

SMRTACSDG

Smart Access Platform Solution Software Developer Guide

Rev. 1 — 19 December 2022

User guide

Document information

Information	Content
Keywords	SMRTACSDG, Smart Access Platform, Matter, PIN pad, UWB, Fingerprint, Face recognition
Abstract	This guide provides details about system design and software architecture for Smart Access Platform.



1 Introduction

NXP Smart Access Platform Solution is a comprehensive platform that supports several MCU-based access options, for example, Bluetooth LE, NFC, UWB, Face recognition, and Matter. This guide provides details about system design and software architecture for Smart Access Platform. It allows developers to easily and efficiently create smart access applications based on this platform.

1.1 Smart Access Platform overview

The solution endeavors to build a complete and scalable platform providing all kinds of MCU-based access options.

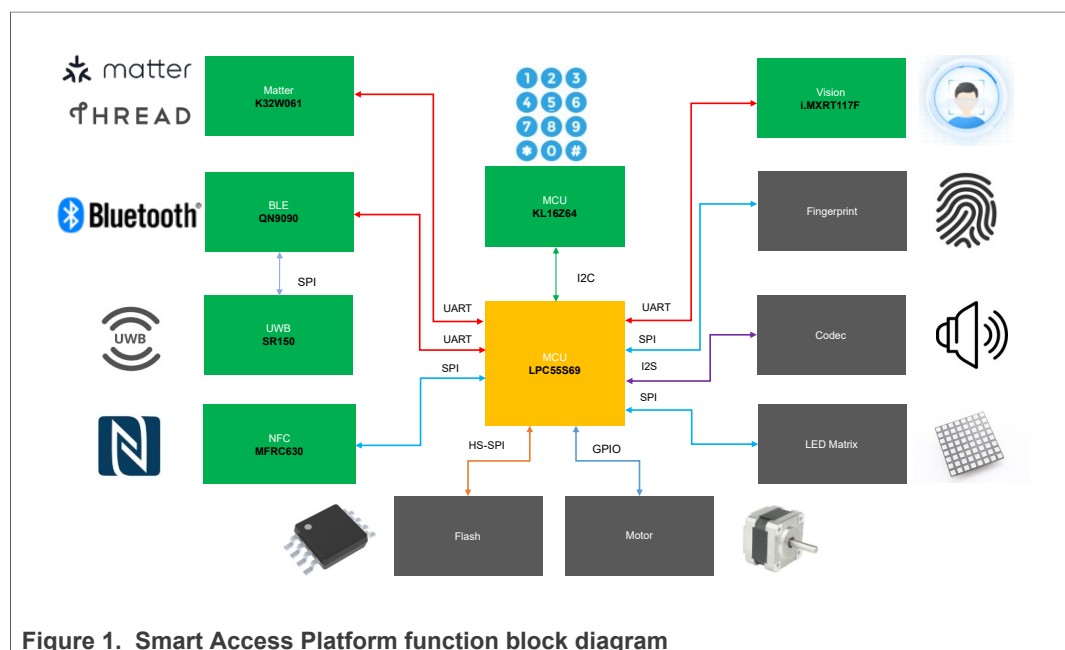


Figure 1. Smart Access Platform function block diagram

This scheme integrates various NXP standalone solutions, including [E-Lock Demo Kit](#), [SLN-VIZN3D-IOT](#), [NFC](#), [BLE](#), [UWB](#), and [Matter](#).

The solution consists of five PCBAs (Main Board, Conversion Board, Vision Board, Wireless Board, and Touch Board) and one APK (Smart Access Manager). The details of each are introduced in their dedicated sections.

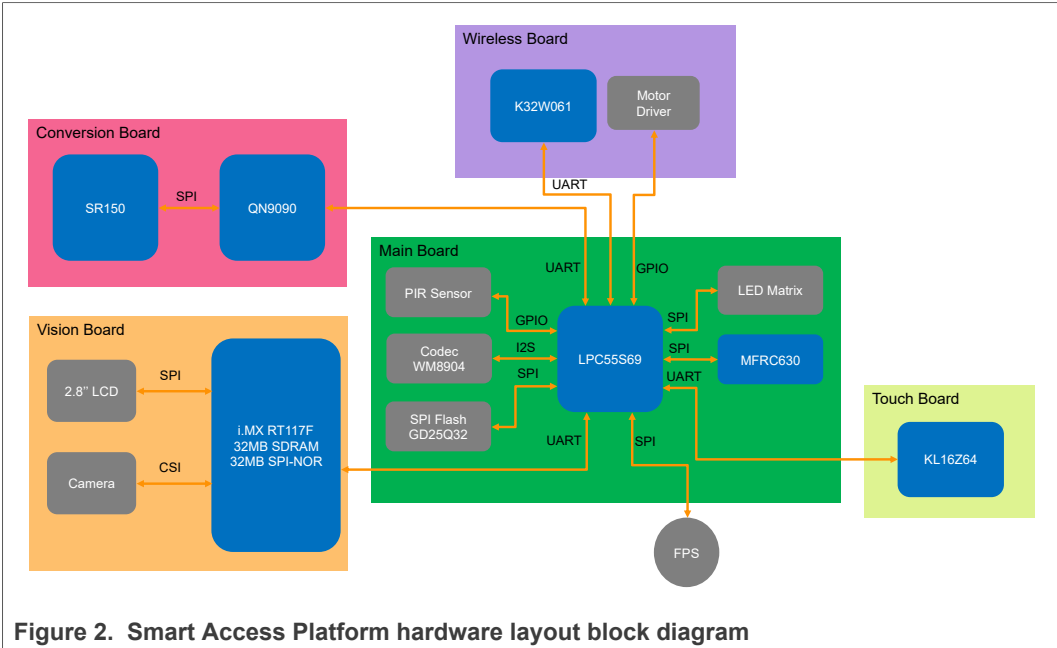


Figure 2. Smart Access Platform hardware layout block diagram

1.2 Software development environment

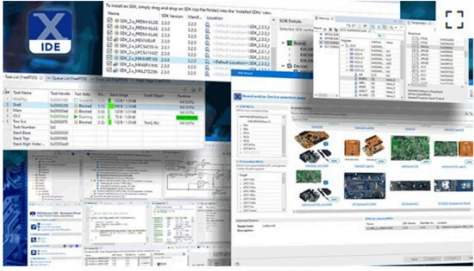
The software components within Smart Access Platform Solution are mainly developed by [MCUXpresso IDE](#). To learn how to set up the IDE, install the SDKs, import the projects, and build out firmware and program them to the MCU, see [MCUXpresso IDE User Guide](#).

Figure 3. Download MCUXpresso IDE

MCUXpresso Integrated Development Environment (IDE)

MCUXpresso-IDE [Receive alerts](#)

[Overview](#) [Software Details](#) [Design Resources](#) [Training](#) [Support](#)



Roll over image to zoom in

The MCUXpresso IDE brings developers an easy-to-use Eclipse-based development environment for NXP® MCUs based on Arm® Cortex®-M cores, including its general purpose crossover and wireless - enabled MCUs. The MCUXpresso IDE offers advanced editing, compiling, and debugging features with the addition of MCU-specific debugging views, code trace and profiling, multicore debugging, and integrated configuration tools. The MCUXpresso IDE debug connections support Freedom, Tower® system, LPCXpresso, i.MX RT-based EVKs, and your custom development boards with optimized open-source and commercial debug probes from NXP, P&E Micro®, and SEGGER®.

[DOWNLOADS](#)

[USER GUIDE](#)

For now, open below projects source code under [GitHub](#): Main Controller (LPC55S69), Face Recognition (i.MX RT117F), and Smart Access Manager (APK).

2 Boards introduction

2.1 Main controller

The MCU [LPC55S69](#) acts as the main controller of the platform. It aggregates various access options based on its powerful processing ability, rich on-chip flash and SRAM memory, and various Flexcomm interfaces and GPIOs.

The LPC55S69 main controller uses **bootloader + application** architecture, both of them are based on SDK_2.11.1_LPCXpresso55S69.

If the pin `GPIO0_20` is asserted, the bootloader updates the audio prompt files, or it directly jumps to the application.

Application is the main controller implementation to communicate with other components.

