WF5000/WF6000 Alphabeam SDK APIs

I&C TECHNOLOGY

www.inctech.co.kr

Version 1.2

2018. 06. 07

	Comment	Date	Author	Approver
1.00	Initial release	2015-06-16	jryu	
1.01	Modified and Added APIs	2015-07-01	jryu	
1.02	Modified P2P Interface	2015-07-06	jryu	
1.03	Added AP Connection API	2015-07-07	jryu	
1.04	Modified P2P Interface and added GMMP Interface	2015.07-15	jryu	
1.05	Added MIB SET/GET functions	2015-07-21	jryu	
1.06	Added Wireless Mode SET/GET functions	2015-07-30	jryu	
1.07	Added AES functions	2015-09-09	jryu	
1.08	Added I2C functions	2015-09-23	jryu	
1.09	Changed name from modem SDK to alphabeam SDK	2015-10-15	jryu	
1.10	Added OS Scheduler Lock / Unlock & Added FTPC functionality	2015-10-22	jryu	
1.11	Added STA join indication at AP mode & Added OS Time Delay functionality	2015-10-27	jryu	
1.12	Added Country code functionality	2015-11-16	jryu	
1.13	Added DHCP Server functionality	2015-11-25	jryu	
1.14	Modified ict_api_uart_change_baudrate() function interface	2015-12-09	jryu	
1.15	Added Modified Utility APIs	2015-12-22	jryu	
1.16	Added DNS Start & Stop APIs	2015-12-29	jryu	
1.17	Added Firmware CRC Validation APIs	2016-01-04	jryu	
1.18	Added UART Init/Open/Close functionality, GIGA IoT/Embedded MQTT functionality	2016-03-09	jryu	
1.19	Added UART Tx Busy functionality.	2016-04-04	jryu	
1.20	Added OneM2M APIs	2017-09-06	cmlee	
1.21	Added Stand-alone(Hardwired) Security Engine APIs	2018-02-01	jhjeong	
1.22	Added WF6000 HW RS485 API	2018-06-07	cylee	

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Overview

This document describes the API library to exploit WF5000/WF6000 module itself. The library is delivered with the intent of providing an easy way to program and configure the modules.
The API library is platform independent and written in simple C language.

The following data type conventions are used.

INT8	signed char	**UINT8**	unsigned char
INT16	signed short	**UINT16**	unsigned short
INT32	signed long	**UINT32**	unsigned long

ICT_FALSE	0	**ICT_TRUE**	!(ICT_FALSE)
ICT_NULL	0		
ICT_OK	0	**ICT_ERR**	(-1)

1.2 Scope

1.3 Definition, acronyms, and abbreviations

- STA Station, a device that has the compatibility to use the IEEE 802.11 protocol
- AP Access Point
- N/A Not Available

2. Alphabeam SDK Architecture

The Alphabeam SDK is one-chip solution to support almost of Wi-Fi functionalities in WF5000/WF6000 module through Alphabeam SDK APIs. All of Wi-Fi functionalities should be provided as a library file. The functionalities could be accessed through the Alphabeam SDK APIs. Application users can make their own source code controlling WF5000/WF6000 module.

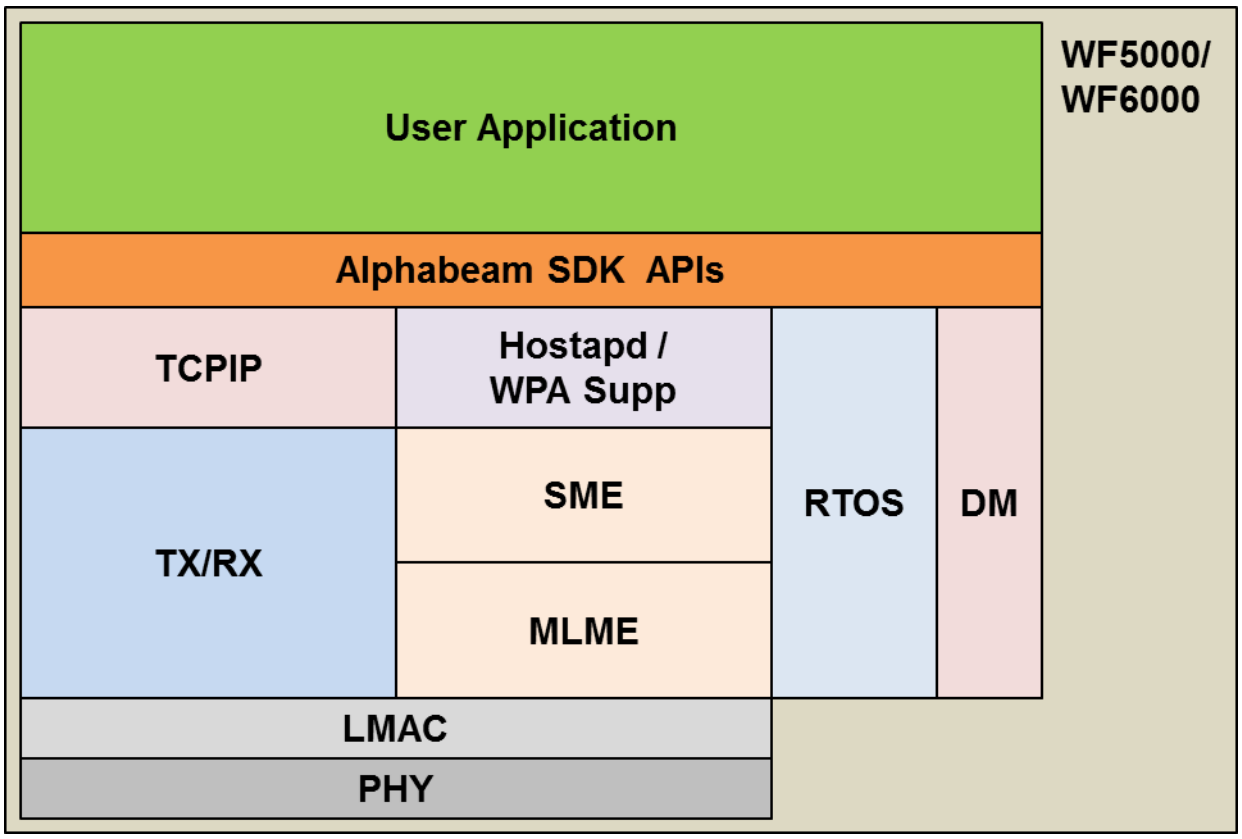


Figure 2-1 Alphabeam SDK Architecture

3. Utility APIs

The Utility APIs are used instead of standard libraries or are used to support convenient tools.

3.1 ICT_MEMSET

The standard library named "memset" is used for ICT_MEMSET.

3.2 ICT_STRLEN

Self-created "ict_api_strlen" is used for ICT_STRLEN.

3.3 ICT_STRLEN_W_SEP

Self-created "ict_api_strlen_w_sep" is used for ICT_STRLEN_W_SEP.
A specific separated value could be used to indicate an end of string.

3.4 ICT_STRCMP

The standard library named "strcmp" is used for ICT_STRCMP.

3.5 ICT_STRNCMP

The standard library named "strncmp" is used for ICT_STRNCMP.

3.6 ICT_STRCASECMP

Self-created "ict_api_strcasecmp" is used for ICT_STRCASECMP.

3.7 ICT_STRNCASECMP

Self-created "ict_api_strncasecmp" is used for ICT_STRNCASECMP.

3.8 ICT_STRSTR

Self-created "ict_api_strstr" is used for ICT_STRSTR.

3.9 ICT_STRNSTR

Self-created "ict_api_strnstr" is used for ICT_STRNSTR.

3.10 ICT_MEMCMP

The standard library named "memcmp" is used for ICT_MEMCMP.

3.11 ICT_STRCPY

The standard library named "strcpy" is used for ICT_STRCPY.

3.12 ICT_STRLCPY

Self-created "ict_api_strlcpy" is used for ICT_STRLCPY.

3.13 ICT_MEMCPY

Self-created "ict_api_memmove" is used for ICT_MEMCPY.

3.14 ICT_MEMMOVE

Self-created "ict_api_memmove" is used for ICT_MEMMOVE.

3.15 ICT_MALLOC

Self-created "ict_api_malloc" is used for ICT_MALLOC.

3.16 ICT_FREE

Self-created "ict_api_mfree" is used for ICT_FREE.

3.17 ICT_ASSERT

Self-created "ict_api_assert" is used for ICT_ASSERT.

3.18 ICT_SPRINTF

Self-created "ict_api_sprintf" is used for ICT_SPRINTF.

3.19 ICT_VSPRINTF

Self-created "ict_api_vsprintf" is used for ICT_VSPRINTF.

3.20 ICT_ATOI

Self-created "ict_api_atoi" is used for ICT_ATOI.

3.21 ICT_STOI

Function convert string to integer value
Prototype BOOL ict_api_str_to_int (char *str, UINT32 *value)
Arguments str is a point to a string to be converted to integer value. value is a point to a integer value which will be returned to your application.
Return TRUE or FALSE

3.22 ICT_HSTOI

Function convert hex string to integer value
Prototype BOOL ict_api_hexStr_to_int (char *str, UINT32 *value)
Arguments str is a point to a hex string to be converted to integer value. value is a point to a integer value which will be returned to your application.
Return TRUE or FALSE

3.23 ICT_DELAY_US

Function delay several micro-seconds to execute next codes.

Prototype void ict_api_delay_us (UINT32 usec)
Arguments usec is a time [the unit of micro-seconds] to be delayed..
Return N/A.

3.24 ICT_DELAY_MS

Function delay several milli-seconds to execute next codes.
Prototype void ict_api_delay_ms (UINT32 msec)
Arguments msec is a time [the unit of milli-seconds] to be delayed.
Return N/A.

4. Linked List APIs

The Linked List APIs are used to create, manage, and delete items of a linked list.

Structure
typedef struct _LIST_ENTRY { struct _LIST_ENTRY * Prev; struct _LIST_ENTRY * Next; } LIST_ENTRY, *PLIST_ENTRY; typedef struct _LIST_TAG { PLIST_ENTRY Head; PLIST_ENTRY Tail; } T_LIST;

4.1 ict_api_initialize_linked_list

Function
initialize a linked list.
Prototype
void ict_api_initialize_linked_list(T_LIST *list)
Arguments list is a point to a linked list which will be initialized.
Return
N/A.

4.2 ict_api_linked_list_empty

Function
check whether a linked list is empty or not.
Prototype
BOOL ict_api_linked_list_empty(T_LIST *list)
Arguments list is a point to a linked list which will be checked.
Return

TRUE or FALSE.

4.3 ict_api_linked_list_empty

Function check whether a linked list is empty or not.
Prototype BOOL ict_api_linked_list_empty(T_LIST *list)
Arguments list is a point to a linked list which will be checked.
Return TRUE or FALSE.

4.4 ict_api_insert_head_list

Function insert an entry into start of a linked list.
Prototype void ict_api_insert_head_list(T_LIST *list, PLIST_ENTRY entry)
Arguments list is a point to a linked list which an entry will be inserted at start of. entry is an entry to be inserted into a linked list.
Return N/A.

4.5 ict_api_insert_tail_list

Function insert an entry into end of a linked list.
Prototype void ict_api_insert_head_list(T_LIST *list, PLIST_ENTRY entry)
Arguments list is a point to a linked list which an entry will be inserted at end of. entry

is an entry to be inserted into a linked list.
Return N/A.

4.6 ict_api_remove_head_list

Function remove and return an entry from start of a linked list.
Prototype PLIST_ENTRY ict_api_remove_head_list(T_LIST *list)
Arguments list is a point to a linked list.
Return an entry removed from start a linked list

4.7 ict_api_remove_tail_list

Function remove and return an entry from end of a linked list.
Prototype PLIST_ENTRY ict_api_remove_tail_list(T_LIST *list)
Arguments list is a point to a linked list.
Return an entry removed from end of a linked list

4.8 ict_api_get_head_list

Function return an entry from start of a linked list.
Prototype PLIST_ENTRY ict_api_get_head_list(T_LIST *list)
Arguments list is a point to a linked list.
Return an entry existing at start of a linked list

4.9 ict_api_get_tail_list

Function return an entry from end of a linked list.
Prototype PLIST_ENTRY ict_api_get_tail_list(T_LIST *list)
Arguments list is a point to a linked list.
Return an entry existing at end of a linked list

4.10 ict_api_insert_front_list

Function insert new entry into previous of specific entry of a linked list.
Prototype void ict_api_insert_front_list(T_LIST *pList, PLIST_ENTRY pEntry, PLIST_ENTRY pNewEntry)
Arguments list is a point to a linked list. pEntry is specific entry of a linked list. pNewEntry is new entry to be inserted into previous of specific entry.
Return N/A.

4.11 ict_api_remove_list_entry

Function remove an entry from a linked list.
Prototype void ict_api_remove_list_entry(T_LIST * pList, PLIST_ENTRY pEntry)
Arguments list is a point to a linked list. pEntry

is an entry of a linked list to be removed.
Return N/A.

4.12 ict_api_move_tail_all_list

Function move items of a linked list to end of the other linked list.
Prototype void ict_api_move_tail_all_list(T_LIST *destList, T_LIST *srcList)
Arguments destList is a point to a destination linked list to be moved. srcList is a point to a source linked list to be moved.
Return N/A.

4.13 ict_api_get_count_from_list

Function return total number of items included in a linked list.
Prototype UINT32 ict_api_get_count_from_list(T_LIST *list)
Arguments list is a point to a linked list.
Return Total number of items included in a linked list.

5. DM Shell and Debug APIs

The DM Shell and Debug APIs are used to handle a DM shell based environment. (Refer to "DM Shell Programming Guide.pdf" and "DM Shell Command Guide.pdf" documents for DM Shell)

5.1 ict_api_dm_shell_get_token

Function parses a word from a string.
Prototype char *ict_api_dm_shell_get_token (char *pInStr, char *pOutToken)
Arguments pInStr is a point to a string which will be parsed. pOutToken is a point to a word which will be returned to your application.
Return is a point to a string without a word which is returned by pOutToken.

5.2 ict_api_dm_shell_send_cmd_handler

Function is used to send DM shell commands
Prototype INT32 ict_api_dm_shell_send_cmd_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf is a pointer to buffer to store DM shell command strings. len is the length of buffer.
Return 0 : Success / Otherwise, Failure

5.3 ict_api_debug_get_timestamp

Function is used to get TIMESTAMP
Prototype UINT32 ict_api_debug_get_timestamp (void)

Arguments N/A.
Return TIMESTAMP

5.4 ict_api_debug_set_poll_mode

Function is used to enable or disable polling mode of Tx UART
Prototype void ict_api_debug_set_poll_mode (BOOL b_poll)
Arguments b_poll 0 : transmit UART data in IDLE task 1 : transmit UART data immediately
Return N/A.

5.5 ict_api_debug_get_poll_mode

Function is used to get polling mode
Prototype UINT32 ict_api_debug_get_poll_mode (void)
Arguments N/A.
Return polling mode

5.6 ict_api_debug_set_md_level

Function change and set display level on MDP window of DM.
Prototype void ict_api_debug_set_md_level (UINT32 level)
Arguments level is a display level on MDP windows.

DBG_LVL_CLEAR : There is not displayed any strings in MDP window. DBG_LVL_TRACE : DBG_LVL_ERR and DBG_LVL_WARN are displayed with DBG_LVL_TRACE. DBG_LVL_NONE : All of debug level are displayed in MDP window. DBG_LVL_CLEAR (0) DBG_LVL_ERR (1) DBG_LVL_WARN (2) DBG_LVL_TRACE (3) DBG_LVL_INFO (4) DBG_LVL_NONE (5)
Return N/A.

5.7 ict_api_debug_dm_enable

Function enable or disable DM when a UART port for DM is used for user application.
Prototype INT32 ict_api_debug_dm_enable(BOOL enable, void (*rxCallBack)(void), UINT32 baudRate)
Arguments enable is a Boolean value to enable or disable DM. void (*rxCallBack)(void) is a function pointer to set call back function for user application when enable is set to FALSE. baudRate is integer value to set baudrate for user application when enable is set to FALSE.
Return 0 : Success / otherwise : Failure

5.8 APP_SH_PRINTF

Function is used instead of PRINTF() in dm shell functions.
Prototype void ict_api_debug_dm_hif_printf (char *fmt,...)
Arguments
Return N/A.

5.9 ICT_TRACEx (x = 0, 1, 2, 3, or 4)

Function is used to print a debug information on FDP window of DM.
Prototype void ict_api_debug_trace0 (const char *str) void ict_api_debug_trace1 (const char *str, UINT32 val1) void ict_api_debug_trace2 (const char *str, UINT32 val1, UINT32 val2) void ict_api_debug_trace3 (const char *str, UINT32 val1, UINT32 val2, UINT32 val3) void ict_api_debug_trace4 (const char *str, UINT32 val1, UINT32 val2, UINT32 val3, UINT32 val4)
Arguments str is a string [Maximum length = 7] to distinguish the debugging point from other point. val1 / val2 / val3 / val4 are variables [unsigned integer] to check specific debugging value.
Return N/A.

6. OS and Task APIs

The OS (Operation System) and Task APIs are used to create and handle tasks that is used by user. the maximum two of user tasks could be supported.

