  return encryptionType;  }  public void setEncryptionType(String encryptionType) {  this.encryptionType = encryptionType;  }  public boolean isEncrypted() {  return isEncrypted;  }  public void setEncrypted(boolean encrypted) {  isEncrypted = encrypted;  }  // Parcelling part  public NFCWifiSetupModel(Parcel in){  String[] data = new String[4];  in.readStringArray(data);  this.ssid = data[0];  this.signal = WifiSignalEnum.getSignalFromValue(Integer.parseInt(data[1]));  this.isEncrypted = Boolean.valueOf(data[2]);  this.encryptionType = data[3];  }  @Override  public int describeContents() {  return 0;  }  @Override  public void writeToParcel(Parcel dest, int flags) {  dest.writeStringArray(new String[]  {  this.ssid,  String.valueOf(this.signal.getValue()),  String.valueOf(this.isEncrypted),  this.encryptionType  });  }  public static final Parcelable.Creator CREATOR = new Parcelable.Creator() {  public NFCWifiSetupModel createFromParcel(Parcel in) {  return new NFCWifiSetupModel(in);  }  public NFCWifiSetupModel[] newArray(int size) {  return new NFCWifiSetupModel[size];  }  };  } |

|  |
| --- |
| **WifiSignalEnum.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp.Enums;  /\*\*  \* Created by shuh on 11/1/2016.  \*/  public enum WifiSignalEnum {  FIVE(5),FOUR(4),THREE(3),TWO(2),ONE(1),NONE(0);  private Integer strength;  WifiSignalEnum(int strength) {  this.strength = strength;  }  public int getValue() {  return strength;  }  public static WifiSignalEnum getSignalFromValue(int val) {  switch (val) {  case 5:  return FIVE;  case 4:  return FOUR;  case 3:  return THREE;  case 2:  return TWO;  case 1:  return ONE;  case 0:  return NONE;  }  return null;  }  public boolean isStrongerThan(WifiSignalEnum other) {  return this.strength > other.strength;  }  public static WifiSignalEnum GetSignalFromLevel(int level) {  if (level >= -50) {  return FIVE;  }  else if (level >= -65) {  return FOUR;  }  else if (level >= -75) {  return THREE;  }  else if (level >= -85) {  return TWO;  }  else if (level >= -95) {  return ONE;  }  else {  return NONE;  }  }  } |

|  |
| --- |
| **CommandEnum.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp.Enums;  /\*\*  \* Created by shuh on 10/23/2016.  \*/  public enum CommandEnum {  NONE, DOORLOCK\_CONTROL, DOORLOCK\_REGISTRATION, WIFI\_SETUP;  public static CommandEnum toCommandEnum (String enumString) {  try {  return valueOf(enumString);  } catch (Exception ex) {  // For error cases  return NONE;  }  }  } |