These components are as follows:

- Capacitive touch PIN Pad (KL16)
- Face recognition (i.MX RT117F)
- Matter (K32W)
- Bluetooth Low Energy (QN9090)
- NFC reader (MFRC630)
- Fingerprint sensor (BTL192)
- SPI NOR flash (GD25Q32)
- Audio codec (WM8904)
- LED matrix controller (74HC595)
- Motor driver (DRV8837), and so on

2.1.1 Hardware layout

[Figure 4](#) shows the hardware layout.

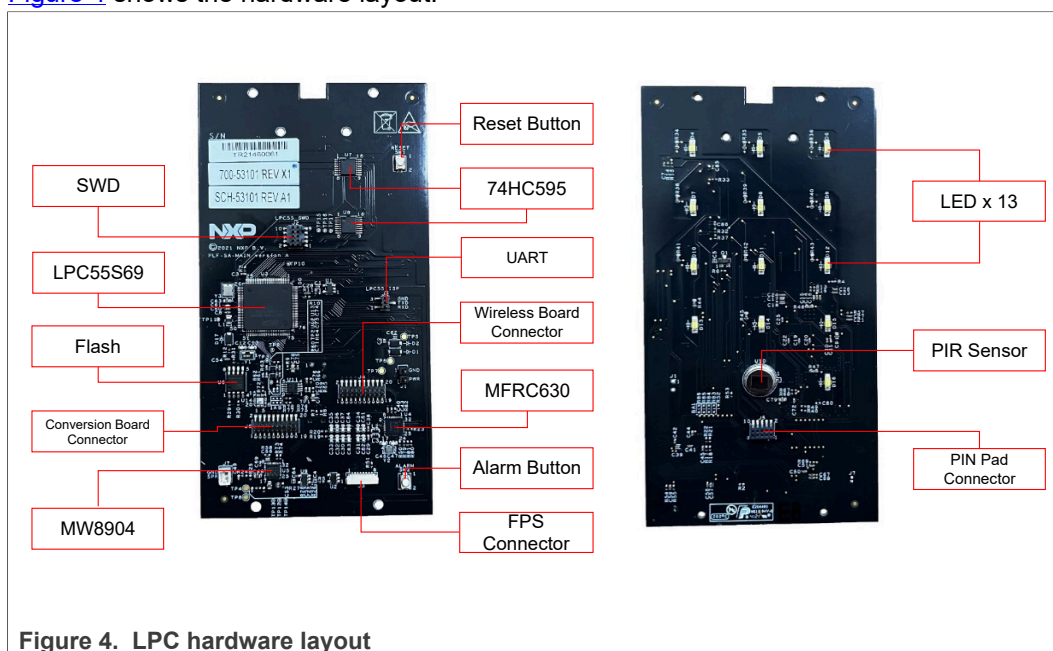


Figure 4. LPC hardware layout

2.1.2 Software work flow

Figure 5 shows the software work flow.

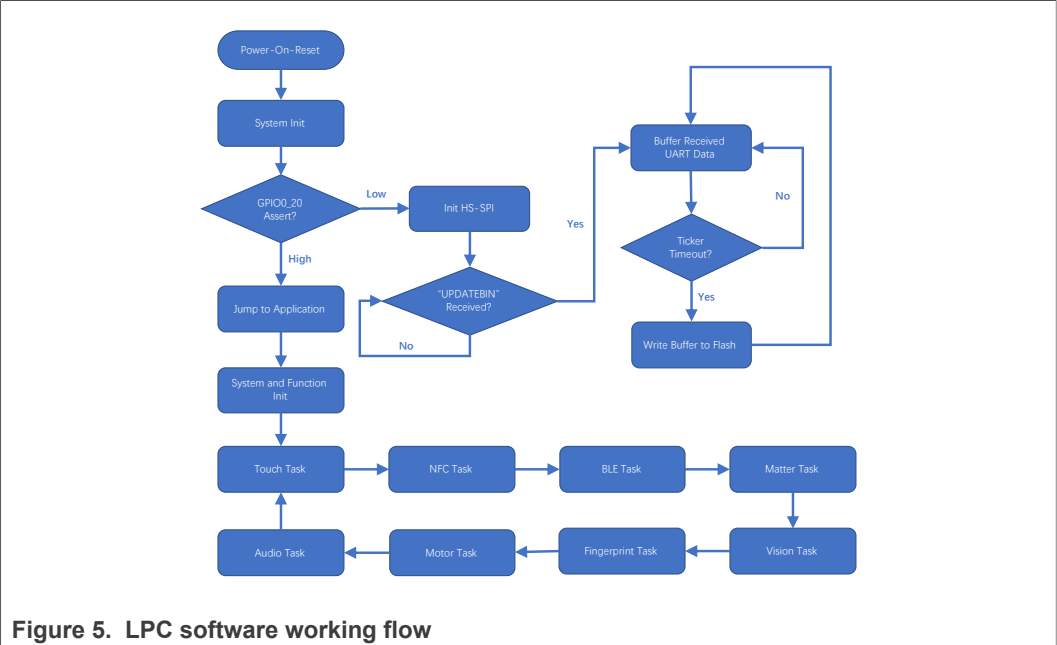


Figure 5. LPC software working flow

2.1.3 Application software diagram

Figure 6 shows the application software diagram.

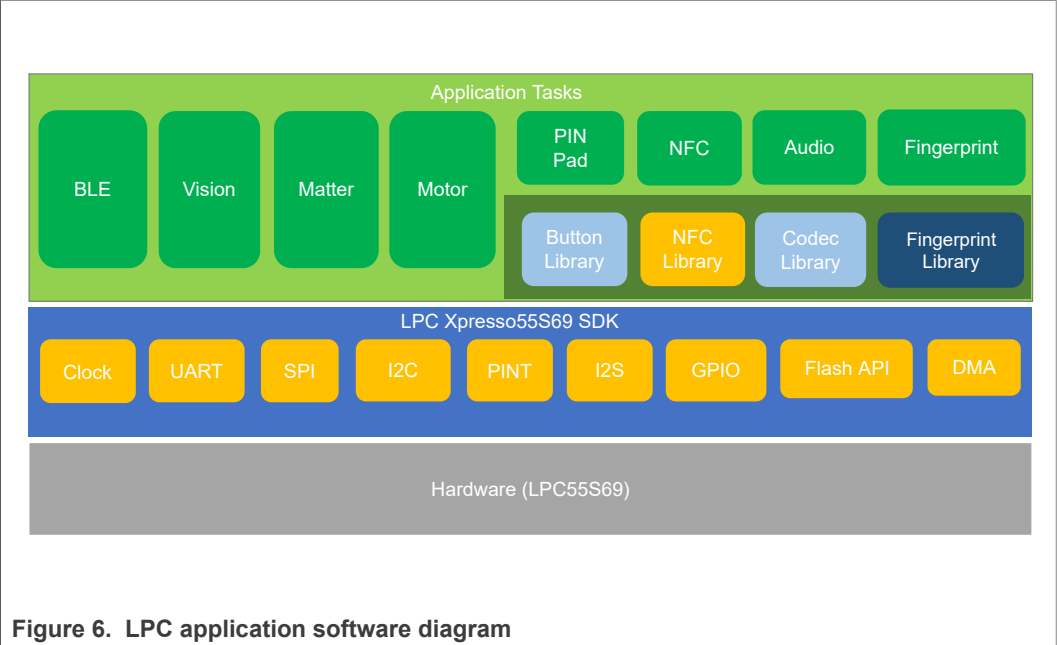


Figure 6. LPC application software diagram

2.1.4 Key APIs

Table 1 shows the key APIs of bootloader.

Table 1. Bootloader APIs

API name	Description	File
DEBUG_UART_IRQ_HANDLER	UART interrupt handler	app_printf.c
binupdate_task	Audio files update task	app_binupdate.c
spiflash_write_file	Write file to external SPI flash	app_spiflash.c
clean_boot	Jump to application	main.c

[Table 2](#) shows the key APIs of application.