Structure

```
typedef struct _MAC_EVENT_TAG
{
    LIST_ENTRY      link;

    UINT32          from;   // Source Task ID
    UINT32          to;     // Destination Task ID
    UINT32          code;   // Command ID to be used to primitive
    UINT32          len;    // Parameter Length
    UINT8 *         buf;    // Parameter
} T_MAC_EVENT;

typedef struct
{
    INT32 (*func_task_primitive_send)(UINT32 from, UINT32 to, UINT32 code, UINT32 length,
UINT8 *p_buf);
    INT32 (*func_task_event_send)(UINT32 to, UINT32 event);
} ICT_ST_FUNC_PTR_T;

typedef enum
{
    TASK_ID_API_MIN = 12,
#if defined(HOST_STDA_CM_INTERWORKING)
    TASK_ID_API_CM = TASK_ID_API_MIN,
    TASK_ID_API_UAPP,
#endif
#if defined(FEATURE_PICOC_SUPP)
    TASK_ID_API_INTP = TASK_ID_API_MIN,
#endif
    TASK_ID_API_MAX
} TASK_ID_API;

typedef enum
{
```

```
        TASK_PRI_API_MIN = 8,
    #if defined(HOST_STDA_CM_INTERWORKING)
        TASK_PRI_API_CM = TASK_PRI_API_MIN,
        TASK_PRI_API_UAPP,
    #endif
    #if defined(FEATURE_PICOC_SUPP)
        TASK_PRI_API_INTP = TASK_PRI_API_MIN,
    #endif
        TASK_PRI_API_MAX
    } TASK_PRI_API;

    typedef enum
    {
    ICT_HIF_CMD_GROUP = 0x1000,
        ICT_HIF_CMD_ST_MAC_ADDR_IND = ICT_HIF_CMD_GROUP,
        ICT_HIF_CMD_ST_SCAN_IND,
        ICT_HIF_CMD_ST_SCAN_RST_IND,
        ICT_HIF_CMD_ST_JOIN_IND,
        ICT_HIF_CMD_ST_DISCONNECTED_IND,
        ICT_HIF_CMD_ST_AP_START_IND,
        ICT_HIF_CMD_ST_AP_STOP_IND,
        ICT_HIF_CMD_ST_STA_ASSOCIATED_IND,
        ICT_HIF_CMD_ST_STA_DISASSOCIATED_IND,
        ICT_HIF_CMD_ST_DM_IND,   /* Not Supported */
        ICT_HIF_CMD_ST_SOCKET_IND,
        ICT_HIF_CMD_ST_TCP_DISCONNECT_IND,   // Only for TCP Server...
        ICT_HIF_CMD_ST_WPS_IND,   /* Not Supported */
        ICT_HIF_CMD_ST_P2P_DEVICE_FOUND_IND,
        ICT_HIF_CMD_ST_P2P_DEVICE_LOST_IND,
        ICT_HIF_CMD_ST_P2P_GO_NEG_IND,
        ICT_HIF_CMD_ST_P2P_RESULT_IND,
        ICT_HIF_CMD_ST_NETWORK_INFO_IND,
        ICT_HIF_CMD_ST_DEVICE_READY_IND,
        ICT_HIF_CMD_ST_DNSQUERY_IND,
        ICT_HIF_CMD_ST_HTTP_CONTROL_IND,
        ICT_HIF_CMD_ST_HTTP_BODY_IND,
        ICT_HIF_CMD_ST_UPNP_EXTIP_IND,
        ICT_HIF_CMD_ST_UPNP_EXTIP_ERROR_IND,
        ICT_HIF_CMD_ST_UPNP_ADDPORTMAPPING_IND,
```

```
    ICT_HIF_CMD_ST_UPNP_ADDPORTMAPPING_ERROR_IND,
    ICT_HIF_CMD_ST_UPNP_DELPORTMAPPING_IND,
    ICT_HIF_CMD_ST_UPNP_DELPORTMAPPING_ERROR_IND,
    ICT_HIF_CMD_ST_DDNS_GETIPADDR_IND,
    ICT_HIF_CMD_ST_DDNS_GETIPADDR_ERROR_IND,
    ICT_HIF_CMD_ST_DDNS_UPDATE_IND,
    ICT_HIF_CMD_ST_DDNS_UPDATE_ERROR_IND,
    ICT_HIF_CMD_ST_OTA_VERSION_IND,
    ICT_HIF_CMD_ST_OTA_VERSION_FIN_IND,
    ICT_HIF_CMD_ST_OTA_UPDATE_IND,
    ICT_HIF_CMD_ST_OTA_UPDATE_FIN_IND,
    ICT_HIF_CMD_ST_HWPS_IND,
    ICT_HIF_CMD_ST_PING_REPLY_IND,
    ICT_HIF_CMD_ST_PING_RESULT_IND,
    ICT_HIF_CMD_ST_UDAP_RESULT_IND,
    ICT_HIF_CMD_ST_SNTP_RESPONSE_IND,
    ICT_HIF_CMD_ST_SNTP_GET_ERROR_IND,
    ICT_HIF_CMD_ST_ALLJOYN_IND,
    ICT_HIF_CMD_ST_XMODEM_IND,
    ICT_HIF_CMD_ST_GMMP_IND,
    ICT_HIF_CMD_ST_GMMP_RECV_DATA_IND,
    ICT_HIF_CMD_ST_MQTT_PUB_IND,
    ICT_HIF_CMD_ST_MQTT_SUB_IND,
    ICT_HIF_CMD_ST_MQTT_PUB_RECV_IND,
    ICT_HIF_CMD_ST_MQTT_SUB_RECV_IND,
    ICT_HIF_CMD_ST_COAP_MSG_IND,
    ICT_HIF_CMD_ST_COAP_MSG_RECV,
    ICT_HIF_CMD_ST_SEP20_EVENT_IND,
    ICT_HIF_CMD_ST_TCP_SSL_IND,
    ICT_HIF_CMD_ST_TCP_SSL_RECV_IND,
    ICT_HIF_CMD_ST_TCP_SSL_CLOSE_IND,
    ICT_HIF_CMD_ST_TCP_SSL_SVR_IND,
    ICT_HIF_CMD_ST_TCP_SSL_SVR_RECV_IND,
    ICT_HIF_CMD_ST_TCP_SSL_SVR_CLOSE_IND,
    ICT_HIF_CMD_ST_TCP_SSL_SVR_ACCEPTED_IND,
    ICT_HIF_CMD_ST_GCM_RECV_IND,
    ICT_HIF_CMD_ST_VENDOR_USER_IE_IND,
    ICT_HIF_CMD_ST_VENDOR_USER_DATA,
    ICT_HIF_CMD_MAX,
```

```
        ICT_HIF_DATA_MIN = 0x1100,
        ICT_HIF_DATA_RX = ICT_HIF_DATA_MIN,
        ICT_HIF_DATA_MAX,

        ICT_HIT_AT_MIN = 0x1120,
        ICT_HIT_REQ_AT_SHELL = ICT_HIT_AT_MIN,
        ICT_HIT_AT_MAX
    } ICT_API_PRIMITIVE_CODE;
```

6.1 ict_api_os_flag_create

Function create an event flag group
Prototype OS_FLAG_GRP *ict_api_os_flag_create(OS_FLAGS flags, UINT8 *err)
Arguments flags contains the initial value to store in the event flag group. err is a pointer to an error code which will be returned to your application: 0 : Success / Otherwise, Failure
Return a pointer to an event flag group or a NULL pointer if no more groups are available.

6.2 ict_api_os_flag_post

Function is called to set or clear some bits in an event flag group.
Prototype void ict_api_os_flag_post(OS_FLAG_GRP *pgrp, OS_FLAGS flags, UINT8 opt, UINT8 *err)
Arguments pgrp is a pointer to the desired event flag group. flags will set the corresponding bit in the event flag group. [bit 0 ~ bit 31] Bit 0 is used to set primitive event and bit 1 is used to set timer event. EVENT_MASK_PRIMITIVE (0x00000001) EVENT_MASK_TIMER (0x00000002)

Bit 2 to bit 31 are used for user application. opt Not used err is a pointer to an error code which will be returned to your application: 0 : Success / Otherwise, Failure
Return N/A.

6.3 ict_api_os_flag_pend

Function is called to wait for a combination of bits to be set in an event flag group.
Prototype OS_FLAGS ict_api_os_flag_pend (OS_FLAG_GRP *pgrp, OS_FLAGS flags, UINT8 wait_type, UINT16 timeout, UINT8 *err)
Arguments pgrp is a pointer to the desired event flag group. flags will set the corresponding bit in the event flag group. [bit 0 ~ bit 31] Bit 0 is used to set primitive event and bit 1 is used to set timer event. EVENT_MASK_PRIMITIVE (0x00000001) EVENT_MASK_TIMER (0x00000002) Bit 2 to bit 31 are used for user application. wait_type Not used Timeout is an optional timeout that your task will wait for the desired bit combination. is the unit of OS tick [10 ms]. err is a pointer to an error code which will be returned to your application: 0 : Success / Otherwise, Failure
Return a pointer to an event flag group or a NULL pointer if no more groups are available.

6.4 ict_api_os_primitive_queue_create

Function

create a primitive queue to be used for communication between tasks.
Prototype void ict_api_os_primitive_queue_create (T_LIST *primitve_queue)
Arguments primitive_queue is a pointer to a liked list consisting of head and tail.
Return N/A.

6.5 ict_api_os_primitive_queue_empty

Function check primitive queue of a task whether it is empty or not.
Prototype BOOL ict_api_os_primitive_queue_empty (T_LIST *primitive_queue)
Arguments primitive_queue is a pointer to a liked list consisting of head and tail.
Return TRUE or FALSE.

6.6 ict_api_os_task_create_ext

Function create a task
Prototype void ict_api_os_task_create_ext(void (*task)(void *p_arg), void *p_arg, OS_STK *ptos, UINT8 prio, UINT16 id, OS_STK *pbos, UINT32 stk_size, void *pext, UINT16 opt)
Arguments task is a pointer to the task's code. p_arg is a pointer to an optional data area which can be used to pass parameters to the task when the task first executes. ptos is a pointer to the task's top of stack. prio is the task's priority. id

is the task's ID. pbos is a pointer to the task's bottom of stack. stk_size is the size of the stack in number of elements. pext is a pointer to a user supplied memory location which is used as a TCB extension. opt contains additional information (or options) about the behavior of the task.
Return 0 : Success / Otherwise, Failure.

6.7 ict_api_os_task_name_set

Function assign a name to a task.
Prototype void ict_api_os_task_name_set (UINT8 prio, UINT8 *pname, UINT8 *err)
Arguments prio is the priority of the task that you want the assign a name to. pname is a pointer to an ASCII string [Max length = 7] that contains the name of the task. err is a pointer to an error code that can contain one of the following values: 0 : Success / Otherwise, Failure
Return N/A.

6.8 ict_api_task_enqueue_primitive

Function en-queue a primitive to be sent from one task to the other task.
Prototype void ict_api_task_enqueue_primitive(T_LIST *primitive_queue, T_MAC_EVENT *p_mac_event)
Arguments primitive_queue is a pointer to the primitive queue of destined task to be sent. p_mac_event

includes primitive information to be parsed in destined task.
Return N/A.

6.9 ict_api_task_dequeue_primitive

Function de-queue a primitive from primitive queue when the queue is not empty.
Prototype void ict_api_os_task_name_set (UINT8 prio, UINT8 *pname, UINT8 *err)
Arguments primitive_queue is a pointer to primitive queue which is used for de-queuing.
Return primitive information included by en-queuing procedure.

6.10 ict_api_task_allocate_primitive

Function allocate a primitive buffer from primitive buffer list.
Prototype void ict_api_os_task_name_set (UINT8 prio, UINT8 *pname, UINT8 *err)
Arguments N/A.
Return allocated primitive buffer

6.11 ict_api_task_free_primitive

Function free allocated primitive buffer..
Prototype void ict_api_task_free_primitive(T_MAC_EVENT *p_mac_event)
Arguments p_mac_event is a pointer to allocated buffer.
Return N/A.

6.12 ict_api_task_send_func_ptr_set

Function register function pointers to send primitives and send events.
Prototype void ict_api_task_send_func_ptr_set(ICT_ST_FUNC_PTR_T *p_st_func_ptr)
Arguments p_st_func_ptr is a pointer to a primitive function and an event function.
Return N/A.

6.13 ict_api_task_primitive_send

Function send a primitive from one task to the other task.
Prototype INT32 ict_api_task_primitive_send(UINT32 from, UINT32 to, UINT32 code, UINT32 length, UINT8 *p_buf)
Arguments from is a task ID sending a primitive. to is a task ID to be received a primitive. code is a unique primitive ID which is distinguished from other primitives. length is length of data to be sent to a destined task. p_buf is a pointer to data to be sent to a destined task.
Return TRUE or FALSE

6.14 ict_api_task_event_send

Function send a event from one task to the other task.
Prototype INT32 ict_api_task_event_send(UINT32 to, UINT32 event)

Arguments to is a task ID to be received a primitive. event is a unique event ID which is distinguished from other events. [Bit 0 ~ Bit 31]
Return TRUE or FALSE.

6.15 ict_api_task_primitive_receive

Function parses primitives which is sent from other tasks to the task.
Prototype INT32 ict_api_task_primitive_receive(T_MAC_EVENT * p_event)
Arguments p_event contains a primitive information to be parsed and executed the destined task.
Return TRUE or FALSE.

6.16 ict_api_task_ready

Function indicates ready status of the task.
Prototype void ict_api_task_ready(void)
Arguments N/A.
Return N/A.

6.17 ict_api_os_sched_lock

Function used to lock OS scheduling.
Prototype void ict_api_os_sched_lock (void)
Arguments

N/A.
Return N/A.

6.18 ict_api_os_sched_unlock

Function used to unlock OS scheduling
Prototype void ict_api_os_sched_unlock (void)
Arguments N/A.
Return N/A.

6.19 ict_api_os_time_delay

Function used to set OS time delay
Prototype void ict_api_os_time_delay (UINT16 ticks)
Arguments N/A.
Return N/A.

7. OS SW timer and HW External timer APIs

The OS SW timer and HW External timer APIs are used to create and handle timers that is used by user. the minimum interval of OS SW timer equals to the unit of OS tick [10 ms]. the interval should be set to multiple of 10. the minimum interval of HW External timer equals to 1mili-second. the interval should be set to multiple of 1.

Structure

```
typedef       void (*SW_TIMER_CALLBACK)(void *pTmrCntx, UINT32 timerId);

typedef struct
{
    SW_TIMER_CALLBACK timerCb;
    UINT32    interval;
    UINT32    expireCnt;
    UINT8     bEnable;
    UINT8     bPeriod;
    UINT8     bTrigger;
} SW_TIMER;

typedef struct
{
    SW_TIMER      *pList;
    OS_TMR        *pOsTimer;
    OS_FLAG_GRP *pRcvTaskFlag;
    UINT32         baseInterval;
    UINT8          noOfTimer;
    UINT8          bOsTimerStarted;
    UINT8          noOfActTimer;
} SW_TIMER_CNTX;
```

7.1 ict_api_sw_timer_init

Function

initialize a SW timer group.

Prototype

```
void ict_api_sw_timer_init   (SW_TIMER_CNTX *pTimerCntx, SW_TIMER *pTimerList,
                              UINT32 noOfTimer, UINT32 baseInterval, OS_FLAG_GRP *pRcvTaskFlag)
```

Arguments

pTimerCntx is a pointer to SW timer context information to manage SW timers on a task. pTimerList is a pointer to a SW timer list of a task. noOfTimer indicates the number of SW timers in a SW timer list of a task. baseInterval indicates the base interval of SW timers of a task. [the unit of 10ms] pRcvTaskFlag is a pointer to allocated event flag group of a task.
Return N/A.

7.2 ict_api_sw_timer_start

Function is called to start a SW timer.
Prototype UINT32 ict_api_sw_timer_start (SW_TIMER_CNTX *pTimerCntx, UINT32 timerId, UINT32 bPeriod, UINT32 interval, SW_TIMER_CALLBACK timerCb)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list. bPeriod indicates whether a timer is executed periodically or not. [TRUE or FALSE] interval is expired time for a timer. timerCb is a callback function to be executed when a timer is expired.
Return TRUE or FALSE.

7.3 ict_api_sw_timer_stop

Function is called to stop a SW timer.
Prototype

UINT32 ict_api_sw_timer_stop (SW_TIMER_CNTX *pTimerCntx, UINT32 timerId)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list.
Return TRUE or FALSE.

7.4 ict_api_sw_timer_task_proc

Function inspects whether a SW timer is expired or not in a task. executes a SW timer which is expired.
Prototype void ict_api_sw_timer_task_proc (SW_TIMER_CNTX *pTimerCntx)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task.
Return N/A.

7.5 ict_api_sw_timer_get_status

Function returns the status of a SW timer whether the SW timer is enabled or not.
Prototype UINT32 ict_api_sw_timer_get_status(SW_TIMER_CNTX *pTimerCntx, UINT32 timerId)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list.
Return TRUE or FALSE.

7.6 ict_api_ext_hw_timer_start

Function

is called to start a HW timer. (GP Timer 3 is used for the external HW timer.)
Prototype void ict_api_ext_hw_timer_start(UINT32 timer_ms)
Arguments timer_ms is an interval of HW timer. [the minimum interval of a HW timer = 1ms]
Return N/A.

7.7 ict_api_ext_hw_timer_stop

Function is called to stop a HW timer.
Prototype void ict_api_ext_hw_timer_stop(void)
Arguments None
Return N/A.

7.8 ict_api_ext_sw_timer_init

Function initialize an extended SW timer group.
Prototype void ict_api_ext_sw_timer_init (SW_TIMER_CNTX *pTimerCntx, SW_TIMER *pTimerList, UINT32 noOfTimer, OS_FLAG_GRP *pRcvTaskFlag)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. pTimerList is a pointer to an timer list of a task. noOfTimer indicates the number of timers in a timer list of a task. pRcvTaskFlag is a pointer to allocated event flag group of a task. priority is the priority value of a task.
Return

N/A.

7.9 ict_api_ext_sw_timer_start

Function is called to start an extended SW timer.
Prototype UINT32 ict_api_ext_sw_timer_start (SW_TIMER_CNTX *pTimerCntx, UINT32 timerId, UINT32 bPeriod, UINT32 interval, SW_TIMER_CALLBACK timerCb)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list. bPeriod indicates whether a timer is executed periodically or not. [TRUE or FALSE] interval is expired time for a timer. timerCb is a callback function to be executed when a timer is expired.
Return TRUE or FALSE.

7.10 ict_api_ext_sw_timer_stop

Function is called to stop an extended SW timer.
Prototype UINT32 ict_api_ext_sw_timer_stop (SW_TIMER_CNTX *pTimerCntx, UINT32 timerId)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list.
Return TRUE or FALSE.

7.11 ict_api_ext_sw_timer_task_proc

Function inspects whether an extended SW timer is expired or not in a task. executes an extended SW timer which is expired.
Prototype void ict_api_ext_sw_timer_task_proc (SW_TIMER_CNTX *pTimerCntx)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task.
Return N/A.

7.12 ict_api_ext_sw_timer_get_status

Function returns the status of an extended SW timer whether the extended SW timer is enabled or not.
Prototype UINT32 ict_api_ext_sw_timer_get_status(SW_TIMER_CNTX *pTimerCntx, UINT32 timerId)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list.
Return TRUE or FALSE.

7.13 ict_api_ext_sw_timer_get_expire_cnt

Function returns the decremented expire count of an extended SW timer.
Prototype UINT32 ict_api_ext_sw_timer_get_expire_cnt(SW_TIMER_CNTX *pTimerCntx, UINT32 timerId)
Arguments pTimerCntx is a pointer to timer context information to manage timers on a task. timerId is an unique timer ID for a timer in a timer list.
Return TRUE or FALSE.

8. System and Peripheral related APIs

8.1 System related APIs

8.1.1 ict_api_sys_boot_reset

Function reset and reboot WF5000/WF6000 modem.
Prototype void ict_api_sys_boot_reset(UINT32 reset_opt)
Arguments reset_opt 0 : IRAM Reset / 1 : FALSH Reset
Return N/A.

8.1.2 ict_api_sys_get_fw_download_state

Function returns the download states of WF5000/WF6000 firmware.
Prototype INT32 ict_api_sys_get_fw_download_state (void)
Arguments N/A.
Return ICT_DOWNLOAD_IDLE (0) ICT_DOWNLOAD_FROM_WEB (1) ICT_DOWNLOAD_FROM_DM (2)

8.2 GPIO related APIs

8.2.1 ict_api_sys_gpio0_set_pad

Function set a GPIO pin whether it is used for GPIO or not.
Prototype void ict_api_sys_gpio0_set_pad(UINT8 gpio, BOOL b_val)
Arguments gpio

is a pin number tied with a GPIO. [GPIO 0 ~ GPIO 15] enable indicates whether a GPIO pin will be used for GPIO or not. [TRUE or FALSE]
Return N/A.

8.2.2 ict_api_sys_gpio0_get_pad

Function returns the status of all GPIO pins whether these are used for GPIO or not.
Prototype UINT16 ict_api_sys_gpio0_get_pad(void)
Arguments N/A.
Return the status of all GPIO pins whether these are used for GPIO or not.

8.2.3 ict_api_sys_gpio0_set_direction

Function set a direction of a GPIO pin whether it is used as input pin or output pin.
Prototype void ict_api_sys_gpio0_set_direction(UINT8 gpio, UINT16 dir)
Arguments gpio is a pin number tied with a GPIO. [GPIO 0 ~ GPIO 15] dir indicates whether a GPIO pin will be used as input pin or output pin. TRUE : used as output pin. [e.g. LED control] FALSE : used as input pin. [e.g. external button]
Return N/A.

