MediaTek LinkIt™ Development Platform for RTOS APB Proxy Developer's Guide

Version: 1.0

Release date: 13 October 2017

Document Revision History

Revision	Date	Description
1.0	13 October 2017	Initial release

Table of contents

Lists of tables and figures

1. Introduction

1.1. Overview

MediaTek MT2625 platform provides a modem domain for protocol implementation and an application domain to develop applications with narrowband Internet of Things technology.

The AT commands or user data from external chip or PC tool arrive at the modem domain through UART interface. To enable AT command and user data exchange between external application chip and application domain on the MT2625 chipset, an AP Bridge feature is introduced that includes an AP Bridge subsystem in the ATCI module (modem domain), an AP bridge task (modem domain) and an AP bridge proxy (application domain).

This development guide provides

- An AP bridge architecture
- How to use the AP Bridge proxy interface to add custom AT commands
- How to transfer user data between external application chip or PC tool and MT2625 application domain

The software architecture on MT2625 application domain is shown in Figure 1.

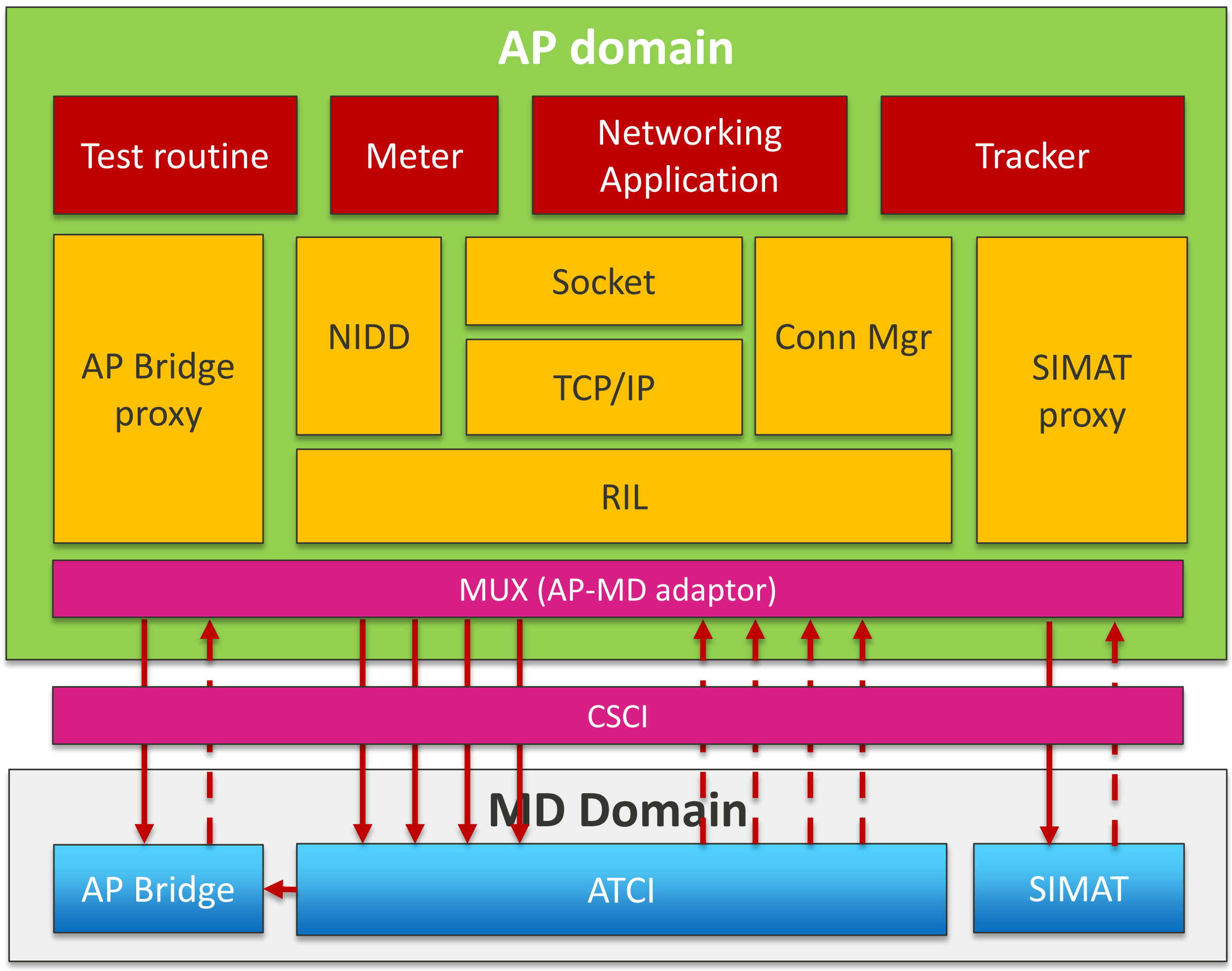

Figure 1. MT2625 software architecture in application domain

AT command's data are transferred on channels virtualized in MT2625 platform. For a channel, there are command mode and data mode. By default, it is in command mode.

- Command mode is used to send AT command and its response.
- Data mode is used to transfer user data between external application or PC tool and MT2625 application domain. Data transfer performance is better in data mode than in command mode.

1.2. AP bridge architecture

The AP bridge architecture is shown in Figure 2. The external application chip or PC tool can send different AT commands at the same time on different channels.