Table 2. Application APIs

API name	Description	File
APP_SYS_InfoLoad	Load user information from flash	app_syscfg.c
APP_SYS_InfoSave	Save user information to flash	app_syscfg.c
APP_CAPT_Task	Capacitive PIN pad task	app_captouch.c
APP_NFC_Task	NFC reader task	app_nfcreader.c
APP_BLE_Task	BLE AT command process task	app_ble.c
APP_MATTER_Task	Matter AT command process task	app_matter.c
APP_FACEID_Task	Vision AT command process task	app_faceid.c
APP_FPS_Enroll_Task	Fingerprint enrollment task	app_fingerprint.c
APP_FPS_Verify_Task	Fingerprint verification task	app_fingerprint.c
APP_MOTOR_Task	Lock or unlock action task	app_motor.c
APP_MP3_Play	MP3 play task	app_mp3play.c

2.1.5 Memory layout

[Table 3](#) shows the memory layout.

Table 3. Memory layout

Start address	End address	Type	Size	Comment
0x0000 0000	0x0000 7FFF	Flash	32 kB	Bootloader
0x0000 8000	0x0005 FFFF	Flash	352 kB	Application
0x0006 0000	0x0009 1FFFF	Flash	200 kB	Fingerprint template
0x0009 2000	0x0009 7FFFF	Flash	24 kB	Reserved
0x0009 8000	0x0009 BFFF	Flash	16 kB	User data
0x0009 C000	0x0009 FFFF	Flash	16 kB	PFR
0x0400 0000	0x0400 3FFF	SRAM	16 kB	Audio file buffer
0x0400 4000	0x0400 7FFF	SRAM	16 kB	Reserved

Table 3. Memory layout...continued

Start address	End address	Type	Size	Comment
0x2000 0000	0x2002 BFFF	SRAM	176 kB	Data (incl. 88 kB heap and 4 kB stack)
0x2002 C000	0x2003 0FFF	SRAM	20 kB	MP3 buffer
0x2003 1000	0x2003 FFFF	SRAM	60 kB	Reserved
0x2004 0000	0x2004 3FFF	SRAM	16 kB	Codec buffer
0x4010 0000	0x4010 3FFFF	SRAM	16 kB	Reserved

2.2 BLE (Bluetooth Low Energy) and UWB (Ultra-Wideband)

The main scope of this application is to showcase the BLE wireless transmission together with UWB precise ranging technologies.

The BLE MCU [QN9090](#) provides the communication bridge between the APK and the main controller. It transmits AT commands from board to phone and reverse. The QN9090 also establishes BLE and UWB connection and interprets UWB ranging data.

The [UWB](#) technology enables secure ranging and precision sensing, creating a dimension of spatial context for wireless devices.

The application combines the BLE "Wireless UART" and UWB "Nearby Interaction" functions.

The association of BLE and UWB uses the following technologies/devices:

- QN9090 BLE: Transparent communication bridge of APK, UWB, and LPC
- SR150: UWB transceiver IC
- Multiple BLE services with wireless UART Service/Characteristic (128 bits customized UUID) + qpps service supporting both Android and iOS
- UWB Nearby Interaction for ranging between Conversion board and iOS/Android UWB capable phones

[Figure 7](#) shows the entire software diagram of the UWB and BLE system.

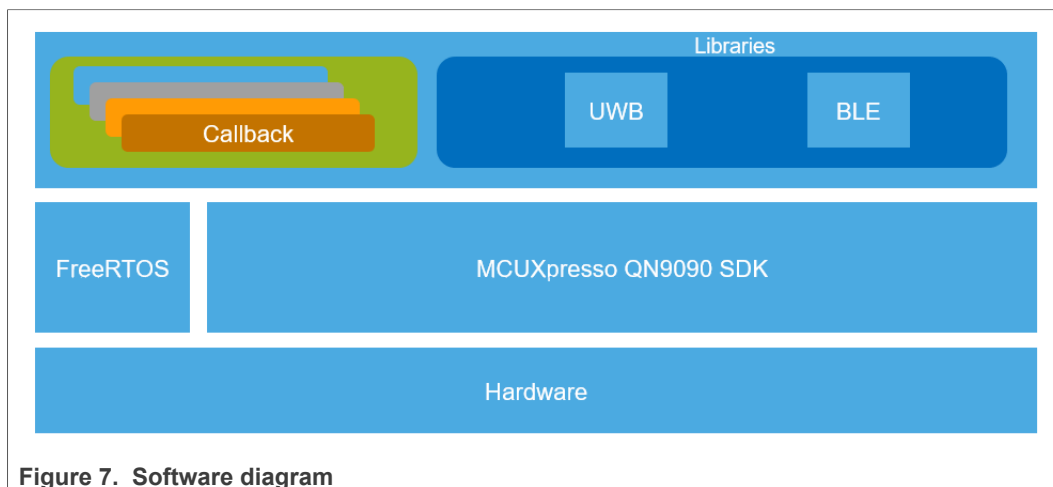


Figure 7. Software diagram

2.2.1 Nearby devices prerequisites

The Conversion Board can be used for interaction with an iOS/Android capable UWB device.

The prerequisites for the iOS phone are:

- iPhone 11 or later.
Note: Starting with iPhone 11, Apple phones come with the U1 Ultra-wideband chip.
- [NXP Trimensions AR app](#) from Apple store.
Note: Requires iOS 15.0 or later.
- For more information about iOS UWB capabilities, see [Nearby Interaction](#).

The prerequisites for the Android phone are:

- Samsung Galaxy S21+, S21 Ultra, and S22+.
- Basic UWB demo APK, which can be downloaded from [Github](#).
- Android 12 or later with Jetpack [UWB library](#).
Note: Xiaomi Mi Mix4 with MIUI 13 (Android 12.x) can also work, but does not support AR (Augmented reality) core.

2.2.2 BLE and UWB commands diagram

[Figure 8](#) shows the AT commands mechanism. It maintains uniformity across different devices, so they can communicate with ease.

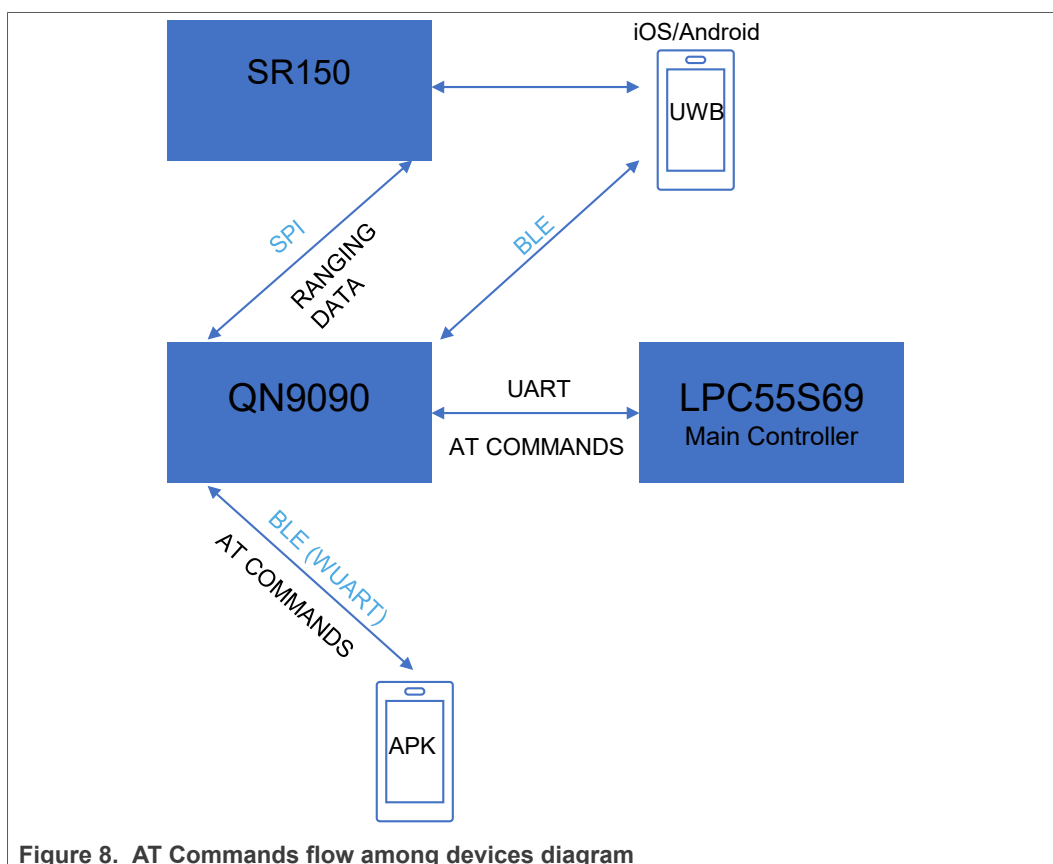


Figure 8. AT Commands flow among devices diagram

The foundation block for BLE is the Bluetooth Generic Attribute Profile (GATT) which defines the way a central device interacts with any peripheral device. To achieve this, the `gatt_database.c` is needed, alongside other configurations.