8.2.4 ict_api_sys_gpio0_get_direction

Function returns the status of all GPIO pins whether these are used as input pin or not.
Prototype UINT16 ict_api_sys_gpio0_get_direction(void)
Arguments

N/A.
Return the status of all GPIO pins whether these are used as input pin or not.

8.2.5 ict_api_sys_gpio0_set_output

Function set a status of a GPIO pin when the pin is used as output pin.
Prototype void ict_api_sys_gpio0_set_output (UINT8 gpio, UINT16 output)
Arguments gpio is a pin number of GPIO. [GPIO 0 ~ GPIO 15] output indicates a status of a GPIO pin when the pin is used as output pin. [1 : High / 0 : Low]
Return N/A.

8.2.6 ict_api_sys_gpio0_get_output

Function returns a status of a GPIO pin when the pin is used as output pin.
Prototype BOOL ict_api_sys_gpio0_get_output(UINT8 gpio)
Arguments gpio is a pin number of GPIO. [GPIO 0 ~ GPIO 15]
Return the status of a GPIO pin when the pin is used as output pin. [1 : High / 0 : Low]

8.2.7 ict_api_sys_gpio0_get_input

Function returns a status of a GPIO pin when the pin is used as input pin.
Prototype BOOL ict_api_sys_gpio0_get_input(UINT8 gpio)
Arguments gpio is a pin number of GPIO. [GPIO 0 ~ GPIO 15]

Return the status of a GPIO pin when the pin is used as input pin. [1 : High / 0 : Low]

8.3 UART related APIs

8.3.1 ict_api_uart_init

Function initialize an UART port.
Prototype void ict_api_uart_init(UINT32 port)
Arguments port is a UART port [UART 0 ~ UART 2]
Return N/A.

8.3.2 ict_api_uart_open

Function open an UART port.
Prototype void ict_api_uart_open(UINT32 port)
Arguments port is a UART port [UART 0 ~ UART 2]
Return N/A.

8.3.3 ict_api_uart_close

Function close an UART port.
Prototype void ict_api_uart_close(UINT32 port)
Arguments port is a UART port [UART 0 ~ UART 2]
Return

N/A.

8.3.4 ict_api_uart_reg_rx_callback

Function registers Rx UART callback function.
Prototype void ict_api_uart_reg_rx_callback(UINT32 port, void (*rxCallBack)(void))
Arguments port is a UART port [UART 0 ~ UART 2] void (*rxCallBack)(void) is a pointer to UART callback function.
Return N/A.

8.3.5 ict_api_sys_gpio0_get_pad

Function change a BAUDRATE of a port.
Prototype void ict_api_uart_change_baudrate(UINT32 port, UINT32 baudRate, UINT16 lcr_val)
Arguments port is a UART port [UART 0 ~ UART 2] baudRate is a BAUDRATE which will be changed. lcr_val is a UARTLCR register value will be changed.
Return N/A.

Bit	Name	Description	Default	Type
15:8	-	Reserved.	-	-
7	SPS	Stick Parity Select. When bits 1, 2 and 7 of the UARTLCR register are set to 1, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set to 1 and bit 2 is set to 0, the parity bit is transmitted and checked as a 1.	1'b0	RW
6:5	WLEN	Word Length. This bits indicate the number of data bits transmitted or received in a frame as follows: 2'b11 = 8 bits 2'b10 = 7 bits 2'b01 = 6 bits 2'b00 = 5 bits.	2'h0	RW
4	FEN	Enable FIFOs. If this bit is set to 1, the transmit and receive FIFOs are enabled (FIFO mode). When cleared to 0, the FIFOs are disabled (character mode).	1'b0	RW
3	STP2	Two Stop Bits Select. If this bit is set to 1, two stop bits are transmitted at the end of frame.	1'b0	RW
2	EPS	Even Parity Select. If this bit is set to 1, even parity generation and check is performed during transmission when the bits 1 and	1'b0	RW
		7 are set to 1, or 0-parity generation and check is performed during transmission when the bit 1 is set to 1 and bit 7 is set to 0. If this bit is cleared, odd parity generation and check is performed during transmission when the bits 1 and 7 are set to 1, or 1-parity generation and check is performed during transmission when the bit 1 is set to 1 and bit 7 is set to 0.		
1	PEN	Parity Enable. If this bit is set to 1, parity generation and check is enabled. Otherwise, parity check is disabled and no parity bit is added to the end of frame.	1'b0	RW
0	BRK	Send Break. If this bit is set to 1, a low-level is continuously output on the UARTTXD output after completing transmission of the current character. For the proper execution of the break command, the software must be set this bit for at least two complete frames.	1'b0	RW

Figure 8-1 UARTLCR register

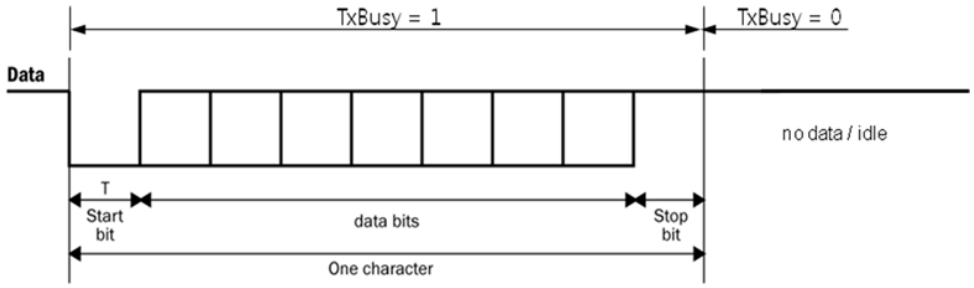
8.3.6 ict_api_uart_is_rx_empty

Function
returns the status whether Rx UART path is empty or not.
Prototype
INT32 ict_api_uart_is_rx_empty(UINT32 port)

Arguments port is a UART port [UART 0 ~ UART 2]
Return -1 : Error / 0 : Rx UART is not empty. / Otherwise, Rx UART is empty.

8.3.7 ict_api_uart_is_tx_busy

Function returns the status whether Tx UART path is busy or not using UARTFR register status.
Prototype INT32 ict_api_uart_is_tx_busy(UINT32 port)
Arguments port is a UART port [UART 0 ~ UART 2]
Return -1 : Error / 0 : Tx UART is not busy. / Otherwise, Tx UART is busy.



8.3.8 ict_api_uart_gets

Function receives strings from Rx UART path.
Prototype INT32 ict_api_uart_gets(UINT32 port, char *buff, UINT32 maxLen)
Arguments port is a UART port [UART 0 ~ UART 2] buff is a pointer buffer to store received data from Rx UART path. maxLen is the maximum length of the buffer to store received data from Rx UART path.
Return

-1 : Error / Otherwise, size of received data

8.3.9 ict_api_uart_getc

Function receives a character from Rx UART path.
Prototype UINT8 ict_api_uart_getc(UINT32 port)
Arguments port is a UART port [UART 0 ~ UART 2]
Return received character from Rx UART path.

8.3.10 ict_api_uart_direct_send_w_size

Function send a string or a character through Tx UART path.
Prototype void ict_api_uart_direct_send_w_size(UINT32 port, UINT8* rsp_ptr, UINT32 size)
Arguments port is a UART port [UART 0 ~ UART 2] str is a pointer to a charter buffer or a string buffer to send through Tx UART path. size is total length of a character or a string to send through Tx UART path.
Return N/A.

8.3.11 ict_api_uart_set_rs485_tx_enable_pin

Function set a GPIO pin as the pin for RS485 communication.
Prototype void ict_api_uart_set_rs485_tx_enable_pin(UINT32 port, UINT8 gpio)
Arguments port is a UART port [UART 0 ~ UART 2] gpio

is a pin number of GPIO. [GPIO 0 ~ GPIO 15] If Tx traffic is available GPIO=High, otherwise, GPIO=Low
Return N/A.

8.3.12 ict_api_uart_set_rs485_delay_time

Function Set to the pre-delay and post-delay of TX_ENABLE pin when data transmits for RS485.
Prototype void ict_api_uart_set_rs485_delay_time (UINT32 port, UINT32 pre_us, UINT32 post_us);
Arguments port is a UART port [UART 0 ~ UART 2] pre_us (microsecond) is a time that can be set before the first data transmission from the time TX ENABLE pin is started for the first time high post_us (microsecond) is a time to set the time until the transition to the low of TX_ENABLE pin since the last data transmission
Return N/A.

8.3.13 ict_api_uart_set_rs485_enable

Function Enable UART to HW RS485 mode and set to the pre-delay and post-delay of TX turn-around time and the polarity of TX_EN pin. Only WF6000 supported feature.
Prototype UINT8 ict_api_uart_set_rs485_delay_time (UINT32 port, UINT8 preDelayBit, UINT8 postDelayBit, UINT8 polarity);
Arguments port is a UART port [UART 1 ~ UART 2] preDelayBit

4'b0000 : no operation 4'b0001 ~ 4'b1111 : in the unit of baud rate bit count postDelayBit 4'b0000 : no operation 4'b0001 ~ 4'b1111 : in the unit of baud rate bit count
Return TRUE or FALSE.

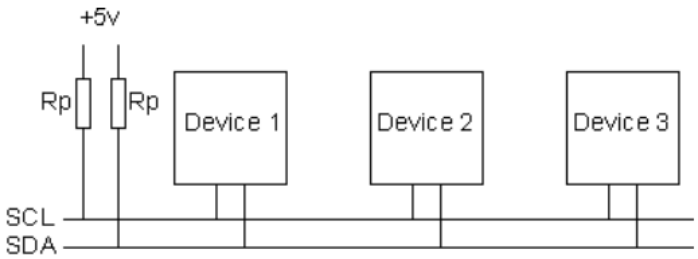
8.3.14 ict_api_send_uart_traffic_handler

Function indicates the number of UART packets to WF5000/WF6000 when HW power save is enabled.
Prototype INT32 ict_api_send_uart_traffic_handler(UINT32 uart_traffic)
Arguments uart_traffic is the number of UART packets
Return TRUE or FALSE.

8.4 I2C related APIs

Reference site : http://www.robot-electronics.co.uk/i2c-tutorial

The physical I2C bus



The value of the registors (Rp) = 1.8k ~ 47k ohms
common values = 1.8k, 4.7k, or 10k
recommended values = 1.8k

8.4.1 ict_api_i2c_init

Function initialize I2C process
Prototype void ict_api_i2c_init (UINT32 scl, UINT32 sda, UINT32 delay)
Arguments scl is a GPIO port number for SCL line. sda is a GPIO port number for SDA line. delay is a delay value between a line pulse and the other line pulse. (Default value = 2)
Return N/A.

8.4.2 ict_api_i2c_start

Function generate I2C start sequence.
Prototype void ict_api_i2c_start (void)
Arguments N/A.
Return N/A.

8.4.3 ict_api_i2c_stop

Function generate I2C stop sequence.
Prototype void ict_api_I2C_stop (void)
Arguments N/A.
Return N/A.

8.4.4 ict_api_i2c_restart

Function generate I2C stop sequence and I2C start sequence.

Prototype void ict_api_i2c_restart (void)
Arguments N/A.
Return N/A.

8.4.5 ict_api_i2c_send_byte

Function send a byte including address, register number, or data.
Prototype void ict_api_i2c_send_byte (UINT8 data)
Arguments data is a value of address, register number, or data
Return N/A.

8.4.6 ict_api_i2c_get_byte

Function get a byte from a I2C slave device.
Prototype UINT8 ict_api_i2c_get_byte (void)
Arguments N/A.
Return a receive byte from a I2C slave device.

8.4.7 ict_api_i2c_get_bytes

Function get several bytes from a I2C slave device.
Prototype void ict_api_i2c_get_bytes (UINT8 no, UINT8 *data)
Arguments no is a number of receiving bytes. [0<no<=8]

data is the data that is received from a I2C slave device.
Return N/A.

9. Serial Flash related APIs

WF5000/WF6000 supports serial flash (SF) command APIs to read and write directly to serial flash sectors. WF5000/WF6000 also supports File System (FS) APIs to store and manage hierarchically files. However, These APIs are not recommended to used by application users. NV memory APIs could be used instead of SF commend APIs and FS APIs.

9.1 Serial Flash commend APIs

9.1.1 ict_api_flash_cmd_read_address

9.1.2 ict_api_flash_cmd_erase

9.1.3 ict_api_flash_cmd_write_multi

9.1.4 ict_api_flash_cmd_read_crc

Function read CRC values for IRAM, DRAM, and Serial Flash Area.
Prototype BOOL ict_api_flash_cmd_read_crc(UINT32 bank_id, UINT16 *crc16_iram, UINT16 *crc16_dram, UINT16 *crc16_sf)
Arguments bank_id is a Serial Flash Bank ID. [1 or 2] crc16_iram is the CRC value for IRAM. crc16_dram is the CRC value for DRAM. crc16_sf_code is the CRC value for Serial Flash Area.
Return TRUE or FALSE.

9.2 File System APIs

9.2.1 ict_api_fs_scan_files

9.2.2 ict_api_fs_disk_free_space

9.2.3 ict_api_fs_write_file

9.2.4 ict_api_fs_read_file_size

9.2.5 ict_api_fs_read_file

9.2.6 ict_api_fs_mkdir

9.2.7 ict_api_fs_remove

9.3 NV memory APIs

Structure

```
#if defined(FEATURE_NV_SYSTEM)
typedef enum
{
#if defined(FEATURE_USE_NV_CM_DATA)
    ICT_CM_NV_ITEM_DATA_LEN,          // 2 bytes
    ICT_CM_NV_ITEM_DATA,              // 1024 bytes
#endif
    ICT_NV_ITEM_CM_MAX
} ICT_CM_NV_ITEM_ID;
#endif
```

9.3.1 ict_api_nv_rebuild

Function

rebuild the NV memory of WF5000/WF6000.

Prototype

void ict_api_nv_rebuild(UINT32 b_init)

Arguments

b_init

1 : Factory Reset / 0 : maintains the value of the previous state.

Return

N/A.

9.3.2 ict_api_nv_write

Function

write data in an user data area of NV memory.

Prototype BOOL ict_api_nv_write (ICT_CM_NV_ITEM_ID nv_id, void *data, UINT32 size)
Arguments nv_id ICT_CM_NV_ITEM_DATA_LEN : the length of data in ICT_CM_NV_ITEM_DATA ICT_CM_NV_ITEM_DATA : the actual data data is a pointer to buffer to be stored in NV memory. size the length of data parameter.
Return TRUE or FALSE.

9.3.3 ict_api_nv_read

Function read data in an user data area of NV memory.
Prototype BOOL ict_api_nv_read (ICT_CM_NV_ITEM_ID nv_id, void *data, UINT32 size)
Arguments nv_id ICT_CM_NV_ITEM_DATA_LEN : the length of data in ICT_CM_NV_ITEM_DATA ICT_CM_NV_ITEM_DATA : the actual data data is a pointer to buffer to be stored in NV memory. size the length of data parameter.
Return TRUE or FALSE.

9.3.4 ict_api_nv_set_apnet

Function set the values of network configurations of AP in NV memory.
Prototype BOOL ict_api_nv_set_apnet (UINT8 *ip, UINT8 *subnet, UINT8 *gateway, UINT8 *lease_ip_min, UINT8 *lease_ip_max)
Arguments ip

is an IP address of AP itself. subnet is a subnet mask of AP itself. gateway is a gateway of AP itself. lease_ip_min is a minimum IP address of lease IP. lease_ip_max is a maximum IP address of lease IP.
Return TRUE or FALSE.

9.3.5 ict_api_nv_get_apnet

Function get the values of network configurations of AP in NV memory.
Prototype BOOL ict_api_nv_set_apnet (UINT8 *ip, UINT8 *subnet, UINT8 *gateway, UINT8 *lease_ip_min, UINT8 *lease_ip_max)
Arguments ip is an IP address of AP itself. subnet is a subnet mask of AP itself. gateway is a gateway of AP itself. lease_ip_min is a minimum IP address of lease IP. lease_ip_max is a maximum IP address of lease IP.
Return TRUE or FALSE.

9.3.6 ict_api_nv_set_httpd_server_port

Function set the port number of a HTTPD server in NV memory.
Prototype BOOL ict_api_nv_set_httpd_server_port(UINT16 port)

Arguments port the port number of a HTTPD server in NV memory.
Return TRUE or FALSE.

9.3.7 ict_api_nv_get_httpd_server_port

Function get the port number of a HTTPD server in NV memory.
Prototype BOOL ict_api_nv_get_httpd_server_port(UINT16 port)
Arguments port the port number of a HTTPD server in NV memory.
Return TRUE or FALSE.

9.3.8 ict_api_nv_set_country_code

Function set the country code of WF5000/WF6000.
Prototype BOOL ict_api_nv_set_country_code(UINT8 *country_code)
Arguments country code the country code of WF5000/WF6000.
Return TRUE or FALSE.

9.3.9 ict_api_nv_get_country_code

Function get the country code of WF5000/WF6000.
Prototype BOOL ict_api_nv_get_country_code(UINT8 *country_code)
Arguments country code the country code of WF5000/WF6000.

Return TRUE or FALSE.

9.3.10 ict_api_nv_set_roam_rssi

Function set the roaming RSSI threshold value of WF5000/WF6000.
Prototype BOOL ict_api_nv_set_roam_rssi(UINT8 *roam_rssi)
Arguments roam_rssi the roaming RSSI threshold value.
Return TRUE or FALSE.

9.3.11 ict_api_nv_get_roam_rssi

Function get the roaming RSSI threshold value of WF5000/WF6000.
Prototype BOOL ict_api_nv_get_roam_rssi(UINT8 *roam_rssi)
Arguments roam_rssi the roaming RSSI threshold value.
Return TRUE or FALSE.

9.3.12 ict_api_nv_set_sntp_svr_addr

Function set the SNTP Server Address.
Prototype BOOL ict_api_nv_set_sntp_svr_addr(UINT8 *svr_addr, UINT32 len)
Arguments svr_addr the SNTP Server Address.
Return TRUE or FALSE.

9.3.13 ict_api_nv_set_sntp_time_offset

Function set the SNTP time offset.
Prototype BOOL ict_api_nv_set_sntp_time_offset(INT16 time_offset)
Arguments time_offset the SNTP time offset.
Return TRUE or FALSE.

9.3.14 ict_api_nv_set_mac_addr

Function set the mac address of WF6000.
Prototype BOOL ict_api_nv_set_mac_addr (UINT8 *mac_addr)
Arguments mac address the 6 bytes mac address of WF6000. (ex. 0x847207010203)
Return TRUE or FALSE.

9.3.15 ict_api_nv_get_mac_addr

Function get the mac address of WF6000.
Prototype BOOL ict_api_nv_get_mac_addr (UINT8 *mac_addr)
Arguments mac address the 6 bytes mac address of WF6000. (ex. 0x847207010203)
Return TRUE or FALSE.