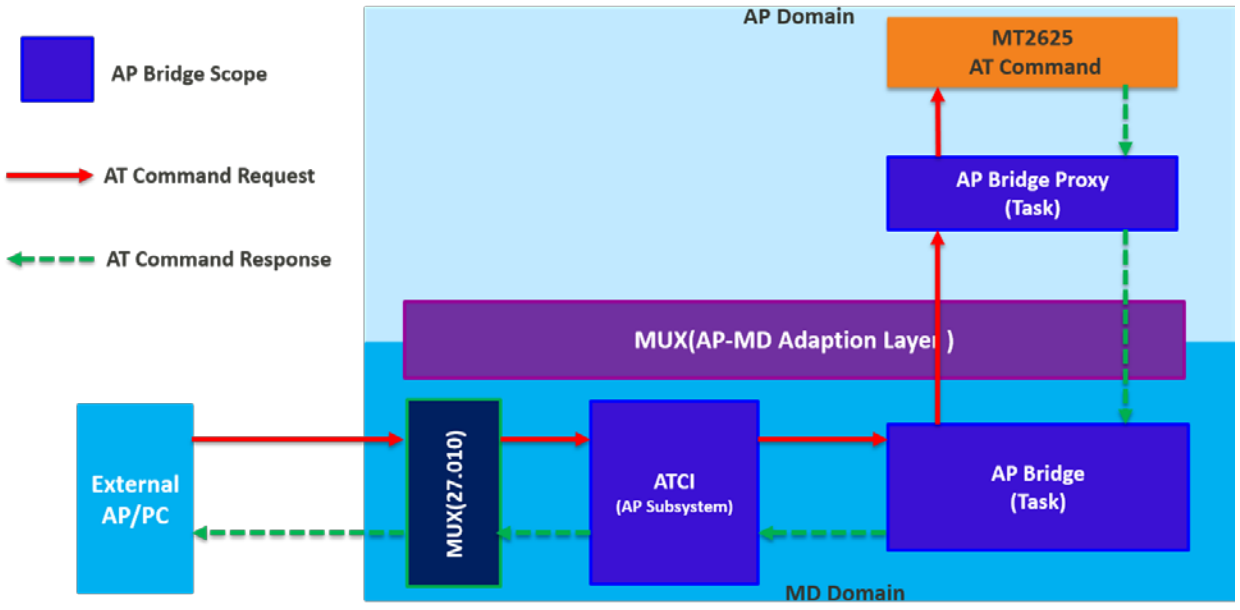

Figure 2. AP bridge architecture

1.3. AP bridge data flow in command mode

The command flow is shown in Figure 3. It describes the data flow with a custom command input implemented in the application domain. The red line indicates the data flow for an AT command request. The green dotted line describes the data flow for an AT command response, as described below:

1) External application chip or PC tool sends an AT command through UART, then then the command is sent to the multiplexer (3GPP Technical Specification 27.010).
2) The ATCI parses the command and transfers the parsed command data to AP Bridge subsystem. The AP bridge subsystem in the ATCI transfers the parsed command data to AP Bridge task in modem domain.
3) AP bridge task encapsulates the AT command data into packets and forwards the packets to MUX-Adapter.
4) MUX-Adapter forwards the packets to AP bridge proxy in application domain.
5) AP bridge proxy unpacks the packets, finds and calls the command handler.
6) The AT command results are delivered to the external application chip or PC tool through the same path.

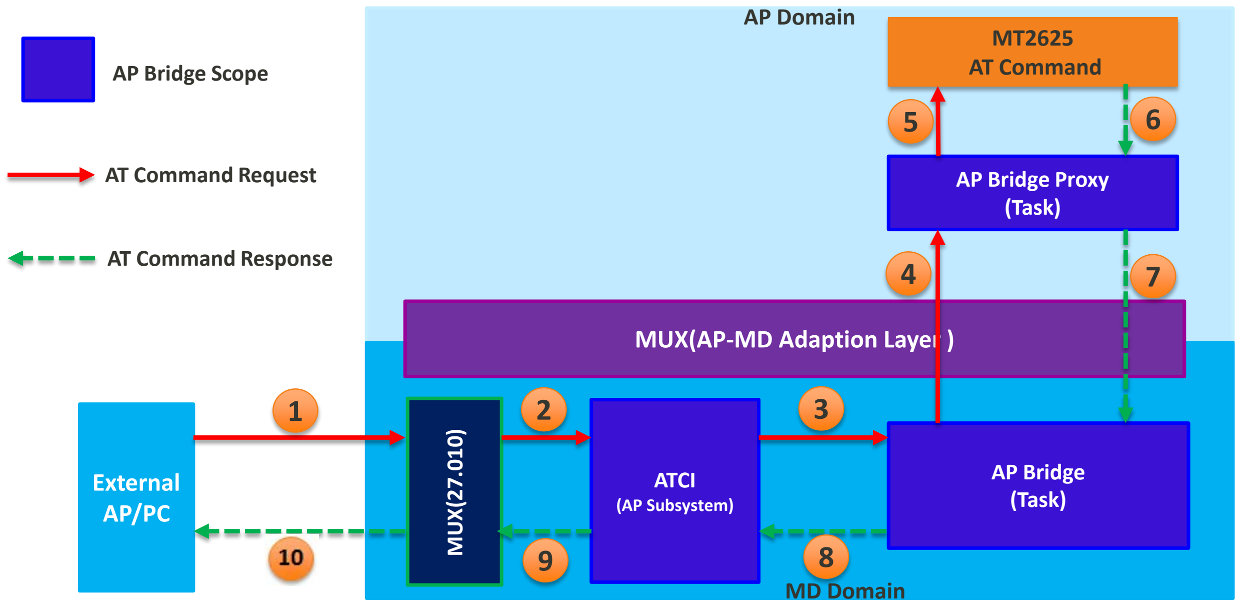

Figure 3. AP bridge data flow in command mode

1.4. AP bridge data flow in data mode

The data flow between application domain and external application chip or PC tool in data mode is shown in Figure 4. The red line indicates the user data from external application chip or PC tool to application domain. The green dotted line describes the user data from application domain to external application chip or PC tool, as described below:

1) After the data mode is created successfully, external application chip or PC tool sends user data to UART, and UART forward the data to multiplexer (3GPP Technical Specification 27.010).
2) MUX (3GPP TS 27.010) directly forwards the user data to AP bridge task in modem domain. Unlike the command mode, the user data does not go to ATCI module, which improves the data transmission performance.
3) AP bridge task in modem domain packs the user data and forwards the data to MUX adapter.
4) The AP bridge proxy unpacks the user data and forwards it to the application.
5) The user data from the application in application domain is transferred to the external application chip or PC tool through the same path in opposite direction.

Note, according to current design, only one command can go to data mode. Sending another command into data mode to switch the mode will result in system failure.

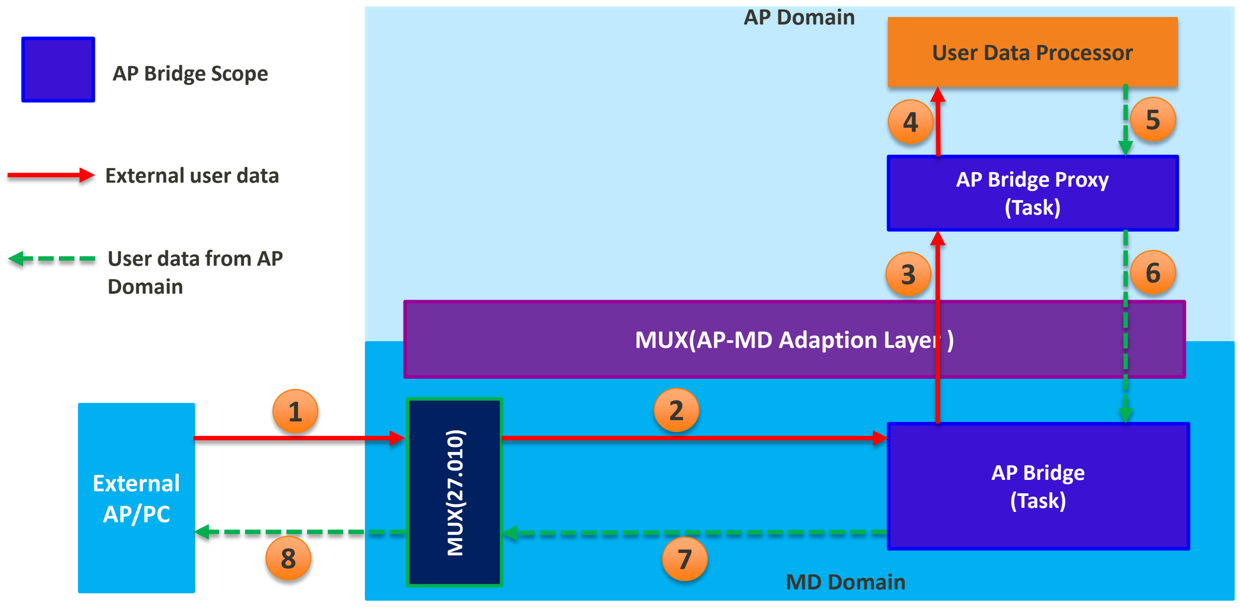

Figure 4. AP bridge data flow in data mode

2. AP bridge proxy APIs

This section describes the source code and header file hierarchy for AP Bridge proxy and its usage. The files are located at `<sdk_root>/middleware/MTK/apb_proxy.`

Table 1 provides details on the main functions of the AP bridge proxy tree hierarchy.

Table 1. AP bridge proxy source code description

File	Description
`middleware/MTK/apb_proxy/module.mk`	AP bridge proxy makefile
`middleware/MTK/apb_proxy/apb_proxy_cmd_hdlr`	This folder contains common command handler implementations.
`middleware/MTK/apb_proxy/inc/apb_proxy.h`	The public interface provided by the AP bridge proxy.
`middleware/MTK/apb_proxy/inc_private`	The internal header file to implement the AP bridge proxy.
`middleware/MTK/apb_proxy/src`	AP bridge proxy source files.

2.1. AP bridge proxy important data types

The following items list the important data structures in AP Bridge proxy interface.

2.1.1. apb_proxy_cmd_mode_t

Each command has four modes. The details are described in Table 2.

Table 2. Command mode

APB_PROXY_CMD_MODE_ACTIVE	Active mode command example "AT+CMD".
APB_PROXY_CMD_MODE_EXECUTION	Execute mode command example "AT+CMD=<op> "
APB_PROXY_CMD_MODE_READ	Read mode command example "AT+CMD?"
APB_PROXY_CMD_MODE_TESTING	Test mode command example "AT+CMD=?"