Additionally, the `wireless_uart` is needed to emulate a UART between the APK and Main Board, therefore, establishing a connection between the central and peripheral devices. The `qpps` is used for establishing the BLE connection between QN9090 and iOS/Android phones.

The last step is to configure the new task corresponding to the BLE inside the ranging demo. The maximum number of connections supported on a service is configured using `gAppMaxConnections_c`.

2.2.3 Demos used

This smart lock demo is based on `demo_NearbyInteraction.c` on `RhodesV4_SE`. Use `UWBIOT_APP_BUILD.h` to select the relevant demo for your application.

The device that is flashed, in our case it is the Conversion Board, with `demo_NearbyInteraction.c` has the `deviceType` of `kUWB_DeviceType_Controller`, which can be correlated with the initiator. The other device, in our case it is the UWB capable phone, has the `deviceType` of `kUWB_DeviceType_Controllee`, which can be correlated with the responder.

2.2.4 Relevant changes for smart lock actuation logic

One parameter that can be changed is the threshold distance based on the use case intended.

This parameter is set using the `UWB_THRES_DIST` macro. It can be fine-tuned using the `UWB_THRES_DIST_LOWER_BOUND` and `UWB_THRES_DIST_UPPER_BOUND`. These macros are used to implement a logic that works like a hysteresis comparator. This logic is needed because the UWB has a +/- 10 cm precision when ranging over small distances.

```
#define UWB_THRES_DIST (100)
#define UWB_THRES_DIST_LOWER_BOUND (UWB_THRES_DIST - 20)
#define UWB_THRES_DIST_UPPER_BOUND (UWB_THRES_DIST + 20)
```

Another very important parameter is the `lock_action_t`. This parameter is used to implement the logic to decide which AT_command the QN9090 can send over UART to the LPC55S69.

```
typedef enum lock_action
{
    LOCK_DOOR = 0,
    UNLOCK_DOOR,
    KEEP_STATE
} lock_action_t;
```

2.2.5 AT commands from QN9090 to LPC55S69

[Figure 9](#) shows the UWB AT commands flow.

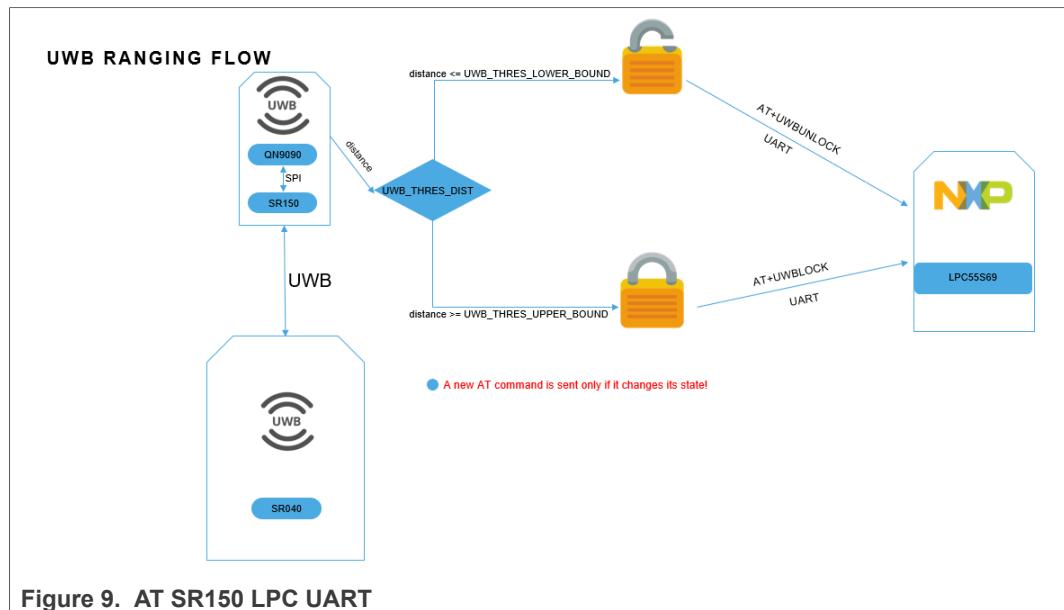


Figure 9. AT SR150 LPC UART

2.2.6 Command API

The actuation logic can be disassembled in two parts. First, the `door_is_locked` flag is used to decide the physical state where the lock should be moved.

Based on it and the lower and upper bound thresholds, a simple logic to either lock, or unlock the smart lock can be implemented. On top of that is a counter logic involved preventing the corner case when the two UWB boards cannot range between them. When there are 15 consecutive iterations with invalid ranging data and the lock is unlocked, the locking command is sent.

If a valid ranging measurement comes during those 15 iterations, the timer gets reset.

Next, when the `Send_To_UART_state` has its value set, one of the `AT+UWBLOCK\r\n` or `AT+UWBUNLOCK\r\n` commands is sent over UART.

To prevent data tangling over UART, computations should be offloaded on the modules to free the LPC55S69.

```
void handle_ranging_data(uint16_t distance)
{
    lock_action_t Send_To_UART_state = KEEP_STATE;

    if (door_is_locked == true)
    {
        if (distance <= UWB_THRES_DIST_LOWER_BOUND)
        {
            Send_To_UART_state = UNLOCK_DOOR;
            door_is_locked = false;
        }
    }
    else if (door_is_locked == false)
    {
        if (distance >= UWB_THRES_DIST_UPPER_BOUND && distance <
INVALID_RANGING_DATA)
        {
            Send_To_UART_state = LOCK_DOOR;
            door_is_locked = true;
        }
        else if (distance == INVALID_RANGING_DATA)
        {
            ++counter;
        }
    }
}
```

```

        if (counter == 15)
        {
            door_is_locked = true;
            counter = 0;
            Send_To_UART_state = LOCK_DOOR;
        }
    }

    if (distance != INVALID_RANGING_DATA)
    {
        counter = 0;
    }

    if (Send_To_UART_state == LOCK_DOOR)
    {
        Serial_Print(gAppSerMgrIf, "AT+UWBLOCK\r\n", gAllowToBlock_d);
    }
    else if (Send_To_UART_state == UNLOCK_DOOR)
    {
        Serial_Print(gAppSerMgrIf, "AT+UWBUNLOCK\r\n", gAllowToBlock_d);
    }
}

```

2.2.6.1 AppCallback and INVALID_RANGING_DATA

Inside this function, the handling logic can be found for `VALID_RANGING_DATA` and `INVALID_RANGING_DATA`. The logic is handled using the status parameter which is `0x00` for valid ranging data and `UWBAPI_STATUS_INVALID_RANGING_DATA` for invalid ranging data. Check `UwbApi_Types.h` for any other status parameters. Furthermore, the parameters that are sent with `handle_ranging_data`, either have the value from `*pData` or a hardcoded value of `0xFFFF` which is seen as `INVALID_RANGING_DATA`.

2.2.6.2 LPC side parsing

On LPC55S69, the logic is based on the `UWBLOCK` and `UWBUNLOCK` commands, which trigger the corresponding action on the motor.

```

g_BLECmdCmp = (volatile uint8_t *)strstr((const char *)buf, (const char *)
"AT+UWBLOCK");
if (g_BLECmdCmp != NULL)
{
    APP_MOTOR_Task(MOTOR_LOCK);
}

g_BLECmdCmp = (volatile uint8_t *)strstr((const char *)buf, (const char *)
"AT+UWBUNLOCK");
if (g_BLECmdCmp != NULL)
{
    APP_MOTOR_Task(MOTOR_UNLOCK);
}

```

2.3 Face recognition

The crossover MCU [i.MX RT117F](#), based on [VIZN3D](#), provides 3D secure face recognition with advanced liveness detection.

Face registration and recognition notifications are sent back and forth through LPC55's serial communication to the [SLN-VIZN3D-IOT Platform](#).