10. MIB and STA APIs

10.1 MIB APIs

Structure

```
typedef enum WIFI_CFG_TAG
{
    WIFI_CFG_SSID,
    WIFI_CFG_CHANNEL,
    WIFI_CFG_NETWORK_MODE,
    WIFI_CFG_ENCRYPT_PROTOCOL,
    WIFI_CFG_PAIRWISE_CIPHER,
    WIFI_CFG_GROUP_CIPHER,
    WIFI_CFG_WEP_KEY,
    WIFI_CFG_WPA_PSK,
    WIFI_CFG_KEY_IDX,
    WIFI_CFG_ENTR_TYPE,
    WIFI_CFG_ENTR_ID,
    WIFI_CFG_ENTR_PASSWD,
    WIFI_CFG_CACERT_FILE,
    WIFI_CFG_CERT_FILE,
    WIFI_CFG_KEY_FILE,
    WIFI_CFG_SN,
    WIFI_CFG_P2P_TYPE,
    WIFI_CFG_P2P_METHOD,
    WIFI_CFG_P2P_PIN,
    WIFI_CFG_MNG_MAX,

    WIFI_CFG_VERSION = 0x20,
    WIFI_CFG_MAC_ADDR,
    WIFI_CFG_BSSID,
    WIFI_CFG_FREQ,
    WIFI_CFG_BAUD_RATE,
    WIFI_CFG_WIFI_CONN_STATUS,
    WIFI_CFG_BAS_MNG_MAX,

    WIFI_CFG_RTS_THRESHOLD = 0x40,
    WIFI_CFG_CTS_THRESHOLD,
    WIFI_CFG_FRAG_THRESHOLD,
```

```
        WIFI_CFG_BEACON_INTERVAL,
        WIFI_CFG_RF_MNG_MAX,

        WIFI_CFG_IP_ADDR = 0x60,
        WIFI_CFG_SUBNET_MASK,
        WIFI_CFG_GATEWAY_ADDR,
        WIFI_CFG_DNS,
        WIFI_CFG_IP_TYPE,
        WIFI_CFG_DHCP_LEASE_IP,
        WIFI_CFG_SVC_PORT,
        WIFI_CFG_UDAP_PORT,
        WIFI_CFG_RT_NET_STATUS,
        WIFI_CFG_RT_IP,
        WIFI_CFG_RT_PORT,
        WIFI_CFG_NET_MNG_MAX,

        WIFI_CFG_RSSI = 0x70,
        WIFI_CFG_ROAM_RSSI,
        WIFI_CFG_STAT_MNG_MAX,

        WIFI_CFG_DATA_MAX,

        WIFI_CFG_SPECIFIC = 0xd0,
        WIFI_CFG_MAX
    } T_WIFI_CFG;
```

10.1.1 ict_api_mac_mib_get_wifi_cfg_ext

Function get Wi-Fi configuration information as string value from MIB.
Prototype void ict_api_mac_mib_get_wifi_cfg_ext(T_WIFI_CFG cmd, void *p_param, UINT32 *p_len)
Arguments type select string type or data type. (default value = 0 and don't change it.) 0 : string type for Wi-Fi configuration information 1 : data type for Wi-Fi configuration information cmd refer to T_WIFI_CFG structure.

p_param is a pointer to data buffer to store Wi-Fi configuration information from MIB. p_len is a pointer to length of data buffer.
Return N/A.

10.1.2 ict_api_mac_mib_set_wifi_cfg_ext

Function set Wi-Fi configuration information as string value from MIB.
Prototype void ict_api_mac_mib_set_wifi_cfg_ext(T_WIFI_CFG cmd, void *p_param, UINT32 *p_len)
Arguments cmd refer to T_WIFI_CFG structure. p_param is a pointer to data buffer to store Wi-Fi configuration information to MIB. p_len is a pointer to length of data buffer.
Return N/A.

10.1.3 ict_api_mac_mib_get_wifi_cfg

Function get Wi-Fi configuration information as data value from MIB.
Prototype UINT32 ict_api_mac_mib_get_wifi_cfg(T_WIFI_CFG cmd, void *p_param, UINT8 *p_len)
Arguments cmd refer to T_WIFI_CFG structure. p_param is a pointer to data buffer to store Wi-Fi configuration information from MIB. p_len is a pointer to length of data buffer.
Return N/A.

10.1.4 ict_api_mac_mib_set_wifi_cfg

Function
set Wi-Fi configuration information as data value from MIB.
Prototype
void ict_api_mac_mib_set_wifi_cfg(T_WIFI_CFG cmd, void *p_param, UINT8 len)
Arguments cmd refer to T_WIFI_CFG structure. p_param is a pointer to data buffer to store Wi-Fi configuration information to MIB. len is the length of data buffer.
Return
N/A.

10.2 STA related APIs

Structure
typedef struct{ UINT32 _11b_rx_sensitivity; UINT32 _11n_rx_sensitivity; UINT32 static_data_rates; } ICT_ST_TRAFFIC_INFO_T;

10.2.1 ict_api_sta_get_traffic_info

Function
get information of Rx sensitivities and static data rates.
Prototype
INT32 ict_api_sta_get_traffic_info(ICT_ST_TRAFFIC_INFO_T *traffic_info)
Arguments traffic_info refer to ICT_ST_TRAFFIC_INFO_T structure.
Return
0 : Success / -1 : Failure

10.2.2 ict_api_sta_get_rx_rssi

Function

get information of RSSI value for the MAC address of a specific STA.
Prototype INT32 ict_api_sta_get_rx_rssi(UINT8 *p_mac_addr)
Arguments p_mac_addr is the MAC address of a specific STA.
Return the status of all GPIO pins whether these are used for GPIO or not.

10.2.3 ict_api_sta_set_tx_pwr_decrement (optional)

Function set TX power decrement level about how much lower than default TX power.
Prototype INT32 ict_api_sta_set_tx_pwr_decrement (UINT32 dec_tx_pwr_dB)
Arguments dec_tx_pwr_dB is the TX power decrement level. [0<= dec_tx_pwr_dB <= 10 in unit of dB scale] If the value is set to 0, then default TX power is used to send frames. If the value is set to 10, then (default TX power - 10 dB) is used to send frames.
Return 0 : Success / -1 : Failure

10.2.4 ict_api_sta_get_tx_pwr_decrement (optional)

Function get TX power decrement level.
Prototype INT32 ict_api_sta_get_tx_pwr_decrement (void)
Arguments N/A.
Return -1 : Failure / Otherwise, the value of TX power decrement level.

10.2.5 ict_api_sta_set_antenna_type

Function set the antenna type of WF5000/WF6000 in NV memory and Variable.
Prototype

BOOL ict_api_sta_set_antenna_type(UINT16 antenna_type)
Arguments port the antenna type of WF5000/WF6000 in NV memory and Variable.
Return TRUE or FALSE.

10.2.6 ict_api_sta_get_antenna_type

Function get the antenna type of WF5000/WF6000 in NV memory and Variable.
Prototype BOOL ict_api_sta_get_antenna_type(UINT16 antenna_type)
Arguments port the antenna type of WF5000/WF6000 in NV memory and Variable.
Return TRUE or FALSE.

10.2.7 ict_api_sta_get_p2p_config

Function get the p2p configuration of WF5000/WF6000 in Variable.
Prototype UINT8 *ict_api_sta_get_p2p_config (void)
Arguments N/A.
Return is a pointer to P2P configuration.

10.2.8 ict_api_sta_get_p2p_pin

Function get the p2p PIN number of WF5000/WF6000 in Variable.
Prototype UINT8 *ict_api_sta_get_p2p_pin (void)
Arguments N/A.
Return

is a pointer to P2P PIN number.

10.2.9 ict_api_sta_ddns_rsp_set (optional)

Function set the DDNS information received from External IP Indication to local buffer in modem side.
Prototype void ict_api_sta_ddns_rsp_set(ICT_ST_DDNS_IND *ddns_ind)
Arguments ddns_ind
Return N/A.

10.2.10 ict_api_sta_mqtt_get (optional)

Structure
typedef struct PACKED { #if defined (FEATURE_MQTT_SUPP) UINT8 server_ip[32]; UINT16 port; UINT8 ssl; UINT8 user_name[32]; UINT8 password[32]; UINT8 pub_topic[32]; UINT8 sub_topic[32]; UINT8 message[150]; } XTENSA_PACKED T_NMS_MQTT_MIB;

Function get the MQTT MIB.
Prototype T_NMS_MQTT_MIB *ict_api_sta_mqtt_get(void)
Arguments N/A.
Return the pointer to "T_NMS_MQTT_MIB" structure.

10.2.11 ict_api_sta_mqtt_set (optional)

Function set the MQTT MIB.
Prototype INT32 ict_api_sta_mqtt_set(UINT32 type, UINT8 *buf, UINT32 buf_len)
Arguments type 0 : Server IP or URL (ex : 192.168.0.1 or www.mqtt.com) 1 : Port 2 : Supported SSL (0 : None / 1 : TLS1.0) 3 : Message (N/A) 4 : User name 5 : Password 6 : Publisher topic 7 : Subscriber topic buf is the pointer to each type. buf_len is the length of buf.
Return 0 : Success / -1 : Failure

10.2.12 ict_api_sta_gmmp_get (optional)

Structure
typedef struct PACKED { #if defined (FEATURE_GMMP_SUPP) UINT8 server_ip[32]; UINT16 port; UINT8 enc_enable; UINT8 enc_type; UINT8 enc_key[32]; UINT8 domain_code[16]; UINT8 manufacture_id[16]; UINT8 auth_id[16]; UINT8 auth_key[16]; UINT8 gw_id[16];

```
        UINT32 rep_peroid;
        UINT32 rep_offset;
        UINT32 resp_timeout;
        UINT32 hb_period;
        UINT8   dev_type[16];
        UINT8   model_id[32];
        UINT8   autopair_enable;
        UINT8   enc_key_length;
    } XTENSA_PACKED T_NMS_GMMP_MIB;
```

Function get the GMMP MIB.
Prototype T_NMS_GMMP_MIB *ict_api_sta_gmmp_get(void)
Arguments N/A.
Return the pointer to "T_NMS_GMMP_MIB" structure.

10.2.13 ict_api_sta_gmmp_set (optional)

Function set the GMMP MIB.
Prototype INT32 ict_api_sta_gmmp_set(UINT32 type, UINT8 *buf, UINT32 buf_len)
Arguments type 0 : Server IP or URL (ex : 192.168.0.1 or www.gmmp.com) 1 : Port 2 : Domain Code or Service ID 3 : Manufacture Id 4 : Authentication Id (MAC Address or S/N) 5 : N/A. 6 : Enable Auto Pairing (0 : Disabled / 1 : Enabled) 7 : N/A. 8 : Enable Encryption (0 : Disabled / 1 : Enabled) 9 : Encryption Algorithm (0 : AES 128 / 1 : AES 192 / 2 : AES 256 / 3 : SEED 128 / 4: SEED 256) 10 : N/A.

11 : N/A. 12 : N/A. 13 : N/A. 14 : Device Type 15 : Model Id buf is the pointer to each type. buf_len is the length of buf.
Return 0 : Success / -1 : Failure

10.2.14 ict_api_sta_sep20_get (optional)

Structure

```
typedef struct PACKED
{
#if defined (FEATURE_SEP20_SUPP)
    UINT8    device_name[64];
    UINT64   sfdi;
    UINT8    lfdi[40+1];
    UINT32   pin;
    UINT32   device_category;
    UINT8    xml_type;
    UINT8    enable_v6;
    UINT8    enable_tls;
    UINT8    enable_xmdns;
    UINT8    enable_reg;
    UINT8    enable_time_sync;
    UINT8    enable_drlc;
    UINT8    enable_tp;
    UINT8    server_name[64];
    UINT8    server_ip[32];
    UINT16   server_port;
    UINT16   server_https_port;
    UINT8    server_level[10];
    UINT8    server_dcap[64];
#endif
} XTENSA_PACKED T_NMS_SEP20_MIB;
```

Function get the SEP2.0 MIB.
Prototype T_NMS_SEP20_MIB *ict_api_sta_sep20_get(void)
Arguments N/A.
Return the pointer to "T_NMS_SEP20_MIB" structure.

10.2.15 ict_api_sta_sep20_set (optional)

Function set the SEP2.0 MIB.
Prototype INT32 ict_api_sta_sep20_set(UINT32 type, UINT8 *buf, UINT32 buf_len)
Arguments type 0 : WF5000/WF6000 Device name for SEP2.0 1 : Device SFDI 2 : Device LFDI 3 : PIN number 4 : XML encoding type (0 : XML / 1 : EXI) 5 : Device category (Bitmap indicating the categories of this device) 6 : SEP2.0 Server name 7 : Server IP address 8 : Server Port address 9 : Server HTTPS Port address 10 : Server level (Preferred schema and extensibility level indication - ex : +S0) 11 : Server DCAP URI (Server Device Capability URI - ex : /dcap) 12 : IPv6 enable (0 : Disabled / 1 : Enabled) 13 : TLS enable (0 : Disabled / 1 : Enabled) 14 : xmDNS enable (0 : Disabled / 1 : Enabled) 15 : Registration enable (0 : Disabled / 1 : Enabled) 16 : Time Sync enable (0 : Disabled / 1 : Enabled) 17 : DRLC enable (0 : Disabled / 1 : Enabled) 18 : TP enable (0 : Disabled / 1 : Enabled) buf is the pointer to each type.

buf_len is the length of buf.
Return 0 : Success / -1 : Failure

10.2.16 ict_api_sta_set_ps_mode (optional)

Function set PS mode in NV memory.
Prototype INT32 ict_api_sta_set_ps_mode (UINT8 power_save_mode)
Arguments power_save_mode 0 - Auto (default), Automatically power save is on /off by traffic conditions 1 – Always power save is off 2 – Always power save is on
Return 0 : Success / -1 : Failure

10.2.17 ict_api_sta_get_ps_mode (optional)

Function get PS mode from variable.
Prototype UINT8 ict_api_sta_get_ps_mode (void)
Arguments N/A.
Return 0 - Auto (default), Automatically power save is on /off by traffic conditions 1 – Always power save is off 2 – Always power save is on

10.2.18 ict_api_sta_set_wireless_mode

Enum
typedef enum { ICT_WM_G_ONLY, ICT_WM_BG, ICT_WM_BGN,

```
        ICT_WM_B_ONLY,
        ICT_WM_N_ONLY,
        ICT_WM_MAX
    } ICT_WM;       // Wireless Mode...
```

Function set wireless mode.
Prototype INT32 ict_api_sta_set_wireless_mode (ICT_WM wireless_mode)
Arguments wireless_mode 0 - G only 1 – BG 2 – BGN 3 – B only 4 – N only
Return 0 : Success / -1 : Failure

10.2.19 ict_api_sta_get_wireless_mode

Function get wireless mode.
Prototype INT32 ict_api_sta_get_wireless_mode (void)
Arguments N/A.
Return 0 - G only 1 – BG 2 – BGN 3 – B only 4 – N only

11. Wi-Fi Management APIs

The Wi-Fi management APIs are used to communicate between the CM and WF5000/WF6000 module for Wi-Fi connection.

Command	Description
Scan	Finds an AP that will be joined.
Join	Associates to a selected AP from a scanned result.
Disconnect	Disconnects from an associated AP.
Get MAC address	Gets the MAC address from WF5000/WF6000 module.
Set IP configuration	Configures the IP to WF5000/WF6000 module.
DM shell	Provides DM (Diagnostic Monitoring) shell command line interface.

Table 11-1. Command of Wi-Fi management APIs

11.1 SCAN

This command is issued by the CM to send *Scan Request* to WF5000/WF6000 module. When *Scan Request* is sent to WF5000/WF6000 module, a frequency list for scanning can be included within the command. If a frequency list is specified in the command, the channels in the list are scanned by WF5000 module. Otherwise, WF5000/WF6000 module performs scanning using its internal static list in which 2.4 GHz band channels are included. If 5 GHz band scanning is required, WF5000/WF6000 module should be configured to scan 5 GHz band channels.

11.1.1 Flow

If a *Scan Request* command is received from the CM, WF5000/WF6000 module sends a *Probe Request* frame to an AP at each channel included in the channel list of the *Scan Request* command. If an AP receives the Probe Request frame, the AP should send a *Probe Response* frame destined to WF5000/WF6000 module. The *Probe Response* frame, received by WF5000/WF6000 module, is indicated by ICT_HIF_CMD_ST_SCAN_IND event to the CM. If the whole channels included in the channel list have been scanned, WF5000/WF6000 module notifies the completion of scanning by issuing ICT_HIF_CMD_ST_SCAN_RST_IND event to the CM. Whether joining an AP using the AP list included in the result or scanning channels again is decided by the CM receiving ICT_HIF_CMD_ST_SCAN_RST_IND event.

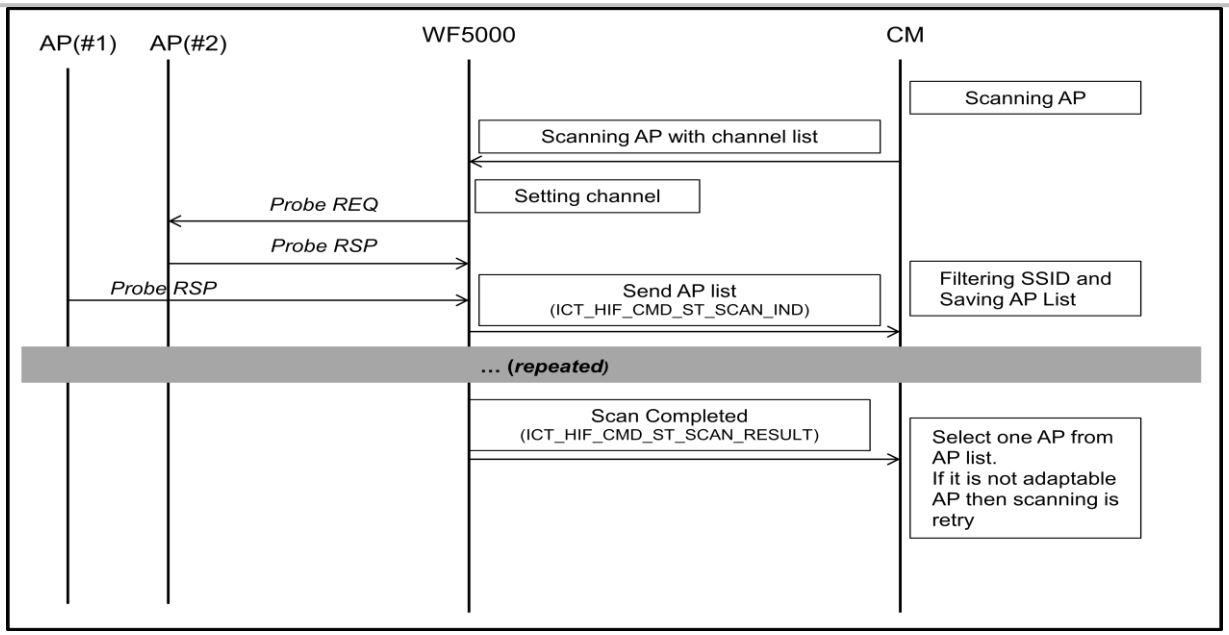


Figure 11-1. Scanning Process

11.1.2 Scan Req (ict_api_scan_handler)

It is issued by the CM and used to send a *Scan Request* command to WF5000/WF6000 module.