|  |
| --- |
| **NFCWifiSetupListAdapter.java** |
| package app.smartdoorlock.com.smartdoorlockandroidapp.Adapters;  /\*\*  \* Created by shuh on 11/1/2016.  \*/  import android.content.Context;  import android.view.LayoutInflater;  import android.view.View;  import android.view.ViewGroup;  import android.widget.ArrayAdapter;  import android.widget.ImageView;  import android.widget.TextView;  import java.util.ArrayList;  import app.smartdoorlock.com.smartdoorlockandroidapp.Model.NFCWifiSetupModel;  import app.smartdoorlock.com.smartdoorlockandroidapp.R;  public class NFCWifiSetupListAdapter extends ArrayAdapter<NFCWifiSetupModel> {  private Context mCtx;  private ArrayList<NFCWifiSetupModel> mWifiList;  private ViewHolder mViewHolder;  private static class ViewHolder {  private TextView tvSSID;  private TextView tvEncrypted;  private ImageView ivWifiSignal;  private ImageView ivWifiEncrypted;  }  public NFCWifiSetupListAdapter(Context context, ArrayList<NFCWifiSetupModel> wifiList) {  super(context, R.layout.listview\_nfc\_wifi\_setup, wifiList);  mWifiList = wifiList;  mCtx = context;  }  public View getView(int position, View convertView, ViewGroup parent) {  if (convertView == null) {  convertView = LayoutInflater.from(this.getContext())  .inflate(R.layout.listview\_nfc\_wifi\_setup, parent, false);  mViewHolder = new ViewHolder();  mViewHolder.tvSSID = (TextView) convertView.findViewById(R.id.lv\_nfc\_wifi\_setup\_tv\_ssid\_name);  mViewHolder.ivWifiSignal = (ImageView) convertView.findViewById(R.id.lv\_nfc\_wifi\_setup\_iv\_wifi\_signal);  mViewHolder.ivWifiEncrypted = (ImageView) convertView.findViewById(R.id.lv\_nfc\_wifi\_setup\_iv\_wifi\_encrypted);  mViewHolder.tvEncrypted = (TextView) convertView.findViewById(R.id.lv\_nfc\_wifi\_setup\_tv\_encrypted);  convertView.setTag(mViewHolder);  } else {  mViewHolder = (ViewHolder) convertView.getTag();  }  NFCWifiSetupModel wifiInfo = getItem(position);  if (wifiInfo!= null) {  mViewHolder.tvSSID.setText(wifiInfo.getSSID());  if (!wifiInfo.isEncrypted()) {  mViewHolder.ivWifiEncrypted.setVisibility(View.GONE);  mViewHolder.tvEncrypted.setVisibility(View.GONE);  }  switch (wifiInfo.getSignal()) {  case FIVE:  mViewHolder.ivWifiSignal.setImageResource(R.drawable.wifi\_5\_bar);  break;  case FOUR:  mViewHolder.ivWifiSignal.setImageResource(R.drawable.wifi\_4\_bar);  break;  case THREE:  mViewHolder.ivWifiSignal.setImageResource(R.drawable.wifi\_3\_bar);  break;  case TWO:  mViewHolder.ivWifiSignal.setImageResource(R.drawable.wifi\_2\_bar);  break;  case ONE:  mViewHolder.ivWifiSignal.setImageResource(R.drawable.wifi\_1\_bar);  break;  case NONE:  mViewHolder.ivWifiSignal.setImageResource(R.drawable.wifi\_0\_bar);  break;  }  }  return convertView;  }  } |

|  |
| --- |
| **activity\_main.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/drawer\_layout"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:fitsSystemWindows="true"  tools:openDrawer="start">  <include  layout="@layout/app\_bar\_main"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent" />  <android.support.design.widget.NavigationView  android:id="@+id/nav\_view"  android:layout\_width="wrap\_content"  android:layout\_height="match\_parent"  android:layout\_gravity="start"  android:fitsSystemWindows="true"  app:headerLayout="@layout/nav\_header\_main"  app:menu="@menu/activity\_main\_drawer" />  </android.support.v4.widget.DrawerLayout> |

|  |
| --- |
| **activity\_nfc\_wifi\_setup\_send.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_nfcwifi\_setup\_send\_fragment"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  tools:context="app.smartdoorlock.com.smartdoorlockandroidapp.NFCWifiSetupSendActivity">  <RelativeLayout  android:id="@+id/activity\_nfc\_wifi\_setup\_rl\_si\_header"  android:background="@drawable/bg\_category\_header"  android:layout\_width="match\_parent"  android:layout\_height="36dp">  <TextView  android:text="Wifi Setup (NFC)"  style="@style/rl\_category\_header" />  </RelativeLayout>  <TableLayout  android:layout\_marginLeft="6dp"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <TableRow android:layout\_marginTop="10dp">  <TextView  android:textSize="17sp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginRight="10dp"  android:text="SSID">  </TextView>  <TextView  android:textSize="17sp"  android:id="@+id/activity\_nfc\_wifi\_setup\_send\_ssid"  android:layout\_width="0dp"  android:layout\_weight="1"  android:gravity="center"  android:layout\_height="wrap\_content">  </TextView>  </TableRow>  <TableRow android:layout\_marginTop="10dp">  <TextView  android:textSize="17sp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginRight="10dp"  android:text="Encryption">  </TextView>  <TextView  android:textSize="17sp"  android:id="@+id/activity\_nfc\_wifi\_setup\_send\_encryption"  android:layout\_width="0dp"  android:layout\_weight="1"  android:gravity="center"  android:layout\_height="wrap\_content">  </TextView>  </TableRow>  <TableRow android:layout\_marginTop="10dp">  <TextView  android:textSize="17sp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"    android:layout\_marginRight="10dp"  android:text="Password">  </TextView>  <EditText  android:textSize="17sp"  android:id="@+id/activity\_nfc\_wifi\_setup\_send\_et\_password"  android:layout\_width="0dp"  android:layout\_weight="1"  android:gravity="center"  android:inputType="textPassword"  android:layout\_height="wrap\_content">  </EditText>  </TableRow>  </TableLayout>  <TextView android:layout\_height="match\_parent"  android:layout\_width="match\_parent"  android:gravity="center"  android:textSize="30sp"  android:textStyle="bold"  android:text="Tap phone to doorlock to\ntransfer wifi information">  </TextView>  </LinearLayout> |