The face registration process and face detection are only done locally on the SLN-VIZN3D-IOT platform.

A description of these commands can be found in the file `sln_at_commands.h`, as given below:

```
typedef enum _sln_at_commands_type
{
    FACEECHO_OK = 0,      /* LPC55 echo to RT117F as CMD received success */
    FACEECHO_FALSE,      /* LPC55 echo to RT117F as CMD received failed */
    FACEREG,             /* Starting Registration / Enroll Face ID 0~n */
    FACEDREG,            /* Starting Deregistration */
    FACERREG,            /* Remote face registration */
    FACEREG_RSP,         /* Response to local face registration */
    FACERREG_RSP,        /* Response to remote face registration */
    FACEDEL,             /* Starting Deregistration face id 0~n */
    FACEDEL_RSP,         /* Deregistration response */
    FACERES,             /* Face Recognition response */
    FACEMODE,            /* Face module enter to sleep mode */
    FACEMODE_CONFIRM,    /* Backup to be used in future */
    ATCOMMAND_ERROR      /* Error in parsing the command */
} sln_at_command_t;
```

2.3.1 Command APIs

Commands are parsed from clear strings using specific method and can be easily added or updated per user needs.

```
static sln_at_command_t Parse_ATCommand(uint8_t *rcv_buff, uint32_t buff_len)
{
    sln_at_command_t command = ATCOMMAND_ERROR;
    uint8_t *command_str;
    uint8_t *result_str;

    strupr((char *)rcv_buff);

    if ((result_str = (uint8_t *)strstr((const char *)rcv_buff, (const char *)
    "AT+")) != NULL)
    {
        command_str = (uint8_t *)strstr((const char *)result_str, (const char *)
        "FACEECHO=");
        if (command_str != NULL)
        {
            result_str = &command_str[strlen("FACEECHO=")];

            if (strncmp((const char *)result_str, "OK", 2) == 0)
            {
                return FACEECHO_OK;
            }

            if (strncmp((const char *)result_str, "FALSE", 5) == 0)
            {
                return FACEECHO_FALSE;
            }

            return ATCOMMAND_ERROR;
        }

        command_str = (uint8_t *)strstr((const char *)result_str, (const char *)
        "FACEREG=");
        if (command_str != NULL)
        {
            return FACEREG;
        }
    }
}
```

Sending AT commands response is similar and uses the `LPUART_RTOS_Send` driver API to forward the specific string messages to the LPC controller.

```
static int Send_ATCommnd_Response(sln_at_command_t command, char *value)
{
    uint32_t size;
```

```

switch (command)
{
    case FACEREG_RSP:
        size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+FACEREG=%s\r\n", value);
        break;

    case FACEREG_RSP:
        size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+FACEREG=%s\r\n", value);
        break;

    case FACEDEL_RSP:
        size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+FACEDEL=%s\r\n", value);
        break;

    case FACERES:
        size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+FACERES=%s\r\n", value);
        break;

    case FACEMODE:
        size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+FACEMODE=%s\r\n", value);
        break;
    default:
        size = -1;
        break;
}

if (size > 0)
{
    LPUART_RTOS_Send(s_LpuartRTOSHandle, (uint8_t *)s_response_buff, size);
}
else
{
    return -1;
}

return 0;
}

```

The `SLN_ATCommands_Task()` processes the commands and uses callback functions to generate internal events which trigger the vision-specific tasks (algorithms).

```

static void SLN_ATCommands_Task()
{
    sln_at_command_t command;
    uint32_t receiverList = 0;
    uint32_t dataLen = 0;

    while(1)
    {
        read_command(AT_COMMANDS_EMPTY_LIMIT, '=');

        /* Add '\0' at the end of the string since it ends with '=' */
        s_commands_buf[s_current_idx] = '\0';
        command = Parse_ATCommand(s_commands_buf, s_current_idx);

        switch (command)
        {
            case FACEREG:

                read_command(AT_COMMANDS_EMPTY_LIMIT, '\r');

                /* Replace the "\r" with \0 so we can use strlen to calculate
the size of the data */
                s_commands_buf[s_current_idx - 1] = '\0';

                receiverList = 1 << kFWKTaskID_VisionAlgo;
                dataLen = (uint32_t)strlen(s_commands_buf);

                if (s_InputDev_ATCommands->cap.callback != NULL)

```

```

        {
            s_ATCommandsEvent.eventBase.eventId =
kEventFaceRecID_AddUser;
            s_ATCommandsEvent.eventBase.respond =
ATCommands_InputDev_Responds;

            if (dataLen > 0)
            {
                s_ATCommandsEvent.addFace.hasName = true;
                strncpy(s_ATCommandsEvent.addFace.name, (const char *)
s_commands_buf, dataLen);
            }
            else
            {
                s_ATCommandsEvent.addFace.hasName = false;
            }

            uint8_t fromISR = __get_IPSR();
            s_InputEvent.inputData = &s_ATCommandsEvent;
            s_InputDev_ATCommands->cap.callback(s_InputDev_ATCommands,
kInputEventID_Recv, receiverList,
                                                    &s_InputEvent, 0,
fromISR);
        }
        break;

case FACEDREG:
    receiverList = 1 << kFWKTaskID_VisionAlgo;

    if (s_InputDev_ATCommands->cap.callback != NULL)
    {
        s_ATCommandsEvent.eventBase.eventId = kEventFaceRecID_DelUser;
        s_ATCommandsEvent.eventBase.respond =
ATCommands_InputDev_Responds;
        s_ATCommandsEvent.delFace.hasName = false;
        s_ATCommandsEvent.delFace.hasID = false;
        uint8_t fromISR = __get_IPSR();
        s_InputEvent.inputData = &s_ATCommandsEvent;
        s_InputDev_ATCommands->cap.callback(s_InputDev_ATCommands,
kInputEventID_Recv, receiverList,
                                                    &s_InputEvent, 0,
fromISR);
    }
    break;
    ...

```

Remark: The callback function API including the parameters is defined in `sln_smart_lock/framework/hal_api/hal_input_dev.h`:

```

/**
 * @brief callback function to notify input manager with an async event
 * @param dev Device structure
 * @param eventId Id of the event that took place
 * @param receiverList List with managers that should be notify
 * @param event Pointer to a event structure.
 * @param size If size is 0 event should be in a persistent memory zone else
the framework will allocate memory for the
 * object Note the message delivery might go slow if the size is too much.
 * @param fromISR True if this operation takes place in an irq, 0 otherwise
 * @return 0 if the operation was successfully
 */
typedef int (*input_dev_callback_t)(const input_dev_t *dev,
                                   input_event_id_t eventId,
                                   unsigned int receiverList,
                                   input_event_t *event,
                                   unsigned int size,
                                   uint8_t fromISR);

```

This function is implemented in the framework module. The framework transfers the input into the AI algorithms and returns the output from the AI algorithms.

The AI algorithms do the detection work and then send back responses that are forwarded to the LPC controller and phone application.

```
void APP_OutputDev_ATCommands_InferCompleteDecode(output_algo_source_t source,
oasis_lite_result_t inferResult)
{
    uint32_t result = inferResult.result;
    char ID[3];

    if (source == kOutputAlgoSource_LPM)
    {
        Send_ATCommnd_Response(FACEMODE, "LP");
    }

    if (inferResult.qualityCheck == kOasisLiteQualityCheck_FakeFace)
    {
        Send_ATCommnd_Response(FACEREG_RSP, "FAKE");
    }

    switch (inferResult.state)
    {
        case kOASISLiteState_Recognition:
            switch (result)
            {
                case kOASISLiteRecognitionResult_Success:
                    itoa(inferResult.face_id, ID, 10);
                    Send_ATCommnd_Response(FACERES, ID);
                    break;
                default:
                    break;
            }
            break;
        case kOASISLiteState_Registration:
            switch (result)
            {
                case kOASISLiteRegistrationResult_Success:
                    itoa(inferResult.face_id, ID, 10);
                    Send_ATCommnd_Response(FACEREG_RSP, ID);
                    break;

                case kOASISLiteRegistrationResult_Timeout:
                    Send_ATCommnd_Response(FACEREG_RSP, "FAIL");
                    break;

                case kOASISLiteRegistrationResult_Duplicated:
                    Send_ATCommnd_Response(FACEREG_RSP, "DUPLICATE");
                    break;

                default:
                    break;
            }
            ...
    }
}
```

2.3.2 Code location

The source code related to the communication between the VIZN and main board can be found in `sln_smart_lock/source/sln_at_commands.c`

The interface between AT commands and framework can be found in `sln_smart_lock/HAL/common/hal_at_commands.c`

2.4 Matter

The [Matter](#) provides a unified, IPv6-based, application layer connectivity standard to connect with the main controller.