Prototype	Description
INT32 **ict_api_scan_hanlder** (ICT_ST_SCAN_REQ_T *params);	It is issued by the CM and used to perform a scanning procedure. Return value - 0: Successful / Otherwise, Failure

```
typedef struct
{
   UINT8        ssid[MAX_SSID_LEN];
   UINT32       ssid_len;
   UINT16       *channel;
} ICT_ST_SCAN_REQ_T;
```

Structure	Description
channel	List of the channels to be scanned. The channel is index of frequency '1' - 2412 '2' - 2417 '3' - 2422 '4' - 2427 '5' - 2432 '6' - 2437 '7' - 2442 '8' - 2447

	'9' - 2452 '10' - 2457 '11' - 2462 '12' - 2467 '13' - 2472 '14' - 2484
ssid	SSID of the Access Point.
ssid_len	Length of *ssid* if it is present The maximum length of ssid is 32 bytes.

Table 11-2. Structure of scan request

11.1.3 Scan Ind (ICT_HIF_CMD_ST_SCAN_IND)

It is issued by WF5000/WF6000 module and used to send *Probe Response* for the scanned result to the CM.

```
typedef struct
{
    UINT32      no;
    UINT8       ssid[MAX_SSID_LEN];
    UINT8       ssid_len;
    UINT8       bssid[MAC_ADDR_LEN];
    UINT16      ch;
    UINT32      bss_type;
    UINT32      bss_sub_type;
    INT32       rssi;
    INT32       noise;
    UINT8       auth_enc_type;
    UINT8       pairwise_cipher;
    UINT8       group_cipher;
} ICT_ST_SCAN_IND_T;
```

Structure	Description
no	Number of scanned Access Points.
ssid	SSID of the Access Point.
ssid_len	Length of SSID.
bssid	Basic Service Set ID of Access Point.
ch	Channel.
bss_type	BSS type used by AP. '0' : BSS_TYPE_UNSPEC '1' : BSS_TYPE_INDEPENDENT '2' : BSS_TYPE_INFRASTRUCTURE '3' : BSS_TYPE_AP '4' : BSS_TYPE_ANY

bss_sub_type	If P2P or WPS is supported by AP, the *bss_sub_type* is included. Otherwise, this field is set to zero. Its bitwise value means; '0' : BSS_SUB_TYPE_NONE '1' : BSS_SUB_TYPE_WPS '2' : BSS_SUB_TYPE_P2P '4' : BSS_SUB_TYPE_OTHERS
rssi	Absolute value of the RSSI information. It indicates the signal strength of the Access Point.
noise	N/A.
auth_enc_type	The authentication encryption type, configured by the AP sending the *Probe Response*, is included in the Auth_enc_type field. '0' : AUTH_ENC_TYPE_NONE (Open) '1' : AUTH_ENC_TYPE_WEP '2' : AUTH_ENC_TYPE_WPA_PSK '4' : AUTH_ENC_TYPE_WPA2_PSK
pairwise cipher	A pairwise cipher type about unicast data frame is included in the pairwise cipher field when WPA or WPA2 is used by the AP. '0' : PAIRWISE_CIPHER_NONE '2' : PAIRWISE_CIPHER_TKIP '4' : PAIRWISE_CIPHER_CCMP
group cipher	A group cipher type about broadcasting data frame is included in the group cipher field when WPA or WPA2 is used by the AP. '0' : GROUP_CIPHER_NONE '2' : GROUP_CIPHER_TKIP '4' : GROUP_CIPHER_CCMP

Table 11-3. Structure of scan indication

11.1.4 Scan Result (ICT_HIF_CMD_ST_SCAN_RST_IND)

Structure	Description
N/A	N/A

Table 11-4. Structure of scan result

It is issued by WF5000/WF6000 module and used to send a *Scan Result* to the CM When WF5000/WF6000 module finishes scanning about all selected channels, the result is sent to the CM. If re-scanning is required, it should be started after receiving the *Scan Result*

11.2 Joining

It is used to associate an AP selected from the scan result.

11.2.1 Flow

If the CM receives a *Scan Result*, it should select an AP from the scan result. If an AP is selected, the information about the SSID and the related values of the selected AP are passed to WF5000/WF6000 module. If WF5000/WF6000 module receives the information about AP, it tries to join the AP after ensuring the AP is in the channel. A four way handshake procedure and a DHCP procedure are performed, if these procedures are needed to join the AP. The whole procedures for joining are finished, *ICT_HIF_CMD_ST_JOIN_IND* event is issued to the CM by WF5000/WF6000 module with the joining status.

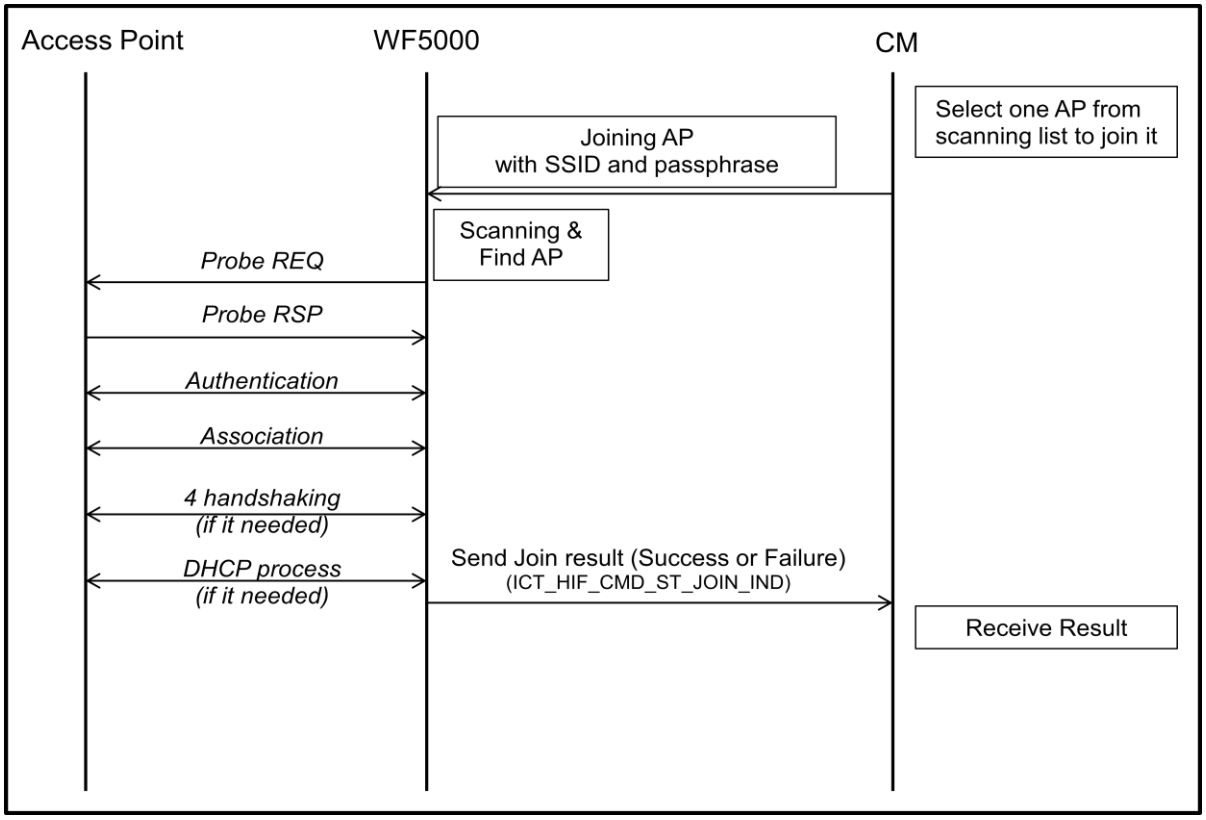


Figure 11-2. Joining Process

11.2.2 Join Req (ict_api_join_handler)

Prototype	Description
INT32 **ict_api_join_handler** (ICT_ST_SCAN_REQ_T *param	It is issued by the CM and used to perform a joining procedure.

);	Return value - 0: Successful / Otherwise, Failure

```
typedef struct
{
    UINT8       ssid[MAX_SSID_LEN];
    UINT32      ssid_len;
    UINT8       bssid[MAC_ADDR_LEN];
    UINT16      ch;
    UINT8       auth_type;
    UINT8       auth_enc_type;
    UINT8       pairwise_cipher;
    UINT8       group_cipher;
    UINT8       key_idx;
    UINT8       key[MAX_PSK_LEN];
    UINT32      key_len;
} ICT_ST_JOIN_REQ_T;
```

Structure	Description
ssid	SSID of the Access Point.
ssid_len	Length of SSID
auth_type	An authentication type used to perform a joining procedure. '0' : AUTH_TYPE_OPEN '1' : AUTH_TYPE_SHARED_KEY '4' : AUTH_TYPE_AUTO_SWITCH
bssid	Basic Service Set ID of Access Point.
ch	Channel
auth_enc_type	It is an authentication encryption type configured by a selected AP. '0' : AUTH_ENC_TYPE_NONE (Open) '1' : AUTH_ENC_TYPE_WEP '2' : AUTH_ENC_TYPE_WPA_PSK '4' : AUTH_ENC_TYPE_WPA2_PSK
pairwise cipher	A pairwise cipher type about unicast data frame is included in the pairwise cipher field when WPA or WPA2 is used by the AP. '0' : PAIRWISE_CIPHER_NONE '2' : PAIRWISE_CIPHER_TKIP '4' : PAIRWISE_CIPHER_CCMP
group cipher	A group cipher type about broadcast data frame is included in the group cipher field when WPA or WPA2 is used by the AP. '0' : GROUP_CIPHER_NONE '2' : GROUP_CIPHER_TKIP '4' : GROUP_CIPHER_CCMP
key_idx	If the encryption type of a selected AP is WEP, the value of the key_idx field is set to a used key index. Otherwise, this field is reserved.

key	It includes a key pushed by user.
key_len	It indicates the length of the value of key field.
vendor_specific_ie_len	It indicates the length of vendor specific IE. N/A.
p_vendor_specific_ie	It includes a vendor specific IE pushed by user. N/A.

Table 11-5. Structure of join request

11.2.3 Join Ind (ICT_HIF_CMD_ST_JOIN_IND or ICT_HIF_CMD_ST_AP_JOIN_IND)

```
typedef struct
{
    UINT32      result;
} ICT_ST_JOIN_IND_T;
```

Structure	Description
result	'0' : APP_JOIN_SUCCESS '1' : APP_JOIN_FAILURE '2' : APP_JOIN_OTHERS

Table 11-6. Structure of join indication

11.2.4 AP Join Ind on AP mode (ICT_HIF_CMD_ST_START/STOP_IND)

See sub-clause 11.2.3

11.2.5 STA Join Ind on AP mode (ICT_HIF_CMD_ST_STA_[DIS]ASSOCIATED_IND)

```
typedef struct
{
    UINT8   mac_address[MAC_ADDR_LEN];
    INT32   rssi;
} ICT_ST_STA_INFO_T;
```

Structure	Description
mac_address	The MAC address of joining STA to the AP
rssi	The RSSI value of joining STA to the AP

Table 11-7. Structure of STA join indication

11.3 Disconnect

It is used to be disconnected from a joined AP by STA or to be received a de-authentication message from the joined AP by STA.

Reason about disconnection indication is currently not supported.

11.3.1 Flow

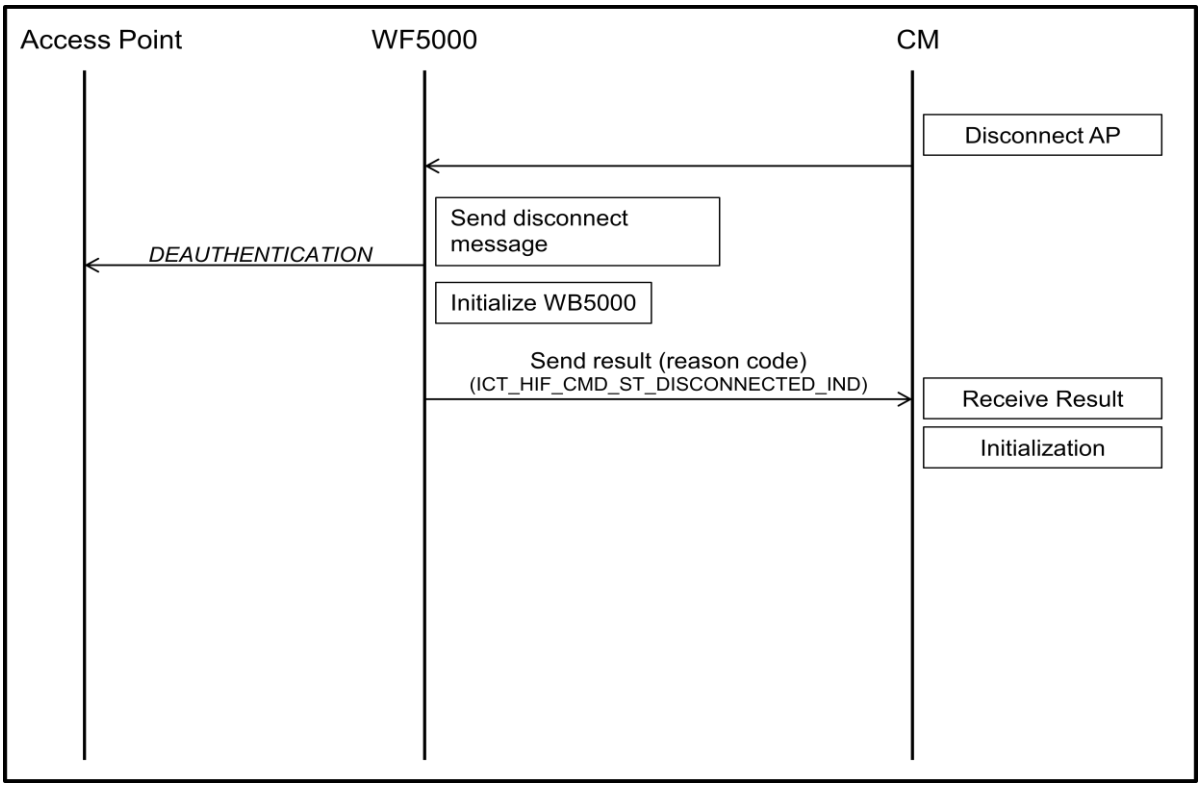


Figure 11-3. Disconnect – STA initiator

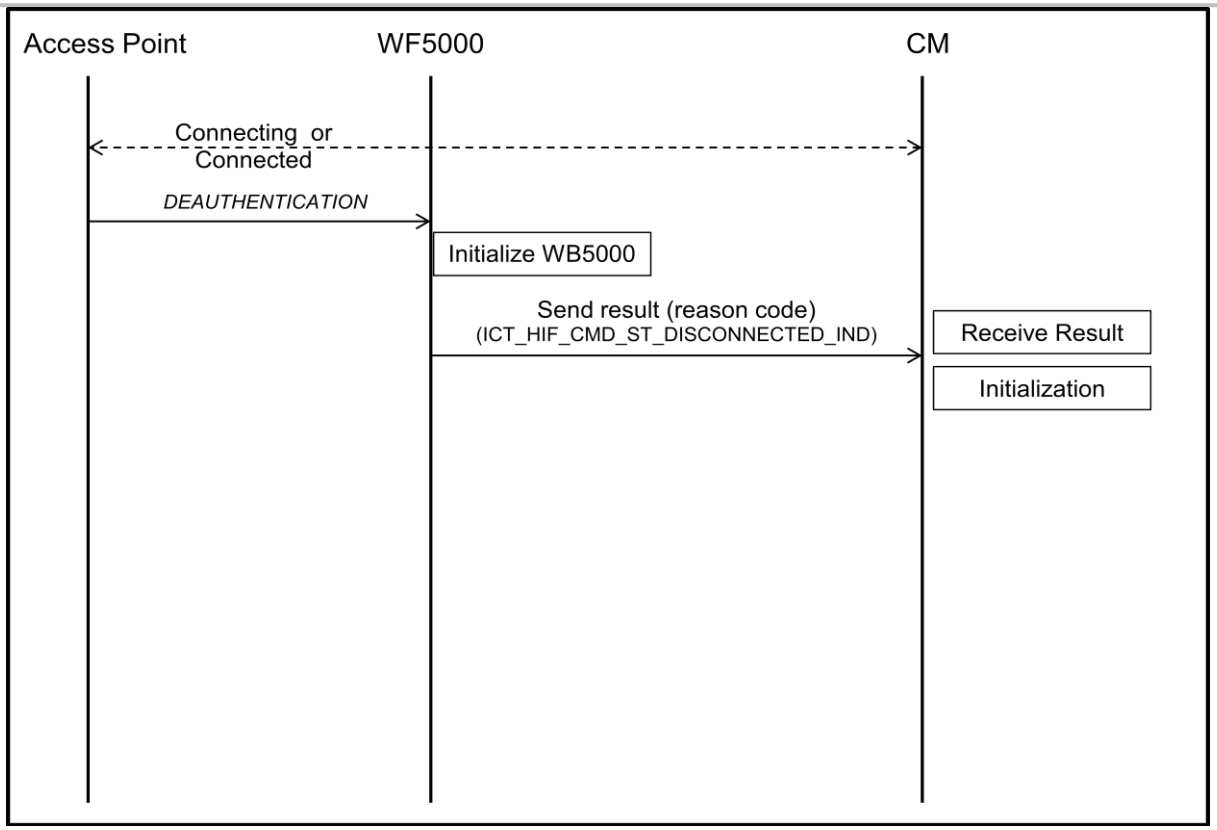


Figure 11-4. Disconnect – AP initiator

11.3.2 Disconnect (ict_api_disconnect_handler)

Prototype	Description
INT32 **ict_api_disconnect_handler** (UINT16 reason);	It is issued by the CM and used to perform a disconnecting procedure. Return value - 0: Successful / Otherwise, Failure

```
typedef struct
{
    UINT16      reason;
} ICT_ST_DISCONNECT_T;
```

Structure	Description
Reason	'1' : UNSPECIFIED '2' : PREV_AUTH_NOT_VALID '3' : DEAUTH_LEAVING '4' : DISASSOC_DUE_TO_INACTIVITY '5' : DISASSOC_AP_BUSY '6' : CLASS2_FRAME_FROM_NONAUTH_STA '7' : CLASS3_FRAME_FROM_NONASSOC_STA

	'8' : DISASSOC_STA_HAS_LEFT '9' : STA_REQ_ASSOC_WITHOUT_AUTH '10' : PWR_CAPABILITY_NOT_VALID '11' : SUPPORTED_CHANNEL_NOT_VALID '13' : INVALID_IE '14' : MICHAEL_MIC_FAILURE '15' : 4WAY_HANDSHAKE_TIMEOUT '16' : GROUP_KEY_UPDATE_TIMEOUT '17' : IN_4WAY_DIFFERS '18' : GROUP_CIPHER_NOT_VALID '19' : PAIRWISE_CIPHER_NOT_VALID '20' : AKMP_NOT_VALID '21' : UNSUPPORTED_RSN_IE_VERSION '22' : INVALID_RSN_IE_CAPAB '23' : IEEE_802_1X_AUTH_FAILED '24' : CIPHER_SUITE_REJECTED '39' : TIMEOUT

Table 11-8. Structure of de-authentication

11.3.3 Disconnected Ind (ICT_HIF_CMD_ST_DISCONNECTED_IND)

11.3.4 AP Connection (ict_api_apconn_handler)

Prototype	Description
INT32 **ict_api_disconnect_handler** (UINT16 reason);	It is issued by the CM and used to operate WF5000/WF6000 as an AP mode or IBSS mode. Return value - 0: Successful / Otherwise, Failure