|  |
| --- |
| **app\_bar\_main.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:fitsSystemWindows="true"  tools:context="app.smartdoorlock.com.smartdoorlockandroidapp.MainActivity">  <android.support.design.widget.AppBarLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:theme="@style/AppTheme.AppBarOverlay">  <android.support.v7.widget.Toolbar  android:id="@+id/toolbar"  android:layout\_width="match\_parent"  android:layout\_height="?attr/actionBarSize"  android:background="?attr/colorPrimary"  app:popupTheme="@style/AppTheme.PopupOverlay" />  </android.support.design.widget.AppBarLayout>  <FrameLayout  android:id="@+id/fl\_frag\_content"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingTop="@dimen/activity\_vertical\_margin"  app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior"  />  </android.support.design.widget.CoordinatorLayout> |

|  |
| --- |
| **fragment\_nfc\_doorlock\_control.xml** |
| <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="app.smartdoorlock.com.smartdoorlockandroidapp.NFCDoorlockControlFragment">  <!-- TODO: Update blank fragment layout -->  <TextView  android:id="@+id/fragment\_nfc\_doorlock\_ctl\_tv\_info"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:gravity="center"  android:textSize="18sp"  android:text="NFC doorlock control active\n\nPlease tap phone to the doorlock" />  </FrameLayout> |

|  |
| --- |
| **fragment\_nfc\_doorlock\_registration.xml** |
| <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="app.smartdoorlock.com.smartdoorlockandroidapp.NFCDoorlockRegistrationFragment">  <RelativeLayout  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <RelativeLayout  android:id="@+id/fragment\_nfc\_doorlock\_reg\_rl\_si\_header"  android:background="@drawable/bg\_category\_header"  android:layout\_width="match\_parent"  android:layout\_height="36dp">  <TextView  android:text="Smartphone Information"  style="@style/rl\_category\_header" />  </RelativeLayout>  <TableLayout  android:id="@+id/fragment\_nfc\_doorlock\_reg\_tl\_si\_info"  android:layout\_below="@id/fragment\_nfc\_doorlock\_reg\_rl\_si\_header"  android:layout\_marginLeft="6dp"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <TableRow  android:layout\_marginTop="10dp">  <TextView  android:textSize="17sp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginRight="10dp"  android:text="Smartphone ID: ">  </TextView>  <TextView  android:textSize="17sp"  android:id="@+id/fragment\_nfc\_doorlock\_reg\_tv\_smartphone\_id"  android:layout\_width="0dp"  android:layout\_weight="1"  android:gravity="center"  android:layout\_height="wrap\_content">  </TextView>  </TableRow>  <TableRow  android:layout\_marginTop="10dp">  <TextView  android:textSize="17sp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginRight="10dp"  android:text="Registration Date: ">  </TextView>  <TextView  android:textSize="17sp"  android:id="@+id/fragment\_nfc\_doorlock\_reg\_tv\_reg\_date"  android:layout\_width="0dp"  android:layout\_weight="1"  android:gravity="center"  android:layout\_height="wrap\_content">  </TextView>  </TableRow>  </TableLayout>  <TextView  android:layout\_below="@id/fragment\_nfc\_doorlock\_reg\_tl\_si\_info"  android:id="@+id/fragment\_nfc\_doorlock\_reg\_tv\_help"  android:text="Press button to generate a random ID\n\nTap phone to doorlock to register"  android:gravity="center"  android:textSize="18sp"  android:textStyle="bold"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"/>  <Button  android:id="@+id/fragment\_nfc\_doorlock\_btn\_gen\_id"  style="@style/btn\_random\_id"/>  </RelativeLayout>  </FrameLayout> |