To enable Matter capabilities, the Smart Access Platform uses the K32W chip with *Thread* functionality to achieve a host-less/standalone device. Additionally, OpenThread Border Router (OTBR) and Radio Co-Processor (RCP) are needed but not supplied by our default accessories. With these, the user can easily implement more matter commands, as described below. By default, the project provides the commands for lock and unlock.

The Matter project (formerly known as CHIP) is maintained and developed on the official [Github](#) page and is open source. The repository hosts multiple documentations, such as how to create a building environment, how to create a new custom border router, information about the project architecture, and so on.

For the border router, an [Apple Homepod Mini](#) is used to test the application. For more relevant information on how to set up and use it, see *Smart Access Platform Solution User Guide* (document [SAPSUG](#)).

2.4.1 Interaction commands

For device interaction with the Matter host control interface, an AT commands specification is used between the LPC controller and the K32W edge node device, which is connected to the Open Thread network. Smart lock status and operation are available in this way.

[Figure 10](#) shows the communication flow.

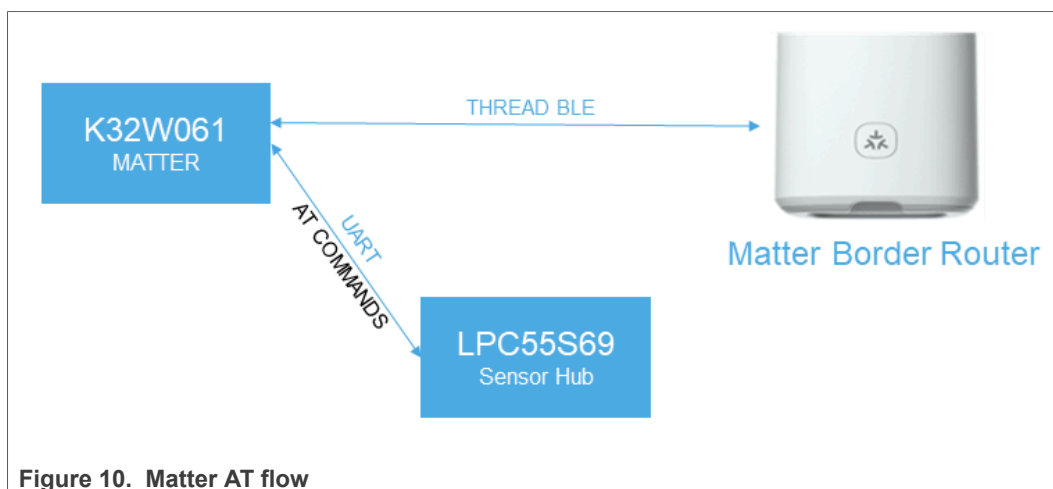


Figure 10. Matter AT flow

A description of the currently implemented commands is described below.

```

typedef enum _sln_at_commands_type
{
    MATTERLOCK,           /* Lock the door */
    MATTERLOCK_OK,        /* Lock successful */
    MATTERLOCK_FAIL,      /* Lock fail */
    MATTERUNLOCK,         /* Unlock the door */
    MATTERUNLOCK_OK,      /* Unlock successful */
    MATTERUNLOCK_FAIL,    /* Unlock fail */
    EXTLOCK,              /* The door was lock from an external source */
    EXTUNLOCK,            /* The door was unlock from an external source */
    MATTERRESET,          /* Factory reset the K32W */
    MATTERADV,            /* Start the BLE advertising */
    ATCOMMAND_ERROR
} sln_at_command_t;
  
```

Additional commands can be added based on the app cluster, for example:

- Unlock with Timeout
- Set/Get User Status
- Set/Get User Type

More commands are considered to be added in the future phase.

2.4.2 Command APIs

Commands are parsed from clear strings and can be easily added or updated as per user needs.

```
static sln_at_command_t Parse_ATCommand(uint8_t *rcv_buff, uint32_t buff_len,
uint8_t **value)
{
    sln_at_command_t command = ATCOMMAND_ERROR;
    uint8_t *command_str;
    uint8_t *result_str;

    strupr((char *)rcv_buff);

    if (strncmp((const char *)rcv_buff, "AT+", 3) == 0)
    {
        command_str = (uint8_t *)strstr((const char *)rcv_buff, (const char
*)"MATTERLOCK=");
        if (command_str != NULL)
        {
            result_str = &command_str[strlen("MATTERLOCK=")];

            if (strncmp((const char *)result_str, "OK", 2) == 0)
            {
                return MATTERLOCK_OK;
            }

            if (strncmp((const char *)result_str, "FAIL", 4) == 0)
            {
                return MATTERLOCK_FAIL;
            }

            return ATCOMMAND_ERROR;
        }
        ...
    }
}
```

Sending AT commands responses is similar and uses the USART_RTOS_Send driver API to forward the specific string messages to the LPC controller.

```
int Send_ATCommnd_Response(sln_at_command_t command)
{
    uint32_t size;

    switch (command)
    {
        case MATTERLOCK:
            size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+MATTERLOCK=\r\n");
            break;

        case MATTERUNLOCK:
            size = snprintf(s_response_buff, AT_COMMANDS_RSP_BUFF_SIZE, "AT
+MATTERUNLOCK=\r\n");
            break;

        default:
            size = -1;
            break;
    }

    if (size > 0)
    {
        if (kStatus_Success !=
```

```

        USART_RTOS_Send(&handle, (uint8_t *)s_response_buff, size))
    {
        return -1;
    }
}
else
{
    return -1;
}

return 0;
}

```

The `SLN_ATCommands_Task()` processes the commands and uses callback functions to trigger specific events.

```

static void SLN_ATCommands_Task()
{
    sln_at_command_t command;

    size_t n = 0;
    uint32_t current_idx = 0;
    uint8_t *value = 0;
    uint32_t receiverList = 0;
    uint32_t dataLen = 0;
    usart_config.srcclk = CLOCK_GetFreq(kCLOCK_Fro32M);
    usart_config.base = AT_COMMANDS_USART;

    NVIC_SetPriority(AT_COMMANDS_USART_IRQn, AT_COMMANDS_USART_NVIC_PRIO);

    if (USART_RTOS_Init(&handle, &t_handle, &usart_config) < 0)
    {
        vTaskSuspend(NULL);
    }

    while(1)
    {
        while (s_lpuart_char != '\r' && s_lpuart_char != '\n')
        {
            USART_RTOS_Receive(&handle, &s_lpuart_char, 1, &n);

            s_commands_buf[current_idx] = s_lpuart_char;
            current_idx++;

            if (current_idx >= AT_COMMANDS_BUFF_SIZE)
            {
                PRINTF("Command buffer overflow\r\n");
                current_idx = 0;
            }
        }
        /* Replace the "\r\n" with \0 so we can use strlen to calculate the
        size of the data */
        s_commands_buf[current_idx - 1] = '\0';
        command = Parse_ATCommand(s_commands_buf, current_idx, &value);

        switch (command)
        {
            ...
            case MATTERRESET:
                ButtonCallback(RESET_BUTTON, RESET_BUTTON_PUSH);
                break;

            case MATTERADV:
                ButtonCallback(BLE_BUTTON, BLE_BUTTON_PUSH);
                break;
            ...
        }
    }
}

```

2.4.3 Code location

The Matter-related sources can be found in `kw-matter/examples/lock-app/k32w/main/sln_at_commands.c`

2.5 Capacitive touch PIN pad

The MCU [KL1x](#) provides PIN pad function based on the capacitive touch sensor interface.

Main controller reads out the touched key event for further handling.

3 APK introduction

3.1 Smart Access Manager

The APK Smart Access Manager runs on the Android (version 8 or upper with BLE support) device. It provides the remote management interface to the Smart Access Platform via Bluetooth Low Energy communication. The APK is developed in [Android Studio](#).

The Smart Access Manager application is a demo reference project. It implements all the communication between user and the board kit, including user management and remote control.