11.4 WPS

11.4.1 Flow

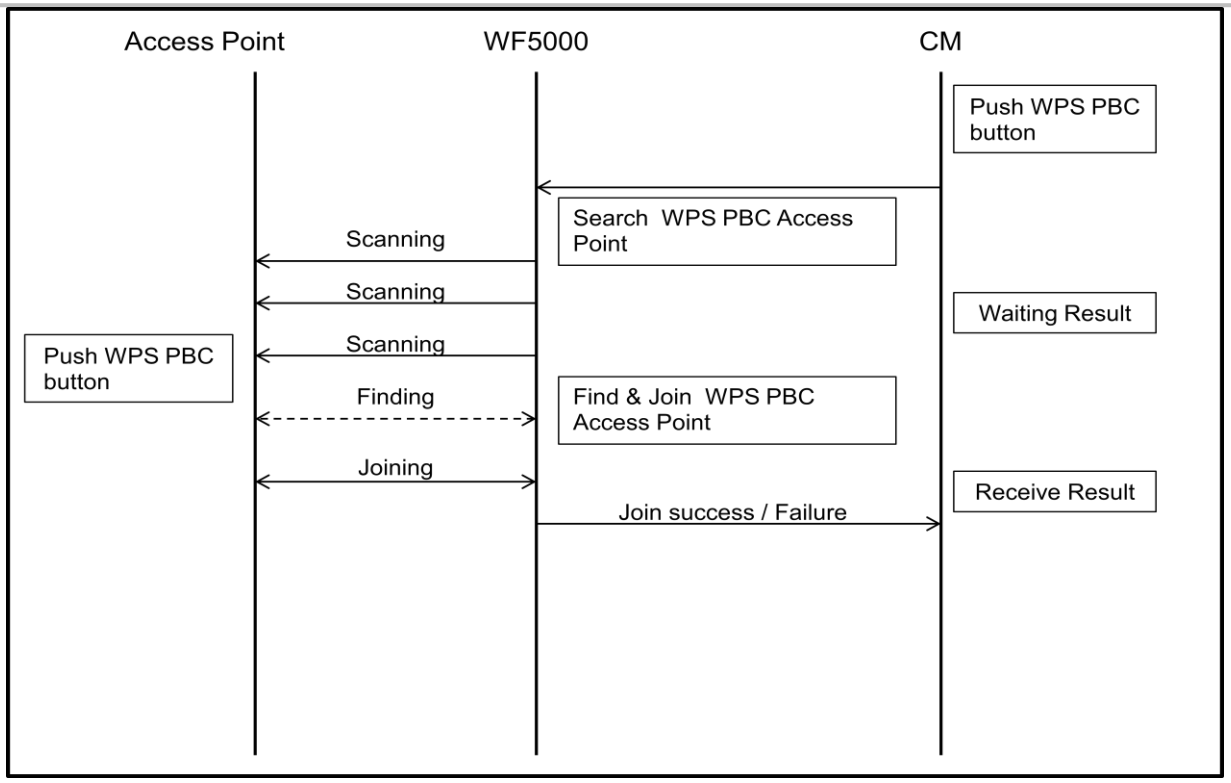


Figure 11-5. WPS PBC Process

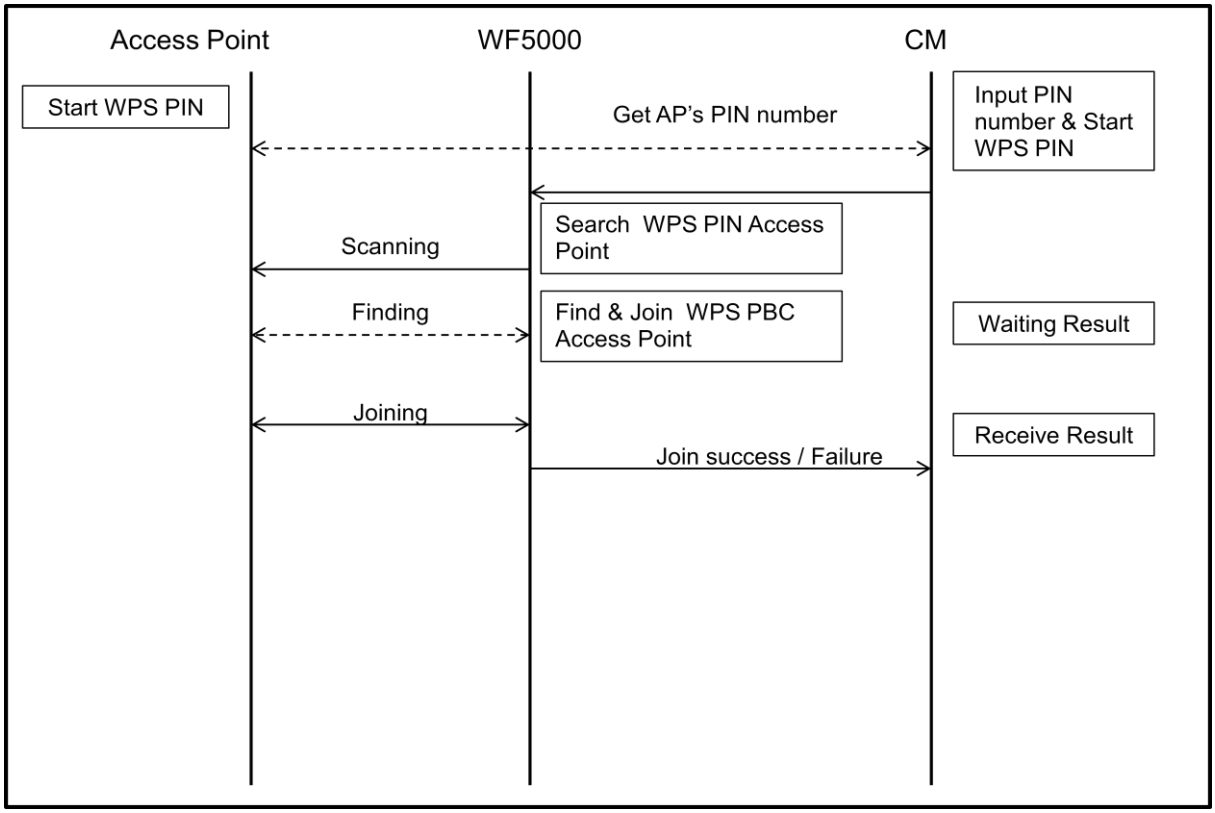


Figure 11-6. WPS Pin Process – using AP's PIN

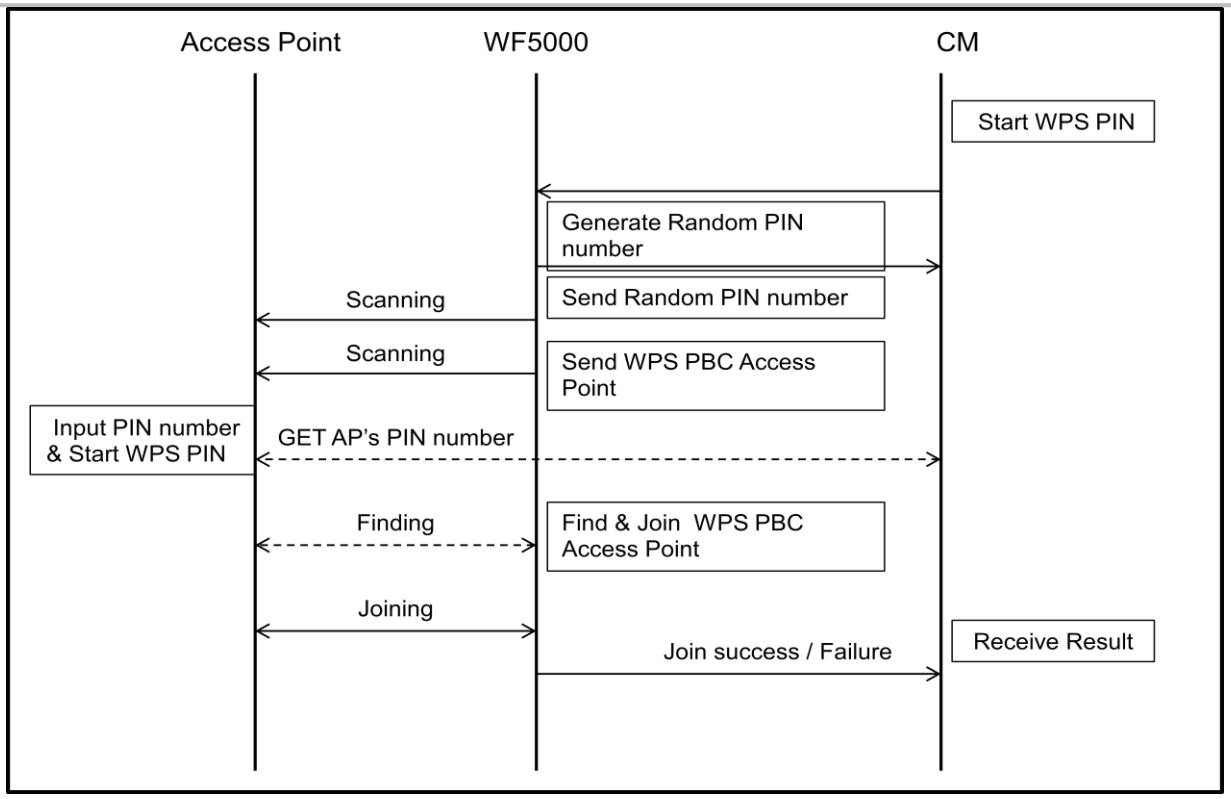


Figure 11-7. WPS PIN process – using STA's PIN

11.4.2 WPS-PBC Req (ict_api_wps_pbc_handler)

Prototype	Description
INT32 **ict_api_wps_pbc_handler** (UINT8 *buf, UINT32 buf_len);	Return value - 0: Successful / Otherwise, Failure

Structure	Description
buf	"any"
buf_len	It indicates of the length of buffer. [length = 3]

Table 11-9. Structure of WPS PBC

11.4.3 WPS-PIN Req (ict_api_wps_pin_handler)

Not Supported.

Prototype	Description
INT32 **ict_api_wps_pin_handler** (ICT_ST_WPS_PIN_T *params	Return value - 0: Successful / Otherwise, Failure

);	

Structure	Description
pin	Pin number (8 digits)

Table 11-10. Structure of WPS-PIN

11.4.4 WPS-Cancel Req (ict_api_wps_cancel_handler)

Not Supported.

Prototype	Description
INT32 **ict_api_wps_cancel_handler** ();	Return value - 0: Successful / Otherwise, Failure

11.4.5 WPS Ind (ICT_HIF_CMD_ST_WPS_IND)

Not Supported.

The indication is only accepted when WPS PIN is required for WPS negotiation.

```
typedef ADS_PACKED struct
{
    UINT16      result;
    UINT16      len;
    UINT8       data[1];
} GCC_PACKED ICT_ST_WPS_IND_T;
```

Structure	Description
Result	'0' : API_WPS_IND_SUCCESS '1' : API_WPS_FAILURE '2' : API_WPS_IND_PIN_NUM
Len	Length of data
Data	Pin number - 8 digits

Table 11-11. Structure of WPS indication

11.5 P2P

11.5.1 Flow

11.5.2 P2P Find (ict_api_p2p_find_handler)

Prototype	Description
INT32 **ict_api_p2p_find_handler** ();	Return value - 0: Successful / Otherwise, Failure

11.5.3 P2P Find Ind (ICT_HIF_CMD_ST_P2P_DEVICE_FOUND_IND)

```
typedef PACKED struct
{
    UINT8  addr[MAC_ADDR_LEN];
    UINT8  dev_name[33];
    UINT16 config_method;
} XTENSA_PACKED ICT_ST_P2P_DEV_FOUND_T;
```

Structure	Description
addr	Peer MAC address
dev_name	
config_method	

Table 11-12. Structure of P2P Find indication

11.5.4 P2P Lost Ind (ICT_HIF_CMD_ST_P2P_DEVICE_LOST_IND)

```
typedef PACKED struct
{
    UINT8  addr[MAC_ADDR_LEN];
    UINT8  dev_name[33];
    UINT16 config_method;
} XTENSA_PACKED ICT_ST_P2P_DEV_FOUND_T;
```

Structure	Description
addr	Peer MAC address
dev_name	
config_method	

Table 11-13. Structure of P2P Lost indication

11.5.5 P2P GO Negotiation Ind (ICT_HIF_CMD_ST_P2P_GO_NEG_IND)

```
typedef struct
{
    UINT8   p2p_peer[MAC_ADDR_LEN];
    UINT16  dev_passwd_id;
} ICT_ST_PEER_GO_REQ_T;
```

Structure	Description
p2p_peer	Peer MAC address
dev_passwd_id	

Table 11-14. Structure of P2P GO Negotiation indication

11.5.6 P2P Result Ind (ICT_HIF_CMD_ST_P2P_RESULT_IND)

Structure	Description
p2p_result	0 : Success; Otherwise, Failure

Table 11-15. Structure of P2P Result indication

11.5.7 P2P Stop Find (ict_api_p2p_stop_find_handler)

Prototype	Description
INT32 **ict_api_p2p_stop_find_handler** ();	Return value - 0: Successful / Otherwise, Failure

11.5.8 P2P Connect (ict_api_p2p_connect_handler)

Prototype	Description
INT32 **ict_api_p2p_connect_handler** (UINT8 *buf, UINT32 buf_len);	Return value - 0: Successful / Otherwise, Failure

Structure	Description
buf	is a pointer of a string that consists of "<peer device address> <pbc | pin | PIN#>" E.g.) "02:0A:F5:99:86:CC pbc" or "02:0A:F5:99:86:CC pin 123456780"
buf_len	It indicates of the length of buffer.

Table 11-16. Structure of P2P Connect

11.5.9 P2P Cancel (ict_api_p2p_cancel_handler)

Prototype	Description
INT32 **ict_api_p2p_cancel_handler** ();	Return value - 0: Successful / Otherwise, Failure

11.5.10 P2P Reject (ict_api_p2p_reject_handler)

Prototype	Description
INT32 **ict_api_p2p_reject_handler** (UINT8 *buf, UINT32 buf_len);	Return value - 0: Successful / Otherwise, Failure

Structure	Description
buf	is a pointer of a string to "<peer device address>" E.g.) "02:0A:F5:99:86:CC "
buf_len	It indicates of the length of buffer.

Table 11-17. Structure of P2P Connect

11.6 Sniffer Mode

11.6.1 ict_api_set_channel

Function Set operating channel of WF5000/WF6000.
Prototype INT32 ict_api_set_channel(UINT32 channel)
Arguments channel 1 ~ 165
Return ICT_OK

11.6.2 ict_api_set_sniffer_mode

Function Set sniffer mode enable/disable of WF5000/WF6000.
Prototype UINT8 ict_api_set_sniffer_mode(UINT32 enable, void *rx_callback)
Arguments enable 1: Enable / 0: Disable rx_callback register callback function. when frame received, this callback function would be called FCS error frame is not handled.
Return ICT_OK

11.7 Indications

11.7.1 Network Ind (ICT_HIF_CMD_ST_NETWORK_INFO_IND)

The network indication is sent to the CM when WF5000/WF6000 module is connected to a network

with an IP address. If each field of the network indication is a valid IP address instead of all zero, it is possible to send and receive data. If disconnected indication is issued by the CM, the fields of the network indication should be set to all zero.

```
typedef struct
{
    UINT8       ipaddr[4];
    UINT8       subnet[4];
    UINT8       gateway[4];
    UINT8       dns[4];
} ICT_ST_NETWORK_INFO_IND_T;
```

Structure	Description
ipaddr	IP address
subnet	Netmask
gateway	Gateway
Dns	DNS server

Table 11-18. Structure of network info indication

11.7.2 MAC Address Ind (ICT_HIF_CMD_ST_MAC_ADDR_IND)

The MAC Address indication is sent to the CM by user application. The indication is NOT depended on WF5000/WF6000 module.

11.7.3 Device Ready Ind (ICT_HIF_CMD_ST_DEVICE_READY_IND)

The Device Ready indication is sent to the CM when WF5000/WF6000 module is on "Device Ready" state turning power on. All of user defined commands such as starting STA, starting AP, or sending DM commands should be pushed in WF5000/WF6000 module from now on the Device Ready indication.

11.7.4 Hardware Power Save Ind (ICT_HIF_CMD_ST_HWPS_IND)

The Hardware Power Save indication is sent to the CM when WF5000/WF6000 module is going to go sleep status or awake status.
If sleep status (0) of WF5000/WF6000 module is indicated, then the counts of incoming TX data frames including TX UART data frames should be notified to WF5000/WF6000 module though ict_api_send_uart_traffic_handler() function at every ONE second.
If awake status (1) of WF5000/WF6000 module is indicated, then External HW timer should be restarted and UART interface for user should be reconfigured.

11.7.5 Firmware Upgrade Ind (ICT_HIF_CMD_ST_XMODEM_IND)

The XMODEM Indication is sent to the CM when firmware of WF5000/WF6000 module is updated by XMODEM. Indicated values are described below :

Indicated values	Description
0	Success
-1	Canceled by remote
-2	Sync error
-3	Too many retry count
-4	Not enough memory
-5	This firmware file is abnormal

Table 11-19. Structure of network info indication

12. TCP/IP APIs

12.1 IP configuration

```
typedef struct
{
   UINT16       dhcp_mode;
   UINT8        ipaddr[4];
   UINT8        subnet[4];
   UINT8        gateway[4];
   UINT8        dns[4];
} ICT_ST_IP_CONFIG_T;
```

Structure	Description
dhcp_mode	DHCP mode configuration '0' : Manual IP configuration '1' : DHCP IP configuration
ipaddr	IP address when working as Manual IP mode
subnet	Netmask address when working as Manual IP mode.
gateway	Gateway address when working as Manual IP mode.
dns	DNS server when working as Manual IP mode.

Table 12-1. Structure of configure IP

12.1.1 ict_api_tcpip_set_ip_config_handler

Prototype	Description
INT32 **ict_api_set_ip_config_handler** (ICT_ST_IP_CONFIG_T *params);	It is issued by the CM and used to set IP configuration to WF5000/WF6000 module. Return value - 0: Successful / Otherwise, Failure

12.1.2 ict_api_tcpip_get_ip_config_handler

Prototype	Description
INT32 **ict_api_get_ip_config_handler** (ICT_ST_IP_CONFIG_T *params);	It is issued by the CM and used to get IP configuration to WF5000/WF6000 module. Return value - 0: Successful / Otherwise, Failure

12.2 TCP/UDP

```
typedef struct
{
   UINT16  socket_type;
```

```
    UINT16  local_port;
    UINT16  remote_port;
    UINT8   remote_ipaddr[4];
} ICT_ST_SOCKET_T;

/*
typedef struct
{
    UINT16  socket_type;
    UINT16  local_port;
    UINT16  remote_port;
    UINT8   remote_ipaddr[4];
} ICT_ST_SOCKET_CREATE_T;
*/

typedef struct
{
    INT32   socket_desc;
} ICT_ST_SOCKET_CLOSE_T;

typedef PACKED struct
{
    UINT16 socket_cmd;
    UINT16 socket_type;
    INT32  socket_desc;
    INT16  result;
} XTENSA_PACKED ICT_ST_SOCKET_IND_T;

typedef struct
{
    UINT8  sa_len;
    UINT8  sa_family;
    UINT16 sa_port;
    UINT8  sa_ipaddr[4];
    UINT8  sa_zero[8];
} ICT_ST_SOCKET_ADDR_T;

typedef struct
{
    INT32  socket_desc;
    ICT_ST_SOCKET_zADDR_T sa;
    INT32  result;
} ICT_ST_TCP_DISCONNECT_IND_T;
```

Prototype	Description
INT32 **ict_api_tcpip_socket_create_handler** (ICT_ST_SOCKET_T *params);	It is issued by the CM and used to perform the corresponding action on opening a socket.
INT32 **ict_api_tcpip_socket_close_handler** (	It is issued by the CM and used to perform

ICT_ST_SOCKET_CLOSE_T *params);	closing a socket.
INT32 **ict_api_tcpip_tcp_disconnect_handler** (ICT_ST_SOCKET_CLOSE_T *params, ICT_ST_SOCKET_ADDR_T *ra);	It is issued by the CM and used to perform disconnecting a TCP client related a TCP server when the TCP server is running on WF5000/WF6000. ra : sa_port & sa_ipaddr are only executable.