|  |
| --- |
| **fragment\_nfc\_wifi\_setup.xml** |
| <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="app.smartdoorlock.com.smartdoorlockandroidapp.NFCWifiSetupFragment">  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <RelativeLayout  android:id="@+id/fragment\_nfc\_wifi\_setup\_rl\_top"  android:background="@drawable/bg\_category\_header"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <Button  android:id="@+id/fragment\_nfc\_wifi\_setup\_btn\_refresh"  android:layout\_marginLeft="5dp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Refresh">  </Button>  <Switch  android:id="@+id/fragment\_nfc\_wifi\_setup\_sw\_wifi\_enable"  android:layout\_alignParentRight="true"  android:layout\_marginRight="5dp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text=" Wifi "  android:layout\_centerVertical="true" />  </RelativeLayout>  <ListView  android:id="@+id/fragment\_nfc\_wifi\_setup\_lv\_wifi"  android:layout\_below="@id/fragment\_nfc\_wifi\_setup\_rl\_top"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent" />  </RelativeLayout>  </FrameLayout> |

|  |
| --- |
| **listview\_nfc\_wifi\_setup.xml** |
| <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context="app.smartdoorlock.com.smartdoorlockandroidapp.NFCWifiSetupFragment">  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <RelativeLayout  android:id="@+id/fragment\_nfc\_wifi\_setup\_rl\_top"  android:background="@drawable/bg\_category\_header"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <Button  android:id="@+id/fragment\_nfc\_wifi\_setup\_btn\_refresh"  android:layout\_marginLeft="5dp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Refresh">  </Button>  <Switch  android:id="@+id/fragment\_nfc\_wifi\_setup\_sw\_wifi\_enable"  android:layout\_alignParentRight="true"  android:layout\_marginRight="5dp"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text=" Wifi "  android:layout\_centerVertical="true" />  </RelativeLayout>  <ListView  android:id="@+id/fragment\_nfc\_wifi\_setup\_lv\_wifi"  android:layo<?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent"  android:orientation="horizontal">  <RelativeLayout  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content">  <ImageView  android:id="@+id/lv\_nfc\_wifi\_setup\_iv\_wifi\_signal"  android:layout\_width="50dp"  android:layout\_height="50dp"  android:layout\_marginBottom="5dp"  android:layout\_marginLeft="5dp"  android:layout\_marginRight="5dp"  android:layout\_marginTop="5dp"  android:src="@drawable/wifi\_5\_bar" />  <ImageView  android:id="@+id/lv\_nfc\_wifi\_setup\_iv\_wifi\_encrypted"  android:layout\_width="25dp"  android:layout\_height="25dp"  android:layout\_marginBottom="5dp"  android:layout\_marginLeft="5dp"  android:layout\_marginRight="5dp"  android:layout\_marginTop="5dp"  android:src="@drawable/wifi\_encrypted" />  </RelativeLayout>  <LinearLayout  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginTop="5dp"  android:layout\_marginLeft="5dp"  android:orientation="vertical">  <TextView  android:id="@+id/lv\_nfc\_wifi\_setup\_tv\_ssid\_name"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:textSize="25sp"  android:text="TEST SSID" />  <TextView  android:id="@+id/lv\_nfc\_wifi\_setup\_tv\_encrypted"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:textSize="15sp"  android:text="Encrypted">  </TextView>  </LinearLayout>  </LinearLayout>ut\_below="@id/fragment\_nfc\_wifi\_setup\_rl\_top"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent" />  </RelativeLayout>  </FrameLayout> |

|  |
| --- |
| **nav\_header\_main.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:background="@drawable/side\_nav\_bar"  android:gravity="bottom"  android:orientation="vertical"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingTop="16dp"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:theme="@style/ThemeOverlay.AppCompat.Dark">  <ImageView  android:id="@+id/imageView"  android:paddingTop="@dimen/nav\_header\_vertical\_spacing"  android:layout\_width="60dp"  android:layout\_height="58dp"  android:src="@drawable/sd\_main\_icon" />  <TextView  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:paddingTop="@dimen/nav\_header\_vertical\_spacing"  android:text="Smart Doorlock"  android:textAppearance="@style/TextAppearance.AppCompat.Body1" />  </LinearLayout> |

|  |
| --- |
| **bg\_category\_header.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <layer-list xmlns:android="http://schemas.android.com/apk/res/android" >  <item>  <shape  android:shape="rectangle">  <stroke android:width="1dp" android:color="#FF000000" />  <solid android:color="#FFDDDDDD" />  </shape>  </item>  <item android:top="1dp" android:bottom="1dp">  <shape  android:shape="rectangle">  <stroke android:width="1dp" android:color="#FFDDDDDD" />  <solid android:color="#00000000" />  </shape>  </item>  </layer-list> |