The APK comes with the following out-of-box features:

- Connects to the board to access the features
- Represents the communication channel between user and product
- The application is built in Android Studio and utilizes BLE communication to send and receive information to and from the LPC55
- Maintains the connection with the QN9090 host MCU, sending and receiving AT commands via Bluetooth serial
- Registration functionality via password, face, fingerprint, and NFC
- Device management, user management, and remote control

Use cases include:

- Board scan
- User list
- Register user
- User options
- Change name
- Delete user
- Re-register
- Password unlock

3.1.1 APK software diagram

[Figure 11](#) shows the APK software diagram.

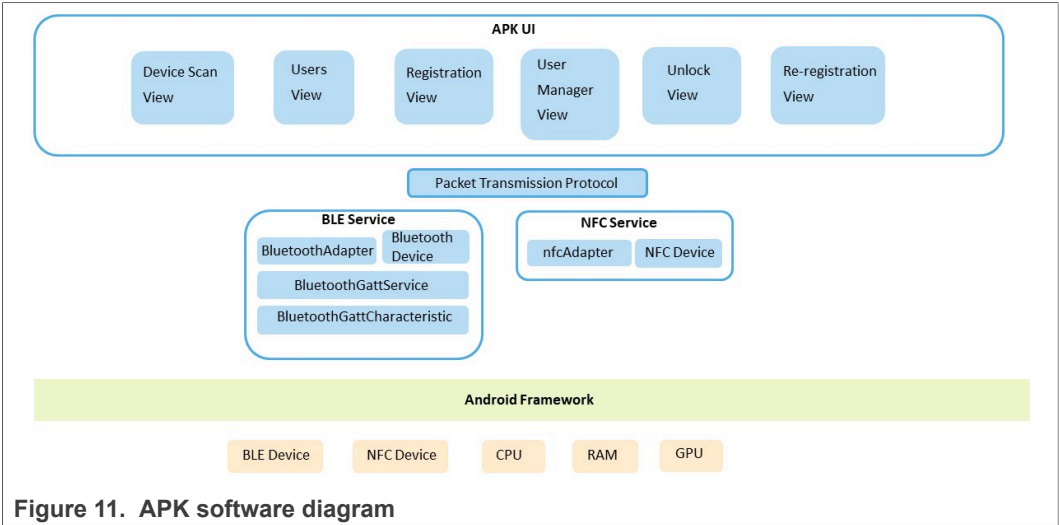


Figure 11. APK software diagram

3.1.2 APK UI flow description

Figure 12, Figure 13, and Figure 14 show the APK UI flow.