12.2.1 Create Socket (ict_api_tcpip_socket_create_handler)

```
typedef struct
{
    UINT16  socket_type;
    UINT16  local_port;
    UINT16  remote_port;
    UINT8   remote_ipaddr[4];
} ICT_ST_SOCKET_T;
```

Structure	Description
socket_type	Type of the created socket. '1' : SOCKET_TYPE_TCP_CLIENT (will be supported) '2' : SOCKET_TYPE_UDP_CLIENT '4' : SOCKET_TYPE_TCP_SERVER (will be supported) '8' : SOCKET_TYPE_UDP_SERVER
local_port	It is used for listening when WF5000/WF6000 is operated as TCP Server or UDP Server. It is not referenced when WF5000/WF6000 is operated as TCP Client or UDP Client.
remote_port	It is used as a destinationport to send data when WF5000/WF6000 is operated as TCP Client.
remote_ipaddr	It is used as a destinationIP address to send data when WF5000/WF6000 is operated as TCP Client.

Table 12-2. Structure of create socket

Create UDP Client

```
ICT_ST_SOCKET_T params;

ICT_MEMSET(&params, 0x00, sizeof(ICT_ST_SOCKET_T));

params.socket_type = SOCKET_TYPE_UDP_CLIENT;
arams.local_port = 0;
```

```
params.remote_port = 50040;
params.remote_ipaddr[0] = 192;
params.remote_ipaddr[1] = 168;
params.remote_ipaddr[2] = 1;
params.remote_ipaddr[3] = 100;

(void)ict_api_create_socket(&params);
```

Create UDP Server

```
ICT_ST_SOCKET_T params;

ICT_MEMSET(&params, 0x00, sizeof(ICT_ST_SOCKET_T));

params.socket_type = SOCKET_TYPE_UDP_SERVER;
params.local_port = 50030;
params.remote_port = 0;
params.remote_ipaddr[0] = 0;
params.remote_ipaddr[1] = 0;
params.remote_ipaddr[2] = 0;
params.remote_ipaddr[3] = 0;

(void)ict_api_create_socket(&params);
```

Create TCP Client

```
ICT_ST_SOCKET_T params;

ICT_MEMSET(&params, 0x00, sizeof(ICT_ST_SOCKET_T));

params.socket_type = SOCKET_TYPE_TCP_CLIENT;
arams.local_port = 0;
params.remote_port = 50020;
params.remote_ipaddr[0] = 192;
params.remote_ipaddr[1] = 168;
params.remote_ipaddr[2] = 1;
params.remote_ipaddr[3] = 100;

(void)ict_api_create_socket(&params);
```

Create TCP Server

```
ICT_ST_SOCKET_T params;

ICT_MEMSET(&params, 0x00, sizeof(ICT_ST_SOCKET_T));

params.socket_type = SOCKET_TYPE_TCP_SERVER;
params.local_port = 50010;
params.remote_port = 0;
params.remote_ipaddr[0] = 0;
params.remote_ipaddr[1] = 0;
params.remote_ipaddr[2] = 0;
params.remote_ipaddr[3] = 0;

(void)ict_api_create_socket(&params);
```

12.2.2 Close Socket (ict_api_tcpip_socket_close_handler)

```
typedef struct
{
    INT32   socket_desc;
} ICT_ST_SOCKET_CLOSE_T;
```

Structure	Description
socket_desc	Socket descriptor of the socket to be closed.

Table 12-3. Structure of close socket

12.2.3 Disconnect TCP Client (ict_api_tcpip_tcp_disconnect_handler)

```
typedef struct
{
    INT32   socket_desc;
} ICT_ST_SOCKET_CLOSE_T;

typedef struct
{
    UINT8  sa_len;
    UINT8  sa_family;
    UINT16 sa_port;
    UINT8  sa_ipaddr[4];
    UINT8  sa_zero[8];
} ICT_ST_SOCKET_ADDR_T;
```

Structure	Description
socket_desc	The socket descriptor of the remote TCP client socket to be closed.
sa_len	N/A.
sa_family	N/A.
sa_port	The port number of the remote TCP client socket to be closed.
sa_ipaddr	The IP address of the remote TCP client socket to be closed.
sa_zero	N/A.

Table 12-4. Structure of disconnecting remote TCP client socket

12.2.4 Socket Ind (ICT_HIF_CMD_ST_SOCKET_IND)

```
typedef PACKED struct
{
    UINT16 socket_cmd;
    UINT16 socket_type;
    INT32  socket_desc;
    INT16  result;
} XTENSA_PACKED ICT_ST_SOCKET_IND_T;
```

Structure	Description
socket_cmd	Type of the socket command. '0' : SOCKET_CMD_CREATE '1' : SOCKET_CMD_CLOSE
socket_type	Type of the corresponding socket. '1' : SOCKET_TYPE_TCP_CLIENT (will be supported) '2' : SOCKET_TYPE_UDP_CLIENT '4' : SOCKET_TYPE_TCP_SERVER '8' : SOCKET_TYPE_UDP_SERVER
socket_desc	It indicates that a valid socket descriptor value as the result of Create Socket Request. It indicates the value set to -1, which is initial value of socket descriptor, as the result of Close Socket Request.
result	'0' : ERR_OK (No error) '-9' : ERR_VALID (Illegal value.)

Table 12-5. Structure of socket indication

12.2.5 TCP Client Disconnect Ind (ICT_HIF_CMD_ST_TCP_DISCONNECT_IND)

```
typedef struct
{
```

```
    UINT8  sa_len;
    UINT8  sa_family;
    UINT16 sa_port;
    UINT8  sa_ipaddr[4];
    UINT8  sa_zero[8];
} ICT_ST_SOCKET_ADDR_T;

typedef struct
{
    INT32  socket_desc;
    ICT_ST_SOCKET_ADDR_T sa;
    INT32  result;
} ICT_ST_TCP_DISCONNECT_IND_T;
```

Structure	Description
socket_desc	It indicates that a valid socket descriptor value as the result of Close TCP Client Socket Request.
sa_len	
sa_family	
sa_port	
sa_ipaddr	
sa_zero	
result	'0' : ERR_OK (No error) / Otherwise, Error

Table 12-6. Structure of disconnecting remote TCP client indication

12.3 Traffic Data

```
typedef PACKED struct
{
    UINT16 socket_type;
    INT32  socket_desc;
    UINT16 remote_port;
    UINT8  remote_ipaddr[4];
    UINT32 data_len;
    UINT8  data[1];
} XTENSA_PACKED   ICT_ST_HIF_DATA_T;
```

Prototype	Description
INT32 **ict_api_send_data_handler** (ICT_ST_HIF_DATA_T *sock_info, UINT32 size);	It is issued by the CM and used to send traffic data to the module.
ICT_ST_HIF_DATA_T ***ict_api_rcvd_data_handler**	It is issued by the CM and used to parse starting

(UINT8 *buf, UINT32 *data_len);	pointer and the length of a received traffic data for user application.
UINT8 ***ict_api_rcvd_data_sw_type_handler** (UINT8 *buf, UINT32 *data_len, UINT32 *sw_type, UINT32 *more_flag);	Not Supported. It is issued by the CM and used to parse SW type such as SMTP and POP3 and starting pointer and the length of a received traffic data for user application.
UINT32 **ict_api_rcvd_data_sw_opt_handler** (UINT8 *buf, UINT32 buf_len)	It is issued by the CM and used to parse SW option such as DATA and Vendor Specific. Vendor Specific includes SMTP and POP3.

12.3.1 Send Traffic Data (ict_api_send_data_handler)

```
typedef PACKED struct
{
    UINT16 socket_type;
    INT32  socket_desc;
    UINT16 remote_port;
    UINT8  remote_ipaddr[4];
    UINT32 data_len;
    UINT8  data[1];
} XTENSA_PACKED   ICT_ST_HIF_DATA_T;
```

Structure	Description
socket_type	Type of the created socket. '1' : SOCKET_TYPE_TCP_CLIENT (will be supported) '2' : SOCKET_TYPE_UDP_CLIENT '4' : SOCKET_TYPE_TCP_SERVER '8' : SOCKET_TYPE_UDP_SERVER
socket_desc	Socket descriptor.
remote_port	It is used for a destination port to send data when WF5000/WF6000 is operated as UDP Server or UDP Client. It is not referenced to send data when WF5000/WF6000 is operated as TCP Server or TCP Client because WF5000/WF6000 had been already connected to a destination port using Create Socket API.
remote_ipaddr	It is used for a destination IP address to send data when WF5000/WF6000 is operated as UDP Server or UDP Client.

	It is not referenced to send data when WF5000/WF6000 is operated as TCP Server or TCP Client because WF5000/WF6000 had been already connected to a destination port using Create Socket API.
data_len	Length of data to be sent. In case of socket_type is equal to 4 and data_len is equal to 0, it indicates that a Remote TCP Client has been connected to the TCP Server. In case of socket_type is equal to 4 and data_len is equal to 0xFFFF, it indicates that a Remote TCP Client has been disconnected to the TCP Server.
data	Actual data to be sent. A maximum of 1460 bytes can be sent in a packet.

Table 12-7. Structure of send data

12.3.2 Receive Traffic Data (ict_api_rcvd_data_handler)

```
typedef PACKED struct
{
    UINT16 socket_type;
    INT32  socket_desc;
    UINT16 remote_port;
    UINT8  remote_ipaddr[4];
    UINT32 data_len;
    UINT8  data[1];
} XTENSA_PACKED   ICT_ST_HIF_DATA_T;
```

Structure	Description
socket_type	Type of the created socket. '1' : SOCKET_TYPE_TCP_CLIENT (will be supported) '2' : SOCKET_TYPE_UDP_CLIENT '4' : SOCKET_TYPE_TCP_SERVER (will be supported) '8' : SOCKET_TYPE_UDP_SERVER
socket_desc	Socket descriptor.
remote_port	Port number of the source terminal
remote_ipaddr	IP address of the source terminal
data_len	The size of the data to be received
data	Actual data received from remote terminal

Table 12-8. Structure of receive data

13. Application Protocol APIs (optional)

The Application Protocol APIs provided in the API library are used to manage applications running upon TCP/IP layer.

13.1 DNS

13.1.1 DNS QUERY (ict_api_dns_query_handler)

Function send IP request matched to host name such as "www.google.com".
Prototype INT32 ict_api_dns_query_handler(UINT8 *data, UINT32 size)
Arguments data is a pointer to a host name such as "www.google.com". size is the length of the host name.
Return 0 : Success / Otherwise, Failure

13.1.2 DNS QUERY Ind (ICT_HIF_CMD_ST_DNSQUERY_IND)

Indicated values	Description
p_ipaddr	If size value is not ZERO, then the IP address is valid.
size	0 : Failure / Otherwise, Success

Table 13-1. Indication values of DNS indication

13.2 PING

13.2.1 PING Req (ict_api_ping_req_handler)

Function send PING request to specific IP address with predefined data length.
Prototype INT32 ict_api_ping_req_handler(UINT8 *ipaddr, UINT16 ping_data_len)
Arguments p_ipaddr is a pointer to a destination IP address of PING request.

ping_data_len is the length of PING data.
Return 0 : Success / Otherwise, Failure

13.2.2 PING Reply Ind (ICT_HIF_CMD_ST_PING_REPLY_IND)

```
typedef PACKED struct
{
    UINT8  ipaddr[4];
    UINT16 ping_data_len;
    UINT32 ping_time;
    UINT32 repeat_num;
} ICT_ST_PING_DATA_INFO_T;
```

Structure	Description
ipaddr	the source IP address of PING Reply device
ping_data_len	the length of PING data
ping_time	the total delayed ping time from PING Request
repeat_num	N/A.

Table 13-2. Structure of PING Reply indication

13.3 DHCP Server

13.3.1 DHCP Server Start (ict_api_dhcpd_start_handler)

Function start DHCP Server.
Prototype INT32 ict_api_dhcpd_start_handler(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.3.2 DHCP Server Stop (ict_api_dhcpd_stop_handler)

Function stop DHCP Server.
Prototype INT32 ict_api_dhcpd_stop_handler(void)

Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.4 HTTP Server

13.4.1 HTTP Server Start (ict_api_httpd_start_handler)

Function start HTTP Server.
Prototype INT32 ict_api_httpd_start_handler(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.4.2 HTTP Server Stop (ict_api_httpd_stop_handler)

Function stop HTTP Server.
Prototype INT32 ict_api_httpd_stop_handler(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.5 HTTP Client

13.5.1 HTTP Client initialization (ict_api_httpc_init)

Function initialize HTTP Client.
Prototype INT32 ict_api_httpc_init(UINT8 *url, UINT32 url_len, BOOL is_post)
Arguments url

<ip/domain>:<port><url> ip/domain : domain name or IP address of HTTP server port : port number of HTTP server url : URL Ex.) 192.168.0.1:8080/index.shtml url_len is the length of URL. is_post indicates whether HTTPC is initialized for POST or NOT.
Return 0 : Success / Otherwise, Failure

13.5.2 HTTP Client POST initialization (ict_api_httpc_post_octetstream_init)

Function initialize HTTP Client Octet Stream (POST).
Prototype INT32 ict_api_httpc_post_octetstream_init(UINT8 *str, UINT32 str_len, UINT8 *payload, UINT32 payload_len)
Arguments url <ip/domain>:<port><url> ip/domain : domain name or IP address of HTTP server port : port number of HTTP server url : URL Ex.) 192.168.0.1:8080/index.shtml url_len is the length of URL. payload is a pointer to payload to be posted. payload_len is the length of payload
Return 0 : Success / Otherwise, Failure

13.5.3 HTTP Client Stop - HTTP session (ict_api_httpc_close)

Function stop HTTP session.

Prototype INT32 ict_api_httpc_close(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.5.4 HTTP Client Stop - HTTPS session (ict_api_https_close)

Function stop HTTPS session
Prototype INT32 ict_api_https_close(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.5.5 HTTP Client Control Ind (ICT_HIF_CMD_ST_HTTP_CONTROL_IND)

Indicated values	Description
buf = result	indicates the result of HTTP Client process, or APP_HTTPC_RESULT_OK = 0, APP_HTTPC_RESULT_ERR_UNKNOWN = 1, APP_HTTPC_RESULT_ERR_CONNECT = 2, APP_HTTPC_RESULT_ERR_HOSTNAME = 3, APP_HTTPC_RESULT_ERR_CLOSED =4, APP_HTTPC_RESULT_ERR_TIMEOUT = 5, APP_HTTPC_RESULT_ERR_SVR_RESP = 6, APP_HTTPC_RESULT_ERR_INITIALIZE = 7, APP_HTTPC_RESULT_ERR_ARGUMENT = 8, APP_HTTPC_RESULT_ERR_MEMORY = 9, indicates the result of connecting session APP_HTTPC_RESULT_SESSION_SUCCESS = 10, APP_HTTPC_RESULT_SESSION_CLOSED = 11,

Table 13-3. Indication values of HTTP Client indication

13.5.6 HTTP Client Body Ind (ICT_HIF_CMD_ST_HTTP_BODY_IND)

Indicated values	Description

buf = body	indicates received body data.
buf_len = size	indicates the length of body data.

Table 13-4. Indication values of HTTP Client Body indication

13.6 DNS Server

13.6.1 DNS Server Start (ict_api_dns_start_handler)

Function start DNS Server.
Prototype INT32 ict_api_dns_start_handler(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.6.2 DNS Server Stop (ict_api_dns_stop_handler)

Function stop DNS Server.
Prototype INT32 ict_api_dns_stop_handler(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.7 OTA

13.7.1 OTA Version Check (ict_api_ota_ver_check)

Function check the lastest firmware version at OTA server (using HTTP).
Prototype INT32 ict_api_ota_ver_check(UINT8 *url, UINT32 url_len)
Arguments url URL of OTA firmware location

<ip/domain>:<port><url> ip/domain : domain name or IP address of OTA server port : port number of OTA server url : URL Ex.) http://ota.domain.com:8080/ota/ url_len is the length of URL.
Return 0 : Success / Otherwise, Failure

13.7.2 OTA Version Ind (ICT_HIF_CMD_ST_OTA_VERSION_FIN_IND)

Indicated values	Description
ota_version	indicates the latest OTA Version. ict_api_ota_process_ver_check_finished() function should be used to get the latest version of OTA.

Table 13-5. Indication values of OTA Version indication

13.7.3 OTA Req (ict_api_ota_request)

Function request to download firmware at OTA server (using HTTP)
Prototype INT32 ict_api_ota_request (UINT8 *url, UINT32 url_len)
Arguments url <ip/domain>:<port><url> ip/domain : domain name or IP address of OTA server port : port number of OTA server url : URL Ex.) http://ota.domain.com:8080/ota/ url_len is the length of URL.
Return 0 : Success / Otherwise, Failure

13.7.4 OTA Update Ind (ICT_HIF_CMD_ST_OTA_UPDATE_FIN_IND)

Indicated values	Description
ota_ind_result	indicates the result of OTA update.

	ict_cm_app_ota_update_fin_ind() function should be used to get the result of OTA update.

Table 13-6. Indication values of OTA Update indication

13.8 UPNP

13.8.1 External IP Req (ict_api_upnp_get_external_ip)

Function get external IP address of AP using UPNP.
Prototype INT32 ict_api_upnp_get_external_ip(void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.8.2 External IP Ind (ICT_HIF_CMD_ST_UPNP_EXTIP_IND)

Indicated values	Description
external_ip	

Table 13-7. Indication values of UPNP External IP indication

13.8.3 Add Port-mapping (ict_api_upnp_add_portmapping)

Function add portmapping in AP using UPNP
Prototype INT32 ict_api_upnp_add_portmapping(UINT8 *ipaddr, UINT16 port_internal, UINT16 port_external, UINT16 protocol, UINT8 *description)
Arguments ipaddr is a pointer to IP address of client port_internal is a internal port number port_external is an external port number protocol 1 : TCP / 2 : UDP description

is a portmapping desciption (optional)
Return 0 : Success / Otherwise, Failure

13.8.4 Add Port-mapping Ind (ICT_HIF_CMD_ST_UPNP_ADDPORTMAPPING_IND)

Indicated values	Description
result	

Table 13-8. Indication values of UPNP Add Port-mapping indication

13.8.5 Del Port-mapping Req (ict_api_upnp_del_portmapping)

Function delete portmapping in AP using UPNP
Prototype INT32 ict_api_upnp_del_portmapping (UINT16 port_external, UINT16 protocol)
Arguments port_external is an external port number protocol 1 : TCP / 2 : UDP
Return 0 : Success / Otherwise, Failure

13.8.6 Del Port-mapping Ind (ICT_HIF_CMD_ST_UPNP_DELPORTMAPPING_IND)

Indicated values	Description
ota_ind_result	indicates the result of OTA update. ict_cm_app_ota_update_fin_ind() function should be used to get the result of OTA update.