2.1.2. apb_proxy_at_cmd_result_code_t

The AT command result can be categorized into three types: final response, intermidate response and unsolicited response. APB_PROXY_RESULT_UNSOLICITED is an unsolicited response. APB_PROXY_RESULT_CONNECT, APB_PROXY_RESULT_PROCEEDING are intermidate responses. Others are final response.

Except APB_PROXY_RESULT_PROCEEDING and APB_PROXY_RESULT_UNSOLICITED , all the result codes are mapped to specific string in ATCI. For example, when the result code is APB_PROXY_RESULT_OK, the string ”OK” is output by ATCI. The result code details are shown in Table 3.

Table 3. Command result code

APB_PROXY_RESULT_OK	OK
APB_PROXY_RESULT_CONNECT	CONNECT
APB_PROXY_RESULT_RING	RING

APB_PROXY_RESULT_NO_CARRIER	NO CARRIER
APB_PROXY_RESULT_ERROR	ERROR
APB_PROXY_RESULT_NO_DIALTONE	NO DIALTONE
APB_PROXY_RESULT_BUSY	BUSY
APB_PROXY_RESULT_NO_ANSWER	NO ANSWER
APB_PROXY_RESULT_PROCEEDING	(No output string.)
APB_PROXY_RESULT_UNSOLICITED	(No output string.)
APB_PROXY_ERROR_CME_PHONE_FAILURE	+CME ERROR: phone failure
APB_PROXY_ERROR_CME_OPERATION_NOT_ALLOWED	+CME ERROR: operation not allowed
APB_PROXY_ERROR_CME_OPERATION_NOT_SUPPORTED	+CME ERROR: operation not supported
APB_PROXY_ERROR_CME_SIM_NOT_INSERTED	+CME ERROR: SIM not inserted
APB_PROXY_ERROR_CME_INCORRECT_PASSWORD	+CME ERROR: incorrect password
APB_PROXY_ERROR_CME_MEMORY_FULL	+CME ERROR: memory full
APB_PROXY_ERROR_CME_MEMORY_FAILURE	+CME ERROR: memory failure
APB_PROXY_ERROR_CME_LONG_TEXT	+CME ERROR: text string too long
APB_PROXY_ERROR_CME_INVALID_TEXT_CHARS	+CME ERROR: invalid characters in text string
APB_PROXY_ERROR_CME_NO_NETWORK_SERVICE	+CME ERROR: no network service
APB_PROXY_ERROR_CME_NETWORK_TIMEOUT	+CME ERROR: network timeout
APB_PROXY_ERROR_CME_EMERGENCY_ONLY	+CME ERROR: network not allowed - emergency calls only
APB_PROXY_ERROR_CME_UNKNOWN	+CME ERROR: unknown
APB_PROXY_ERROR_CME_PSD_SERVICES_NOT_ALLOWED	+CME ERROR: psd services not allowed
APB_PROXY_ERROR_CME_PLMN_NOT_ALLOWED	+CME ERROR: plmn not allowed
APB_PROXY_ERROR_CME_LOCATION_AREA_NOT_ALLOWED	+CME ERROR: location area not allowed
APB_PROXY_ERROR_CME_ROAMING_NOT_ALLOWED	+CME ERROR: roaming not allowed in this location area
APB_PROXY_ERROR_CME_SERVICE_OPTION_NOT_SUPPORTED	+CME ERROR: service option not supported
APB_PROXY_ERROR_CME_SERVICE_OPTION_NOT_SUBSCRIBED	+CME ERROR: requested service option not subscribed
APB_PROXY_ERROR_CME_SERVICE_OPTION_OUT_OF_ORDER	+CME ERROR: service option temporarily out of order
APB_PROXY_ERROR_CME_UNSPECIFIED_PSD_ERROR	+CME ERROR: unspecified psd error
APB_PROXY_ERROR_CME_PDP_AUTHENTIFICATION_ERROR	+CME ERROR: PDP authentication failure

2.1.3. apb_proxy_parse_cmd_param_t

This structure contains specific AT command request parameters, as shown in Table 4.

Table 4. Command request parameters

`apb_proxy_cmd_id_t  cmd_id`	AT command identifier. The command ID generated at the system start up for every AT command.
`char* string_ptr`	AT command raw data. For example: "AT+cmdtest1=1\r".
`uint32_t string_len`	AT command raw data length.
`uint32_t name_len`	AT command header's length. For example, in "`AT+EXAMPLE=1, 2, 3`", `name_len = 10 (without symbol "=")`.
`uint32_t parse_pos`	The length after detecting the AT command mode.
`apb_proxy_cmd_mode_t mode`	Command mode. For more information, please refer to `apb_proxy_cmd_mode_t`.

2.1.4. apb_proxy_at_cmd_result_t

This structure contains the command result, as shown in Table 5.

Table 5. Command result parameters

`apb_proxy_cmd_id_t cmd_id`	AT command identifier, received after a command request callback.
`apb_proxy_at_cmd_result_code_t result_code`	AT command's result code.
`void* pdata`	Extra output results except the above result code. If no extra result is needed, the field as NULL.
`uint32_t length`	Total length of the result string including '\0'. If no extra result string, the value is 0. The maximum value is `APB_PROXY_MAX_DATA_SIZE`.