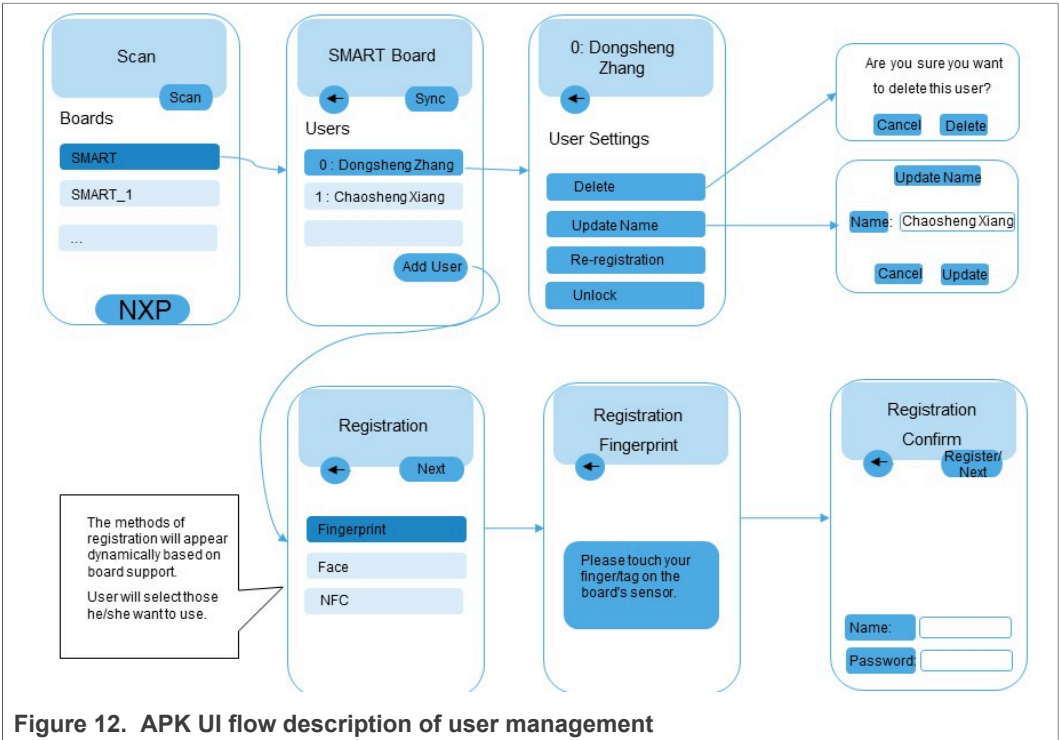


Figure 12. APK UI flow description of user management

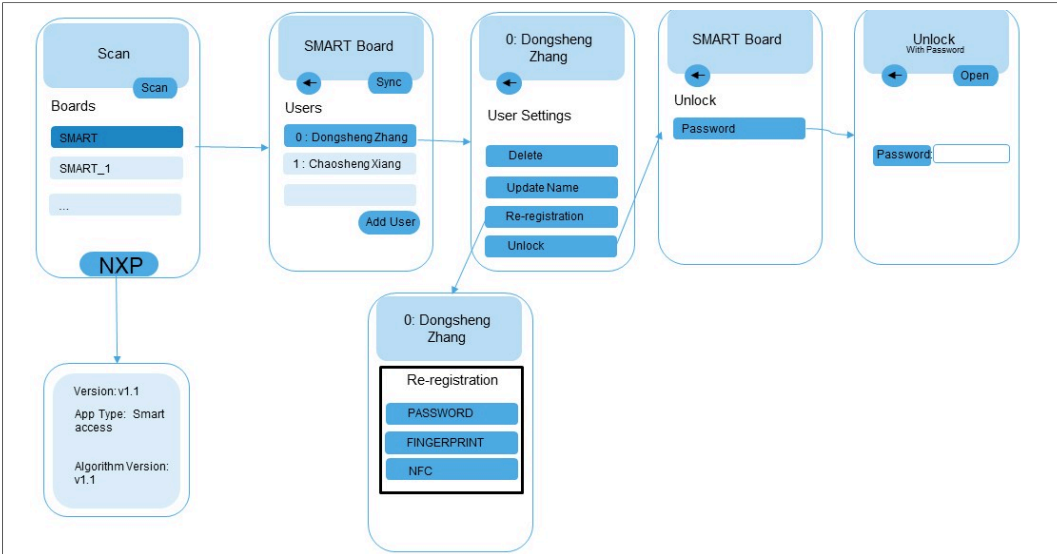


Figure 13. APK UI flow description of user re-registration

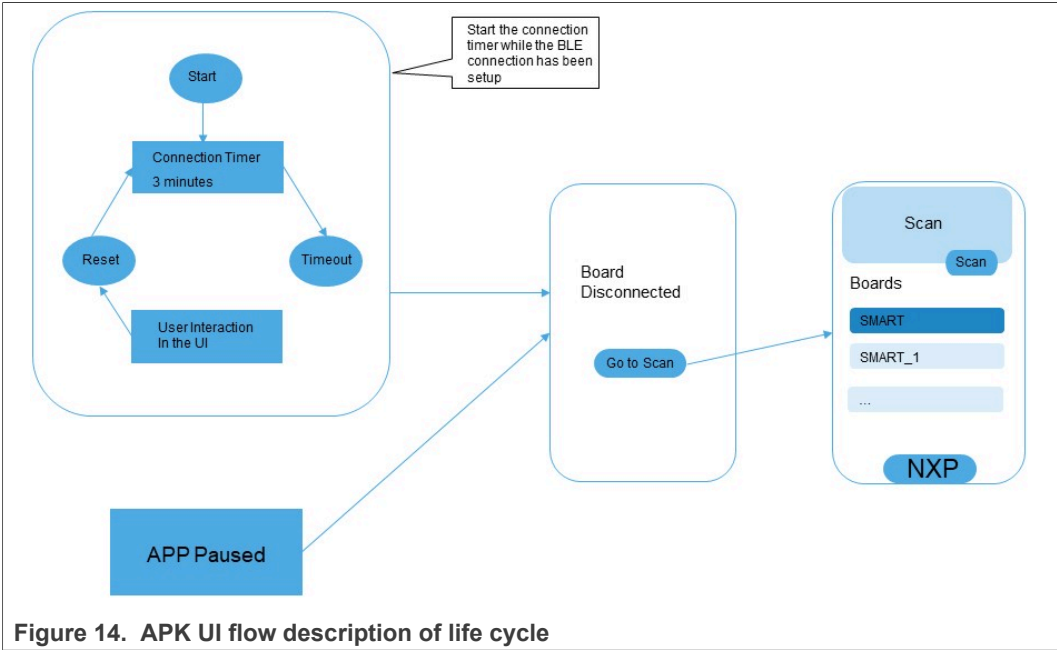


Figure 14. APK UI flow description of life cycle

4 AT commands introduction

The main controller running on the LPC55S69 is interconnected with the other components through serial interfaces. The AT commands based protocol implements the communication among them. The protocol is convenient for the synchronization and control of various events and functionalities.

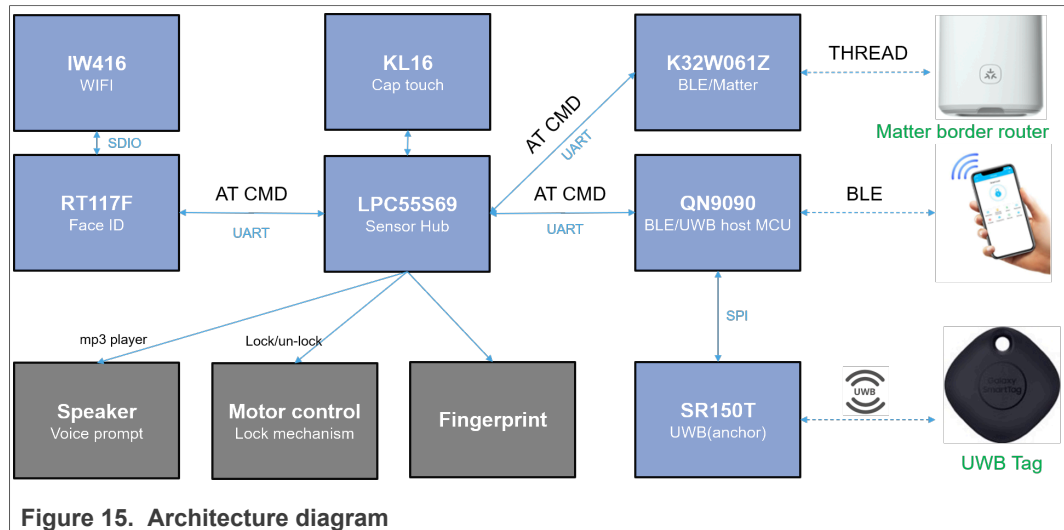


Figure 15. Architecture diagram

Each of these commands is sent in clear over the serial interfaces and processed at their destination by directly parsing the string messages.

Table 4 shows the registration related AT commands.

Table 4. Registration related AT commands

AT commands	Direction	Function
AT+APPTYPE=\r\n	QN9090 > LPC55	Request to get app type of device
AT+APPTYPE="type"\r\n	LPC55 > QN9090	Response to get the app type of device
AT+PWD=name,123456\r\n	QN9090 > LPC55	Create new user with provided name and password
AT+PWD=ID\r\n	LPC55 > QN9090	Create new user successfully
AT+NFC=ID\r\n	QN9090 > LPC55	Enroll NFC Tag
AT+NFC=OK\r\n	LPC55 > QN9090	Enroll NFC successfully
AT+FINGERPRINT=ID\r\n	QN9090 > LPC55	Enroll fingerprint
AT+FINGERPRINT=OK\r\n	LPC55 > QN9090	Enroll fingerprint successfully
AT+UNLOCKPASS=ID,123456\r\n	QN9090 > LPC55	Unlock using the password
AT+UNLOCKPASS=OK\r\n	LPC55 > QN9090	Unlock successfully

Table 5 shows the user management related AT commands.

Table 5. User management related AT commands

AT commands	Direction	Function
AT+GETUSERNO=\r\n	QN9090 > LPC55	Get user count
AT+GETUSERNO=NO\r\n	LPC55 > QN9090	Response of user count
AT+GETINFO=\r\n	QN9090 > LPC55	Get user information
AT+GETINFO="data"\r\n	LPC55 > QN9090	Response of user information
AT+UPDTUSER=ID,name\r\n	QN9090 > LPC55	Update user name

Table 5. User management related AT commands...continued

AT commands	Direction	Function
AT+UPDTUSER=OK\r\n	LPC55 > QN9090	Update user name successfully
AT+UPDTUSERPASS=ID,123456\r\n	QN9090 > LPC55	Update user password
AT+UPDTUSERPASS=OK\r\n	LPC55 > QN9090	Update user password successfully
AT+USERDEL=ID\r\n	QN9090 > LPC55	Delete user
AT+USERDEL=OK\r\n	LPC55 > QN9090	Delete user successfully

Table 6 shows the face recognition related AT commands.

Table 6. Face recognition related AT commands

AT commands	Direction	Function
AT+FACEREG=name\r\n	LPC55 > RT117F	Enroll face
AT+FACEREG=ID\r\n	RT117F > LPC55	Enroll face successfully
AT+FACEDEL=ID\r\n	LPC55 > RT117F	Delete face
AT+FACEDEL=SUCCESS\r\n	RT117F > LPC55	Delete face successfully
AT+FACERES=ID\r\n	RT117F > LPC55	Face recognition successfully

Table 7 shows the Matter and UWB access related AT commands.

Table 7. Matter and UWB access related AT commands

AT commands	Direction	Function
AT+MATTERLOCK=\r\n	K32W0 > LPC55	Matter locks the door
AT+MATTERLOCK=OK\r\n	LPC55 > K32W0	Matter locks the door successfully
AT+MATTERUNLOCK=\r\n	K32W0 > LPC55	Matter unlocks the door
AT+MATTERUNLOCK=OK\r\n	LPC55 > K32W0	Matter unlocks the door successfully
AT+UWBLOCK=\r\n	QN9090 > LPC55	UWB lock the door
AT+UWBUNLOCK=\r\n	QN9090 > LPC55	UWB unlock the door

5 Revision history

Table 8 summarizes the changes done to this document since the initial release.

Table 8. Revision history

Revision number	Date	Substantive changes
1	19 December 2022	Updated content for UWB Nearby Interaction and Matter Apple HomePod support
0	28 April 2022	Initial release

6 Legal information

6.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <http://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

6.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Apple — is a registered trademark of Apple Inc.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Contents

1	Introduction	2
1.1	Smart Access Platform overview	2
1.2	Software development environment	3
2	Boards introduction	4
2.1	Main controller	4
2.1.1	Hardware layout	4
2.1.2	Software work flow	5
2.1.3	Application software diagram	5
2.1.4	Key APIs	5
2.1.5	Memory layout	6
2.2	BLE (Bluetooth Low Energy) and UWB (Ultra-Wideband)	7
2.2.1	Nearby devices prerequisites	8
2.2.2	BLE and UWB commands diagram	8
2.2.3	Demos used	9
2.2.4	Relevant changes for smart lock actuation logic	9
2.2.5	AT commands from QN9090 to LPC55S69	9
2.2.6	Command API	10
2.2.6.1	AppCallback and INVALID_RANGING_ DATA	11
2.2.6.2	LPC side parsing	11
2.3	Face recognition	11
2.3.1	Command APIs	12
2.3.2	Code location	15
2.4	Matter	15
2.4.1	Interaction commands	16
2.4.2	Command APIs	17
2.4.3	Code location	18
2.5	Capacitive touch PIN pad	19
3	APK introduction	19
3.1	Smart Access Manager	19
3.1.1	APK software diagram	19
3.1.2	APK UI flow description	20
4	AT commands introduction	21
5	Revision history	23
6	Legal information	24

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.