Table 13-9. Indication values of UPNP Del Port-mapping indication

13.9 DDNS

13.9.1 External IP Req (ict_api_ddns_get_external_ip)

Function get external IP address of AP using DDNS
Prototype

INT32 ict_api_ddns_get_external_ip (void)
Arguments N/A.
Return 0 : Success / Otherwise, Failure

13.9.2 External IP Ind (ICT_HIF_CMD_ST_DDNS_GETIPADDR_IND)

Indicated values	Description
external_ip	
result	

Table 13-10. Indication values of DDNS External IP indication

13.9.3 Update Information (ict_api_ddns_update_info)

Function request DDNS UPDATE GET message to DDNS server.
Prototype INT32 ict_api_ddns_update_info (UINT8 ddns_server, UINT8 *host_name, UINT8 *user_id, UINT8 *user_pw, UINT32 timer)
Arguments ddns_server select DDNS server 0 : noip.com 1 : dyndns.com host_name host names that you wish to update user_id user name (or id) of DDNS server user_pw password of DDNS server timer DDNS repetition update period (Unit : minute) 0 – 40320 minutes (default value = 28 days), or value
Return 0 : Success / Otherwise, Failure

13.9.4 Update Information Ind (ICT_HIF_CMD_ST_DDNS_UPDATE_IND)

Indicated values	Description
result	0 : Success - good: Completed Update 1 : Success - nochg: Although update is completed, the IP address is not changed 2 : Failure - Nohost: The hostname does not exist in your account 3 : Failure - Badauth: Wrong username or password 4 : Failure - Badagent: Agent sent incorrect Http request format 5 : Failure - !donator: You have specific options which require a service fee 6 : Failure - Abuse: The hostname is blocked due to abuse 7 : Failure - 911: Service Maintenance is in progress. Please contact with DynDns Support 8 : Failure - Unknown Error
external_ip	

Table 13-11. Indication values of DDNS Update indication

13.10 LPD

13.11 GMMP

13.11.1 Register (ict_api_gmmp_register_handler)

Function register M2M GW to OMP.
Prototype INT32 ict_api_gmmp_register_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf indicates type only. [type] 0 : register M2M GW to OMP. buf_len

is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.11.2 Unregister (ict_api_gmmp_unregister_handler)

Function deregister M2M GW from OMP.
Prototype INT32 ict_api_gmmp_unregister_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf indicating type only. [type] 0 : deregister M2M GW from OMP. buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.11.3 Set information to send (ict_api_gmmp_send_info_handler)

Function set information to send gmmp control frame or data frame.
Prototype INT32 ict_api_gmmp_send_info_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf indicates type only for data frame or consists of type and result for control frame. [type] type of control, or the control value in control request message type of data 1 : Collect Data 2 : Alarm Data 3 : Event Data 4 : Alarm Clear Data [result] the result of the control request message

buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.11.4 Send control frame (ict_api_gmmp_send_ctrl_handler)

Function send a GMMP control data.
Prototype INT32 ict_api_gmmp_send_info_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf [data] GMMP data buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.11.5 Event Ind (ICT_HIF_CMD_ST_GMMP_IND)

Indicated values	Description
result	
GMMP Type	
Result codes	

Table 13-12. Indication values of GMMP event indication from OMP

13.11.6 Send data frame (ict_api_gmmp_send_data_handler)

Function issues GMMP_Packet_Delivery_Request to OMP.
Prototype INT32 ict_api_gmmp_send_data_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf [data] GMMP data buf_len is the length of buffer.

Return 0 : Success / Otherwise, Failure

13.11.7 Data Ind (ICT_HIF_CMD_ST_GMMP_RECV_DATA_IND)

Indicated values	Description
GMMP Type	
length	
data	

Table 13-13. Indication values of GMMP data indication from OMP

13.12 MQTT

13.12.1 Set Configuration (ict_api_sta_mqtt_set)

Function Set MQTT Configuration.
Prototype INT32 ict_api_sta_mqtt_set (UINT32 type, UINT8 *buf, UINT32 buf_len)
Arguments type 0: Server IP 1: Server Port 2: TLS version (2: TLS1.0, 4: TLS1.2) 4: User Name 5: Password 6: Publish Topic 7: Subscribe Topic 8: Publish QoS 9: Subscribe QoS 10: Client ID 11: Connection Persistense (2: always on) 12: will topic 13: will message 14: keepalive period buf data buf_len is the length of buffer.

Return
0 : Success / Otherwise, Failure

13.12.2 Get Configuration (ict_api_sta_mqtt_get)

```
typedef struct PACKED
{
#if (MQTT_MIB_V2)
    UINT8   server_ip[32];
    UINT16  port;
    UINT8   ssl;
    UINT8   user_name[32];
    UINT8   password[64];
    UINT8   pub_topic[64];
    UINT8   sub_topic[64];
    UINT8   pub_qos;
    UINT8   sub_qos;
    UINT8   client_id[32];
    UINT8   persistence;
    UINT8   will_topic[32];
    UINT8   will_msg[64];
    UINT16  keepalive;
    UINT8   reserved[120];
#elif (MQTT_MIB_V3)
    UINT8   server_ip[32];
    UINT16  port;
    UINT8   ssl;
    UINT8   user_name[48];
    UINT8   password[64];
    UINT8   pub_topic[128];
    UINT8   sub_topic[128];
    UINT8   pub_qos;
    UINT8   sub_qos;
    UINT8   client_id[32];
    UINT8   persistence;
    UINT8   will_topic[32];
    UINT8   will_msg[32];
    UINT16  keepalive;
    UINT8   reserved[8];
#else
    UINT8   server_ip[32];
    UINT16  port;
    UINT8   ssl;
    UINT8   user_name[32];
    UINT8   password[32];
    UINT8   pub_topic[64];
    UINT8   sub_topic[64];
    UINT8   pub_qos;
    UINT8   sub_qos;
    UINT8   client_id[20];
    UINT8   persistence;
    UINT8   will_topic[32];
    UINT8   will_msg[32];
    UINT16  keepalive;
```

```
#endif
} T_NMS_MQTT_MIB;
```

Function Get MQTT Configuration.
Prototype T_NMS_MQTT_MIB *ict_api_sta_mqtt_get(void)
Arguments
Return structure `T_NMS_MQTT_MIB`

13.12.3 Publish (ict_api_mqtt_pub_handler)

Function publish messages to MQTT server.
Prototype INT32 ict_api_mqtt_pub_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf consists of type and value. [type] Mandatory 3 : Message Optional 0 : Server IP or URL 1 : Server Port 2 : SSL 4: User name 5: Password 6: Publish Topic buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.12.4 Publish Ind (ICT_HIF_CMD_ST_MQTT_PUB_IND)

Indicated values	Description

result	
result code	
error code	

Table 13-14. Indication values of MQTT Publish indication

13.12.5 Subscribe (ict_api_mqtt_sub_handler)

Function subscribe messages from MQTT Server (Broker).
Prototype INT32 ict_api_mqtt_sub_handler(UINT8 *buf, UINT32 buf_len)
Arguments buf consists of type and value. [type] Optional 0 : Server IP or URL 1 : Server Port 4: User name 5: Password 6: Subscribe Topic buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.12.6 Subscribe Ind (ICT_HIF_CMD_ST_MQTT_SUB _IND)

Indicated values	Description
result	OK or ERROR
result code	
error code	

Table 13-15. Indication values of MQTT Subscribe indication

13.12.7 Subscribe Receive Ind (ICT_HIF_CMD_ST_MQTT_SUB_RECV_IND)

Indicated values	Description
Length	MAX size : 1024
data	

Table 13-16. Indication values of MQTT Subscribe Receive indication

13.13 SEP2.0

13.14 TCP_SSL

13.15 GCM

13.16 COAP

13.17 ALLJOYN

13.18 SNTP

13.19 OneM2M (LG U+)

13.19.1 Set Configuration (ict_api_sta_onem2m_set)

Function Set OneM2M Configuration.
Prototype INT32 ict_api_sta_onem2m_set (UINT32 type, UINT8 *buf, UINT32 buf_len)
Arguments type 0: MEF server 1: OGS server buf data buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.19.2 Get Configuration (ict_api_sta_onem2m_get)

```
typedef struct PACKED
{
    UINT8   mef_server[64];
    UINT8   ogs_server[64];
    UINT8   csr_RID[64];
    UINT8   csr_RN[32];
    UINT8   nod_RN[32];
```

```
    UINT8   acp_RN[32];
    UINT8   fw_RN[32];
    UINT8   uuid[40];
    UINT8   fw_name[20];
    UINT8   reserved[132];          // 512 bytes
} T_ONEM2M_MIB;
```

Function Get OneM2M Configuration.
Prototype T_ONEM2M_MIB *ict_api_sta_onem2m_get(void)
Arguments
Return structure `T_ONEM2M_MIB`

13.19.3 Event Ind (ict_cm_app_onem2m_ind)

Indicated values	Description
Message	CONFIG CERTIFICATE AUTHENTICATE CONNECT DISCONNECT SEND SUBSCRIBE UNSUBSCRIBE UUID
status	OK/ERROR
Result codes	ret code/error code

Table 13-17. Indication values of OneM2M event indication

13.19.4 Send data frame (ict_api_onem2m_send_handler)

Function issues OneM2M_Packet_Delivery_Request to OGS.
Prototype INT32 ict_api_onem2m_send_handler (UINT8 *buf, UINT32 buf_len)
Arguments buf : [type] [container] [data] [type]

0: request 1: response [container] OneM2M container [data] OneM2M data Example 0 cnt-data {header":{},"type":"data","content":{}} buf_len is the length of buffer.
Return 0 : Success / Otherwise, Failure

13.19.5 Data Ind (ict_cm_app_onem2m_recv_ind)

Indicated values	Description
type	0: Request 1: Response 2: Error Response
rqi	received frame's RQI (container) from OGS
data	only type 0/1 received data from OGS server
response code	only type 2 received response code from OGS/OneM2M server

Table 13-18. Indication values of OneM2M data indication from Server

14. Useful APIs (optional)

14.1 JSON Parser

cJSON parser is used as JSON parser.

Refer to official sites named " http://json.org/" and " http://sourceforge.net/projects/cjson/".

14.2 XML Parser

iksemel is used as XML parser.

Refer to official site named " https://code.google.com/p/iksemel/".

14.3 AES Encryption & Decryption

14.3.1 Initialization (ict_api_aes_init)

Function intialize AES with key value
Prototype INT32 ict_api_aes_init(UINT8 *key)
Arguments key 16 bytes random value
Return -1 : Failure / Otherwise, Success

14.3.2 De-initialization (ict_api_aes_deinit)

Function de-intialize AES
Prototype INT32 ict_api_aes_deinit(void)
Arguments None
Return 0 : Success / Otherwise, Failure

14.3.3 Encryption (ict_api_aes_encryption)

Function encrypt data frame
Prototype INT32 ict_api_aes_encryption(char *data, const UINT16 len)
Arguments data is the value that should be encrypted. unit of 16 bytes (Padding should be needed if data frame is not unit of 16 bytes.) len is the length of data.
Return -1 : Failure / Otherwise, Success

14.3.4 Decryption (ict_api_aes_decryption)

Function decrypt data frame
Prototype INT32 ict_api_aes_decryption(char *data, const UINT16 len)
Arguments data is the value that should be decrypted. unit of 16 bytes (Padding should be needed if data frame is not unit of 16 bytes.) len is the length of data.
Return -1 : Failure / Otherwise, Success

14.4 FTP Client

14.4.1 ict_api_ftpc_set

Function set User ID or User Password.
Prototype INT32 ict_api_ftpc_set(UINT32 type, UINT8 *str)
Arguments type

0 : User Id 1 : User Password str is the value of User Id or User Password
Return -1 : Failure / Otherwise, Success

14.4.2 ict_api_ftpc_get

Function get User ID or User Password.
Prototype INT32 ict_api_ftpc_get(UINT32 type, UINT8 *str)
Arguments type 0 : User Id 1 : User Password str is the value of User Id or User Password
Return 0 : Success / Otherwise, Failure

14.5 WDS

14.5.1 ict_api_wds_info

Function get WDS Information.
Prototype INT32 ict_api_wds_info(UINT8 *p_wds_enable, UINT8 *p_wds0_addr, UINT8 *p_wds1_addr)
Arguments p_wds_enable 0 : WDS Disabled Otherwise, WDS Enabled p_wds0_addr return a mac address of WDS0 p_wds1_addr return a mac address of WDS1
Return

1 : Success / Otherwise, Failure

14.5.2 ict_api_wds_info_update

Function update WDS Information.
Prototype INT32 ict_api_wds_info_update(UINT8 wds_enable)
Arguments wds_enable set WDS enable option
Return 1 : Success / Otherwise, Failure

15. Stand-alone(Hardwired) Security Engine APIs

The Stand-alone Security Engine(SSE) aims to support the cryptographic functions used in various security protocols that will be used in SW applications.

Supported cryptographic functions are as follows.

(1) Hash Functions
   a) MD5
   b) SHA-1/SHS-2/SHA-3

(2) Ciphering Functions
   a) AES
   b) ARIA
   c) DES/TDES

15.1 Hash Functions

15.1.1 ict_api_sse_get_hash

Function
Get Hash Value

Prototype
INT32 ict_api_sse_get_hash(const char *algorithm, UINT8 *in_data, UINT8 *hash, UINT16 in_len)

Arguments

algorithm
must be one of "md5", "sha1", "sha224", "sha256", "sha3224", "sha3256", "sha3384" or "sha3512"

in_data
The input to compute the hash code for.

hash
The computed hash code.
Must be allocated to the next size.

algorithm	Original Algorithm	Hash Size
md5	MD5	16 bytes (128 bits)
sha1	SHA1	20 bytes (160 bits)
sha224	SHA2-224	28 bytes (224 bits)
sha256	SHA2-256	32 bytes (256 bits)
sha3224	SHA3-224	28 bytes (224 bits)
sha3256	SHA3-256	32 bytes (256 bits)
sha3384	SHA3-384	48 bytes (384 bits)
sha3512	SHA3-512	64 bytes (512 bits)

in_len Length of in_data
Return 0 : Success / Otherwise, Failure

15.2 Ciphering Functions

This engine uses zero padding.

All the bytes that are required to be padded are padded with zero.

15.2.1 ict_api_sse_init

Function initialize a Ciphering Functions of Stand-alone Security Engine(SSE)
Prototype int ict_api_sse_init(const char *algorithm, const char* mode)
Arguments algorithm must be one of "aes128", "aes256", "aria128", "aria256", "des" or "tdes" mode must be one of "ecb", "cbc", "cfb8", "cfb16", "cfb32", "cfb64", "cfb128", "ofb", "ctr", "ccm", "cmac" or "gcm"
Return 0 : Success / Otherwise, Failure

15.2.2 ict_api_sse_set_encryption_key

Function Set the key to be used for encryption
Prototype INT32 ict_api_sse_set_encryption_key (UINT8 *key, UINT16 length));
Arguments key is promised string be allocated to the next size

algorithm	Original Algorithm	Key Size
aes128	AES-128	16 bytes (128 bits)
aes256	AES-256	32 bytes (256 bits)
aria128	ARIA-128	16 bytes (128 bits)

aria256	ARIA-256	32 bytes (256 bits)
des	DES	8 bytes (64 bits)
tdes	TDES	24 bytes (192 bits)

length

It is the same as the above.

Return

0 : Success / Otherwise, Failure

15.2.3 ict_api_sse_set_decryption_key

Function

Set the key to be used for encryption

Prototype

INT32 ict_api_sse_set_decryption_key (UINT8 *key, UINT16 length));

Arguments

key

is promised string be allocated to the next size

algorithm	Original Algorithm	Key Size
aes128	AES-128	16 bytes (128 bits)
aes256	AES-256	32 bytes (256 bits)
aria128	ARIA-128	16 bytes (128 bits)
aria256	ARIA-256	32 bytes (256 bits)
des	DES	8 bytes (64 bits)
tdes	TDES	24 bytes (192 bits)

length

It is the same as the above.

Return

0 : Success / Otherwise, Failure

15.2.4 ict_api_sse_create_iv

Function

Random generate initial vector values.

Prototype

INT32 ict_api_sse_create_iv(UINT8 *out_iv);

Arguments

out_iv

is output buffer be allocated 16 bytes
Return 0 : Success / Otherwise, Failure

15.2.5 ict_api_sse_encryption

Function Encryption of data
Prototype INT32 ict_api_sse_encryption(UINT8 *in_data, UINT8 *out_buffer, UINT16 in_len, UINT8 *iv);
Arguments in_data The input data to encryption for. out_buffer The output encryptor object with the specified Key and initialization vector (IV). Must be allocated to the next size. If the mode is ccm, (input_length / 16 + 1) * 16 + 24 Otherwise ((input_length / 16 + 1) * 16) bytes in_len Length of in_data Iv the specified initialization vector (IV) of 16bytes or NULL
Return 0 : Success / Otherwise, Failure

15.2.6 ict_api_sse_decryption

Function Decryption of data
Prototype INT32 ict_api_sse_encryption(UINT8 *in_data, UINT8 *out_buffer, UINT16 in_len, UINT8 *iv);
Arguments in_data The input data to decryption for. out_buffer The output decryptor object with the specified Key and initialization vector (IV). Must be allocated the same size as input data

in_len Length of in_data Iv the specified initialization vector (IV) of 16bytes or NULL
Return 0 : Success / Otherwise, Failure

15.3 Example Code

```
unsigned char *input = "Here is some data to encrypt!";
int in_length = strlen((char *)input) ;
unsigned char *output = (unsigned char *)malloc( (in_length / 16 + 1) * 16 );
unsigned char *allocinput = (unsigned char *)malloc( (in_length / 16 + 1) * 16 );
unsigned char hashcode[16];

ict_sse_init("aes256", "cbc");
ict_api_sse_set_encryption_key ((UINT8 *)"ABCDEFGHIJKLMNOPQRSTUVWXYZ123456", 32);
ict_api_sse_set_decryption_key ((UINT8 *)"ABCDEFGHIJKLMNOPQRSTUVWXYZ123456", 32);

ict_sse_encryption(input, output, in_length, NULL);
// expect output = {0x55,0x89,0x88,0x58,0x03,0x87,0xe4,0xa9,0xe1,0xe9,0xbf,0x12,0xc0,0x4c,0x06,0x76,0xd4,
//                  0xf5,0x48,0x1d,0xf1,0x7b,0x8c,0xab,0x38,0x82,0x69,0xd9,0x17,0xc5,0x5d,0x86}


ict_sse_decryption(output, allocinput, ( (in_length / 16 + 1) * 16, NULL);
// expect memcmp(input, allocinput, in_length) == 0

ict_sse_get_hash("md5", input, hashcode, in_length);
// expect hashcode = { 0x97,0x23,0xE5,0xFE,0x7C,0xA0,0x80,0xB1,0x11,0x94,0xD3,0xE9,0x95,0xFA,0x76,0xEC }

free(output);
free(allocinput)
```

16. Appendix