2.1.5. apb_proxy_user_data_t

This structure is only used in data mode to tranfer user data. The detailed structure is shown in Table 6.

Table 6. User data structure

void* pbuffer	A pointer to the user data. The sender is responsible for memory allocation.
uint32_t length	User data length. The maximum value is `APB_PROXY_MAX_DATA_SIZE`.

2.2. AP bridge Proxy supported APIs

The following items list all the APIs provided by the AP bridge proxy.

2.2.1. apb_proxy_at_cmd_handler

Description	• This is the callback function prototype for very AT command handler. Every AT command has a specific command handler. • Restrictions: o No CPU heavy calculations are allowed. o No calls to the timer delay function in the handler are allowed. Or, it will block the AP bridge proxy task. • The user is responsible to implement this function. The function's result is reserved for future use.

Parameter	`apb_proxy_parse_cmd_param_t *p_parse_cmd`. A pointer to the specific AT command parameters when user inputs an AT command from an external application chip or PC tool.

2.2.2. apb_proxy_send_at_cmd_result()

Description	User calls this function to send an AT command result data. This is a non-blocking function and can be used for multi-tasking. ● If the function returns APB_PROXY_STATUS_OK, the AP bridge proxy guarantees the data is sent out successfully. ● If the result is APB_PROXY_STATUS_ERROR, there are four causes for the error. (1) not enough heap memory, (2) more than one command try to switch to data mode, (3) before closing the data mode, it tried to send command result, (4) invalid command ID.
Parameter	● `apb_proxy_at_cmd_result_t *presult`. A pointer to the AT command result structure. Command's response data will be included in the structure. The result type can be final response, unsolicited message, and intermediate response.

2.2.3. apb_proxy_data_mode_event_callback_t

Description	The APP registers a callback function to the APB using `apb_proxy_create_data_mode()` to transfer user data or binary data. Don't do any CPU heavy calculation.
Parameter	● `apb_proxy_event_type_t event`. Please refer to the details in `apb_proxy.h`. ● `void* pdata`. A pointer to the event data. The APB proxy is responsible for managing the memory which the pointer points to.

2.2.4. apb_proxy_create_data_mode ()

Description	If the AT command's result is "CONNECT", the APP requests APB proxy to set up data connection (data mode) to transfer user data or binary data. Before calling this function, APP must call `apb_proxy_send_at_cmd_result()` to send the command result "CONNECT". Data connection ID will be returned. If data connection is successfully created, return a non-zero value. If data connection failed to be created, return 0.
Parameter	● `apb_proxy_data_mode_event_callback_t call_back`. Event handler function for data mode. ● `apb_proxy_cmd_id_t cmd_id`. It indicates which command will try to go to data mode. Command ID can be saved when command handler is called. When system starts up, the command ID will not change.

2.2.5. apb_proxy_close_data_mode ()

Description	This function closes the data mode connection. The APP must call `apb_proxy_send_at_cmd_result()` to send the final result (APB_PROXY_RESULT_NO_CARRIER) of the AT command to close data mode connection. ● If the result is APB_PROXY_STATUS_ERROR, the cause would be: connection ID is not valid, data mode is not active, or message queue is full.

Parameter	• `apb_proxy_data_conn_id_t conn_id`. Data connection ID.

2.2.6. `apb_proxy_send_user_data ()`

Description	The APP calls this function to send user data to the APB proxy. This is a none-blocking function. • If the result is APB_PROXY_STATUS_OK, APB proxy will guarantee to send the data out. • If the result is APB_PROXY_STATUS_ERROR, the cause would be: connection is not valid, data mode is not active, or message queue is full.
Parameter	• `apb_proxy_data_conn_id_t conn_id`. Data connection ID. • `apb_proxy_user_data_t* puser_data`. A pointer to the user data structure.

2.2.7. `apb_proxy_stop_send_user_data ()`

Description	The APP requests APB proxy to stop sending user data for flow control. Call this function, if the APP cannot process more user data. This is a non-blocking function. If the result is APB_PROXY_STATUS_OK, APB proxy will guarantee to send the data out. • If the result is #APB_PROXY_STATUS_ERROR, the cause would be: connection is not valid, data mode is not active, or message queue is full.
Parameter	• `apb_proxy_data_conn_id_t conn_id`. Data connection ID.

2.2.8. `apb_proxy_resume_send_user_data ()`

Description	The APP requests APB to resume sending user data. Call this function to process more user data in the APP. This is a none-blocking function. • If the result is APB_PROXY_STATUS_OK, APB proxy will guarantee to send the data out. • If the result is APB_PROXY_STATUS_ERROR, the cause would be: connection is not valid, data mode is not active, or message queue is full.
Parameter	• `apb_proxy_data_conn_id_t conn_id`. Data connection ID.

3. Custom AT commands

3.1. Creating a custom commands handler

Each AT command has a specific command handler. The command handler prototype is `apb_proxy_at_cmd_handler`, described in `apb_proxy.h`.

An example to create AT+EXAMPLE custom command is shown below.

```
apb_proxy_status_t
apb_proxy_example_at_command_handler(apb_proxy_parse_cmd_param_t *p_parse_cmd)
 {
         configASSERT(p_parse_cmd != NULL);
         apb_proxy_at_cmd_result_t cmd_result;
         // Please note, if the output result is short, stack buffer also can
be used.
         uint8_t* p_out_buffer = NULL;
         // Initialize cmd_result.
         cmd_result.result_code = APB_PROXY_RESULT_OK; // “OK” string is sent
out by AP Bridge.
         cmd_result.pdata = NULL;
         cmd_result.length = 0;
         cmd_result.cmd_id = p_parse_cmd->cmd_id;
         switch (p_parse_cmd->mode) {
                case APB_PROXY_CMD_MODE_READ: {
                        p_out_buffer = (uint8_t *)pvPortMalloc(buffer_size);
                        memset(p_out_buffer, 0,  buffer_size);
                        // Do something. Put the custom command result data
into p_out_buffer.
                        break;
               }
              case APB_PROXY_CMD_MODE_ACTIVE: {
                       p_out_buffer = (uint8_t *)pvPortMalloc(buffer_size);
                       memset(p_out_buffer, 0,  buffer_size);
                       // Do something. Put the custom command result data into
p_out_buffer.
                       break;
             }
             case APB_PROXY_CMD_MODE_EXECUTION: {
                       p_out_buffer = (uint8_t *)pvPortMalloc(buffer_size);
                       memset(p_out_buffer, 0,  buffer_size);
```

```
                        // Do something. Put the custom command result data into
p_out_buffer.
                       break;
              }
             case APB_PROXY_CMD_MODE_TESTING: {
                        p_out_buffer = (uint8_t *)pvPortMalloc(buffer_size);
                        memset(p_out_buffer, 0,  buffer_size);
                        // Do something. Put the custom command result data into
p_out_buffer.
                       break;
             }
             case APB_PROXY_CMD_MODE_INVALID: {
                        p_out_buffer = (uint8_t *)pvPortMalloc(buffer_size);
                        memset(p_out_buffer, 0,  buffer_size);
                        // Do something. Put the custom command result data into
p_out_buffer.
                        cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
String “ERROR” will be sent out by AP Bridge.
                        break;
             }
             default: {
                        cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
String “ERROR” will be sent out by AP Bridge.
                        break;
             }
         }
        if (p_out_buffer != NULL){
               cmd_result.pdata = p_out_buffer;
               cmd_result.length = strlen(cmd_result.pdata) + 1;
        }
         if (apb_proxy_send_at_cmd_result(&cmd_result) ==
APB_PROXY_STATUS_ERROR){
                  // Failed to send the AT command result, confirm whether the
input parameter is correct.
         }
         // Free the allocated buffer.
         if (p_out_buffer != NULL){
                 vPortFree((void *)p_out_buffer);
                 p_out_buffer = NULL;
         }
```

```
 }
```

Notes to consider when creating an AT command handler:

- Always initialize all the members in `apb_proxy_at_cmd_result_t` before using it.
- When calling the interface `apb_proxy_send_at_cmd_result()` to send the result, the command ID is obtained from the callback's input parameters. The command ID is not changed after the system starts up. Thus, the user can save the command ID for future use.
- Every command must return a final response code, otherwise, the channel cannot accept the next AT command in the same channel. `APB_PROXY_RESULT_CONNECT`, `APB_PROXY_RESULT_PROCEEDING`, `APB_PROXY_RESULT_UNSOLICITED` are not final response codes listed in Table 3.
- If `apb_proxy_send_at_cmd_result()` returns an error, check whether the input parameter is correct.

3.2. Adding the command into a command table

To add the command into AT command table:

- Use a macro to map the command string with command handler:

```
AT_CMD_HDLR_ITEM_ENTRY_DEF("AT+EXAMPLE",apb_proxy_example_at_command_handler)
```

- For projects using AT commands, add the command definition into "`apb_proxy_cmd_def.h` "and "`apb_proxy_cmd_handler.h`". If the files are missing from the project, add them from an existing source.
- For commonly used AT commands, add the command definition into `g_apb_proxy_at_cmd_handler_table` in file "`apb_proxy_at_cmd_tbl.c`".
- When an external application chip or PC tool sends the command to MT2625, a callback function is executed.

3.3. Sending unsolicited messages

3.3.1. Unsolicited messages data flow

To send random unsolicited message, use APB_PROXY_RESULT_UNSOLICITED. For example, user sends AT command: AT+URCEXAMPLE=1 to create a task to randomly send a string message out. In Figure 5, the yellow dotted lines are the data path for unsolicited messages. URC test task can send unsolicited messages at any time.

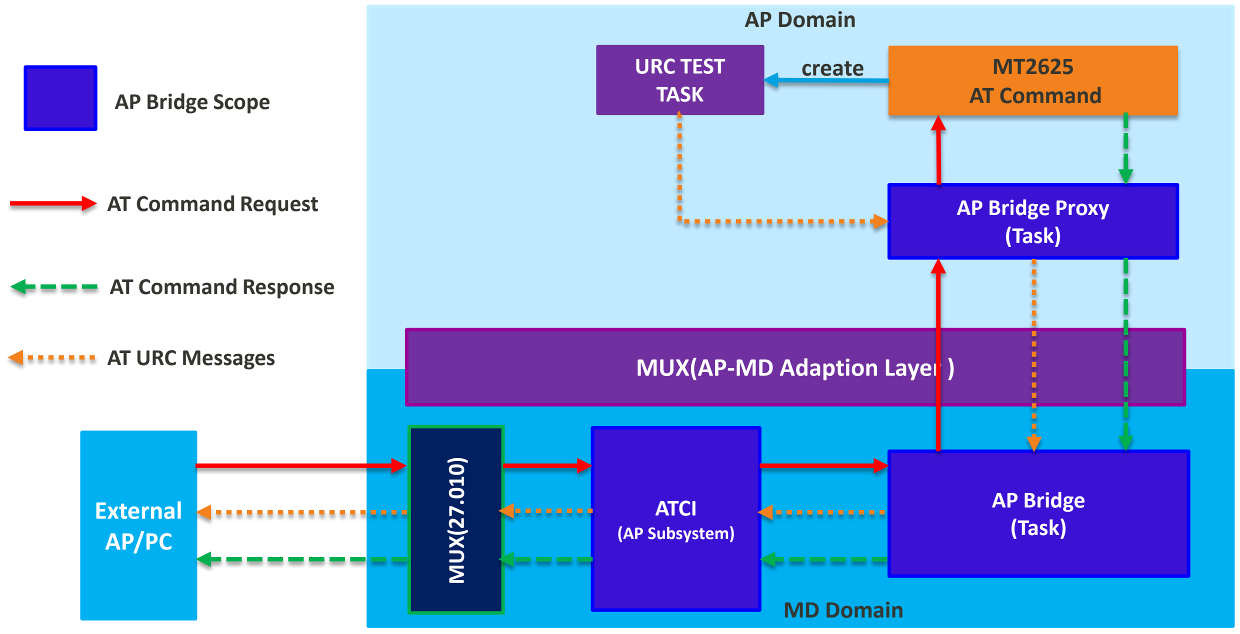

Figure 5. Unsolicited Message Data Flow

To support unsolicited messages:

- To add the command handler, please refer to section 3.1, "Creating a custom commands handler" and 3.2, "Adding the command into a command table";
- When the command handler sets up the environment, the handler returns APB_PROXY_RESULT_OK. Then, other users can input commands on this channel.
- Use `apb_proxy_send_at_cmd_result()` to send an unsolicited messages out. The result field must be APB_PROXY_RESULT_UNSOLICITED. The unsolicited message content is stored in the `pdata` in `apb_proxy_at_cmd_result_t`.
- To add the header pattern before the unsolicited message, set the corresponding command ID into the command result field. For example, the user sends "AT+URCEXAMPLE=1" to activate the unsolicited message, when unsolicited message "This is URC" is sent by `apb_proxy_send_at_cmd_result()`, the final output from MT2625 is "+URCEXAMPLE: This is URC".
- To omit the header pattern, set the command ID as APB_PROXY_INVALID_CMD_ID. For example, the user sends "AT+URCEXAMPLE=1" to active the unsolicited message in this case, when unsolicited message "This is URC" is sent by `apb_proxy_send_at_cmd_result()`. The final output from MT2625 is "This is URC".

3.3.2. Sample code for unsolicited messages

- Write a command handler to activate an unsolicited message.

```
apb_proxy_status_t
apb_proxy_urc_example_cmd_handler(apb_proxy_parse_cmd_param_t *p_parse_cmd)
 {
        configASSERT(p_parse_cmd != NULL);
        apb_proxy_at_cmd_result_t cmd_result;
        // Must firstly initialize cmd_result.
        cmd_result.result_code = APB_PROXY_RESULT_OK; // “OK” string will be
sent out by AP Bridge.
         cmd_result.pdata = NULL;
         cmd_result.length = 0;
         cmd_result.cmd_id = p_parse_cmd->cmd_id;
         switch (p_parse_cmd->mode) {
                case APB_PROXY_CMD_MODE_READ: {
                        cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
“ERROR” string will be sent out by AP Bridge.
                        break;
               }
              case APB_PROXY_CMD_MODE_ACTIVE: {
                      cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
“ERROR” string will be sent out by AP Bridge.
                      break;
             }
             case APB_PROXY_CMD_MODE_EXECUTION: {
                      create_urc_task(); // User implements the function to
create their own task.
                      cmd_result.result_code = APB_PROXY_RESULT_OK; // “OK”
string will be sent out by AP Bridge.
                      break;
              }
             case APB_PROXY_CMD_MODE_TESTING: {
                      cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
“ERROR” string will be sent out by AP Bridge.                    break;
             }
             case APB_PROXY_CMD_MODE_INVALID: {
                      cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
String “ERROR” will be sent out by AP Bridge.
                      break;
             }
             default: {
                     cmd_result.result_code = APB_PROXY_RESULT_ERROR; // String
“ERROR” will be sent out by AP Bridge.
                     break;
             }
         }
         if (apb_proxy_send_at_cmd_result(&cmd_result) ==
APB_PROXY_STATUS_ERROR){
                // Failed to send at command result.
        }
}
```

- Send the unsolicited messages:

```
void usc_test_task()
{
    uint8_t* p_urc_buffer = NULL;
    apb_proxy_at_cmd_result_t cmd_result;
    p_urc_buffer = (uint8_t *)pvPortMalloc(buffer_size);
    memset(p_urc_buffer, 0, buffer_size);
    while(1){
           // Task gets the unsolicited messages, and puts them into
p_urc_buffer.
..............................................................
           // Send an unsolicited message
           cmd_result.result_code = APB_PROXY_RESULT_UNSOLICITED;
           cmd_result.pdata = p_urc_buffer;
           // The user should calculate the result length.
           cmd_result.length = result_length;
           // cmd_id is saved when command handler is called.
           cmd_result.cmd_id = cmd_id;
if (apb_proxy_send_at_cmd_result(&cmd_result) != APB_PROXY_STATUS_OK{
                  // Error handling.
           }
}
}
```

3.4. Transferring user data in data mode

To transfer large amount of binary data between external application or PC tool and application domain, use data mode. The data transfer performance is better in command mode than in data mode. Details on how to create data mode, how to transfer user data and how to return back to command mode are provided next.

3.4.1. Main steps for supporting data mode

The main steps to support data mode:

- Write the command handler. In data mode case, the user must set the result code as APB_PROXY_RESULT_CONNECT.
- The command also needs to add into command table.

- After APB_PROXY_RESULT_CONNECT is sent out by `apb_proxy_send_at_cmd_result()`, user will call `apb_proxy_create_data_mode()`, to let the command go to data mode.
- After data mode is created, user can call `apb_proxy_send_user_data()` to send user data out.
- The user data from external application or PC tool is called back by the AP bridge proxy.
- To close data mode and switch back to command mode, call `apb_proxy_close_data_mode`, and then send APB_PROXY_RESULT_NO_CARRIER result code out by `apb_proxy_send_at_cmd_result`. APB_PROXY_RESULT_OK or APB_PROXY_RESULT_ERROR also can be sent out.
- Please note, only when APB_PROXY_RESULT_NO_CARRIER or APB_PROXY_RESULT_OK, APB_PROXY_RESULT_ERROR is sent out, the data mode is really closed.

3.4.2. Data mode sample code

The sample code about data mode in detail as below.

- Command handler sample code.

```
apb_proxy_status_t
apb_proxy_example_at_command_handler(apb_proxy_parse_cmd_param_t *p_parse_cmd)
 {
         configASSERT(p_parse_cmd != NULL);
         apb_proxy_at_cmd_result_t cmd_result;
         //Must firstly initialize cmd_result.
        cmd_result.result_code = APB_PROXY_RESULT_OK; // “OK” string will be
sent out by AP Bridge.
         cmd_result.pdata = NULL;
         cmd_result.length = 0;
         cmd_result.cmd_id = p_parse_cmd->cmd_id;
         switch (p_parse_cmd->mode) {
                 case APB_PROXY_CMD_MODE_READ: {
                         cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
“ERROR” string will be sent out by AP Bridge.
                         break;
               }
              case APB_PROXY_CMD_MODE_ACTIVE: {
                       cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
“ERROR” string will be sent out by AP Bridge.
                      break;
             }
             case APB_PROXY_CMD_MODE_EXECUTION: {
                      // “CONNECT” string will be sent out by AP Bridge.
                      cmd_result.result_code = APB_PROXY_RESULT_CONNECT;
                     break;
```

```
                }
              case APB_PROXY_CMD_MODE_TESTING: {
                       cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
“ERROR” string will be sent out by AP Bridge.          break;
              }
              case APB_PROXY_CMD_MODE_INVALID: {
                        cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
String “ERROR” will be sent out by AP Bridge.
                        break;
              }
              default: {
                        cmd_result.result_code = APB_PROXY_RESULT_ERROR; //
String “ERROR” will be sent out by AP Bridge.
                        break;
              }
          }
          if (apb_proxy_send_at_cmd_result(&cmd_result) ==
APB_PROXY_STATUS_ERROR){
               // Failed to send at command result.
          }
}
```

- Add the command handler into the command table.

```
AT_CMD_HDLR_ITEM_ENTRY_DEF("AT+DATAEXAMPLE",
apb_proxy_example_at_command_handler)
```

- Write the data mode event callback function.

```
static void app_data_mode_event_call_back(apb_proxy_event_type_t event, void
*pdata)
{
     apb_proxy_at_cmd_result_t cmd_result;
     switch (event) {
         case APB_PROXY_USER_DATA_IND: {
                  // Process user data sent from external application or PC
tool.
                 break;
         }
         case APB_PROXY_STOP_SEND_USER_DATA: {
                 // Flow control event.
                 break;
```

```
            }
            case APB_PROXY_RESUME_SEND_USER_DATA: {
                    // Flow control event.
                    break;
            }
            default: {
                    configASSERT(0);
                    break;
            }
        }
}
```

- Create data mode, transfer user data and close data mode.

Before executing the following code, ensure the “CONNECT” result is sent out to the AP bridge proxy.

```
apb_proxy_data_conn_id_t conn_id = 0;conn_id =
apb_proxy_create_data_mode(app_data_mode_event_call_back, cmd_id);
if (conn_id == 0){
// Data mode creation is failed. Only one command can go to data mode at the
same time.
}
if (apb_proxy_send_user_data(conn_id, pdata) != APB_PROXY_STATUS_OK){
    // Send user data failed.
}
// Close the data mode, and return to command mode.
if (apb_proxy_close_data_mode(conn_id) != APB_PROXY_STATUS_OK){
       // Close data mode failed.
}
// Send final result for this command.
apb_proxy_at_cmd_result_t cmd_result;
cmd_result.result_code = APB_PROXY_RESULT_NO_CARRIER;
cmd_result.pdata = NULL;
cmd_result.length = 0;
cmd_result.cmd_id = cmd_id;
if (apb_proxy_send_at_cmd_result(&cmd_result) == APB_PROXY_STATUS_OK){
       // Send the result out successfully.
}else{
      // Failed to send the result out.
}
```