|  |
| --- |
| **side\_nav\_bar.xml** |
| <shape xmlns:android="http://schemas.android.com/apk/res/android"  android:shape="rectangle">  <gradient  android:angle="135"  android:centerColor="#4CAF50"  android:endColor="#2E7D32"  android:startColor="#81C784"  android:type="linear" />  </shape> |

|  |
| --- |
| **AndroidManifest.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="app.smartdoorlock.com.smartdoorlockandroidapp">  <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  <uses-permission android:name="android.permission.INTERNET" />  <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  <uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />  <uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />  <uses-permission android:name="android.permission.NFC" />  <application  android:allowBackup="true"  android:icon="@mipmap/ic\_launcher"  android:label="@string/app\_name"  android:supportsRtl="true"  android:theme="@style/AppTheme">  <activity  android:name=".MainActivity"  android:label="@string/app\_name"  android:theme="@style/AppTheme.NoActionBar">  <intent-filter>  <action android:name="android.intent.action.MAIN" />  <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>  <!-- BEGIN\_INCLUDE(CardEmulationManifest) -->  <!-- Service for handling communication with NFC terminal. -->  <service  android:name=".NFCDoorlockHCE"  android:exported="true"  android:permission="android.permission.BIND\_NFC\_SERVICE">  <!-- Intent filter indicating that we support card emulation. -->  <intent-filter>  <action android:name="android.nfc.cardemulation.action.HOST\_APDU\_SERVICE" />  <category android:name="android.intent.category.DEFAULT" />  </intent-filter>  <!--  Required XML configuration file, listing the AIDs that we are emulating cards  for. This defines what protocols our card emulation service supports.  -->  <meta-data  android:name="android.nfc.cardemulation.host\_apdu\_service"  android:resource="@xml/aid\_list" />  </service>  <!-- END\_INCLUDE(CardEmulationManifest) -->  <activity android:name=".NFCWifiSetupSendActivity"></activity>  </application>  </manifest> |

# **10. References**

[1] "SimpleLink Wi-Fi CC3200 LaunchPad," in *Texas Instruments*. [Online]. Available: http://www.ti.com/tool/cc3200-launchxl. Accessed: May 12, 2016.

[2] "TRF7970A Multi-Protocol Fully Integrated 13.56-MHz NFC / RFID Transceiver IC," in *Texas Instruments*, 2016. [Online]. Available: http://www.ti.com/product/TRF7970A. Accessed: May 12, 2016.

[3] "Near field communication," in *Android Developers*, Google. [Online]. Available: http://developer.android.com/guide/topics/connectivity/nfc/index.html. Accessed: May 12, 2016

[4] "5 things to know about MQTT – the protocol for Internet of things," in *IBM Developer Works*, 2014. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/5things/entry/5\_things\_to\_know\_about\_mqtt\_the\_protocol\_for\_internet\_of\_things?lang=en. Accessed: May 12, 2016.

[5] "Storm 720GFX series, 12 button Keypad with snap-on Keytops," in *Storm Interface*. [Online]. Available: http://www.storm-interface.com/keypads/gfx-series/storm-720gfx-series-12-button.html. Accessed: May 12, 2016.

[6] "LCD MODULE 4x20 - 3.75mm," in *Electronic Assembly*. [Online]. Available: http://www.lcd-module.de/fileadmin/eng/pdf/doma/dip203-4e.pdf. Accessed: May 12, 2016.

<http://www.embedx.com/pdfs/ISO_STD_7816/info_isoiec7816-4%7Bed2.0%7Den.pdf>

<http://www.atmel.com/images/doc2056.pdf>

<http://jpkc.szpt.edu.cn/2007/sznk/UploadFile/biaozhun/iso14443/14443-3.pdf>

<http://jpkc.szpt.edu.cn/2007/sznk/UploadFile/biaozhun/iso14443/14443-4.pdf>

https://www.safaribooksonline.com/library/view/beginning-nfc/9781449324094/ch04.html

<https://developer.android.com/guide/topics/connectivity/nfc/index.html>

<https://en.wikipedia.org/wiki/Internet_of_things>

<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

http://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe