

The MUT SDK User Manual for Linux

v2.2.38



November 20, 2024

Contents

1.	Introduction.....	1
2.	The MUT SDK Architecture.....	3
2.1	SDK Block Diagram	3
2.2	SDK with Hardware.....	4
2.3	SDK Software Architecture	5
3.	MUT_SDK.....	7
3.1	Software Operation	7
3.1.1	Decompress the MUT_SDK	7
3.1.2	Source Code Structure	8
3.2	Standalone Program	9
3.2.1	mcu.ap	10
3.2.2	mcu_menu_ap	11
3.2.3	API Function	12
3.2.4	Demo Screenshot	13
3.3	Client/Server Framework Program	14
3.3.1	Server	14
3.3.2	Client.....	15
3.3.3	API Function	17
3.3.4	Demo Screenshot	18
3.4	Common API Table	19
4.	Flash_MCU.....	21
4.1	Software Operation	21
4.1.1	Decompress the Flash_MCU	21
4.1.2	Source Code Structure	21
4.2	Flash MCU/CAN	22
	Appendix A – Histories.....	23

1. Introduction

The **MUT (MCU Utility Tools) SDK** is based on service/client framework. The MUT service of the MUT SDK can serve multiple client programs developed using client API at the same time. Besides, the MUT SDK also can support standalone program if you didn't want to use service/client framework. So far, it already supports a lot of MCS products.

The MUT SDK can control peripherals devices or I/O using API provided. It also supports our own CAN module simulated using MCU device.

There are two files in the MUT SDK, **MUT_SDK** and **FLASH_MCU**. Both of them are briefly described as follows. More of information will expose in chapter 3 and chapter 4.

- **MUT_SDK:** Most important part of MUT SDK. In the files, you can use tools to get/set MCU and CAN data.
- **Flash_MCU:** Use to update MCU firmware.

1.1 Support Product

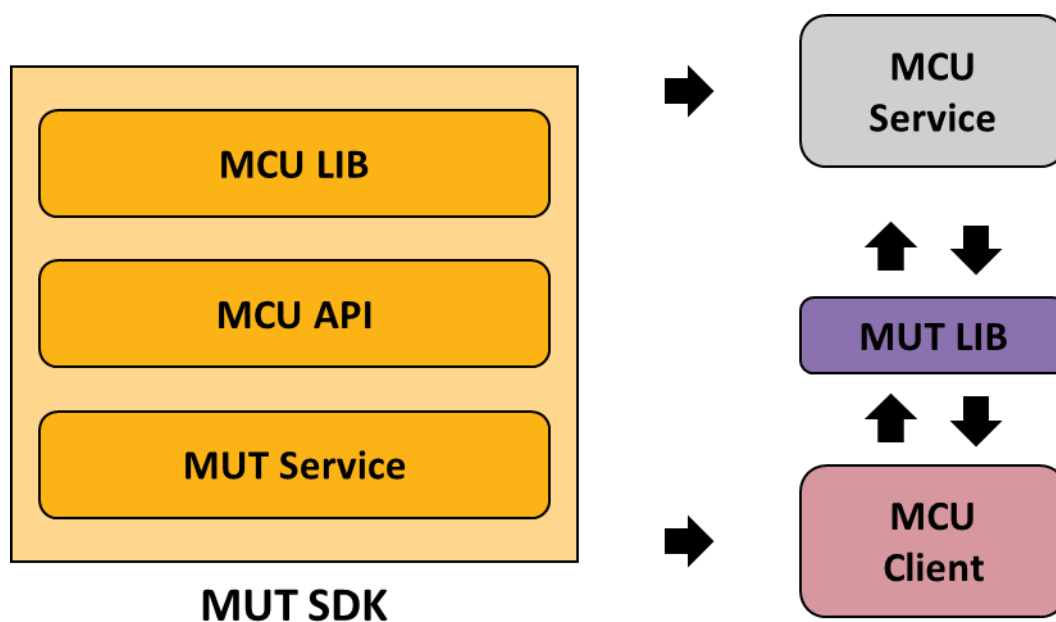
Supported Product

ALITS-2121ES、ATC3200、ATC-8010、ATC8110、ATC3530
MVS2623、MVS5200、MVS5603_MVS5600
nROK1020-R、nROK-7251、nROK7252、nROK1031、nROK6231-ST、nROK7270
aROK5510、aROK8110、ATC3750、ATC3750-8M、ATC3540、ATC3520、X80
VMD3002、VMC-2020、VMC3020、VMC3021、VMC4020、VMC320、VMC1110
VTC-1011、VTC1020、VTC1021、VTC1910、VTC1911、VTC1920、VTC7270、
VTC1031
VTC6220、VTC6221、VTC-6222、VTC-7250、VTC-7251、VTC-7252、VTC-7260
TX64-R131、nROK6221-IP、VIOB-POE2-01(POE board)
VTC1030、VTC6231、VTC6221-NMD、VTC6232
vROK3030
VTK62B、VTKSCAP

2. The MUT SDK Architecture

2.1 SDK Block Diagram

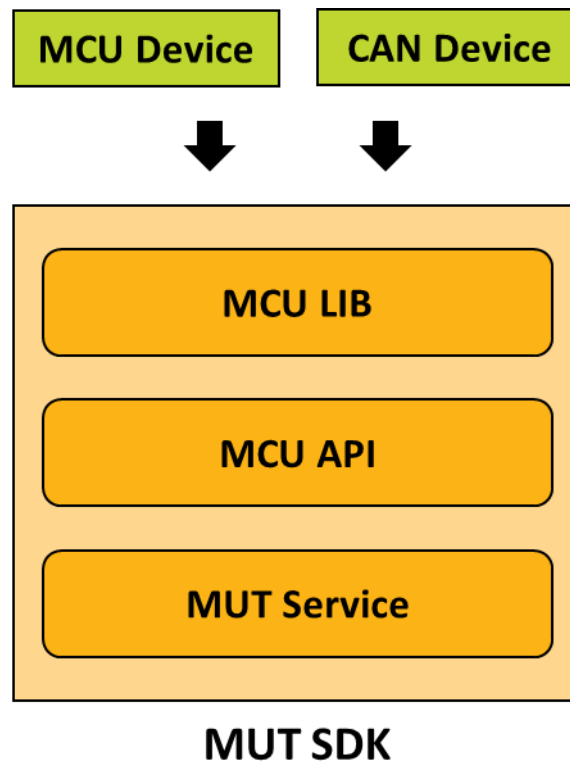
The MUT SDK (yellow block) has **MCU LIB** , **MCU API** and **MUT Service**. It can develop **standalone** or **client/server framework** program to get info by using MUT SDK. Picture 1 presents a simple client/server framework program by using MUT SDK. In the picture (see picture 1), you will understand the contents of MUT SDK, and how the MCU service and client co-work clearly.



Picture 1 - SDK Block Diagram

2.2 SDK with Hardware

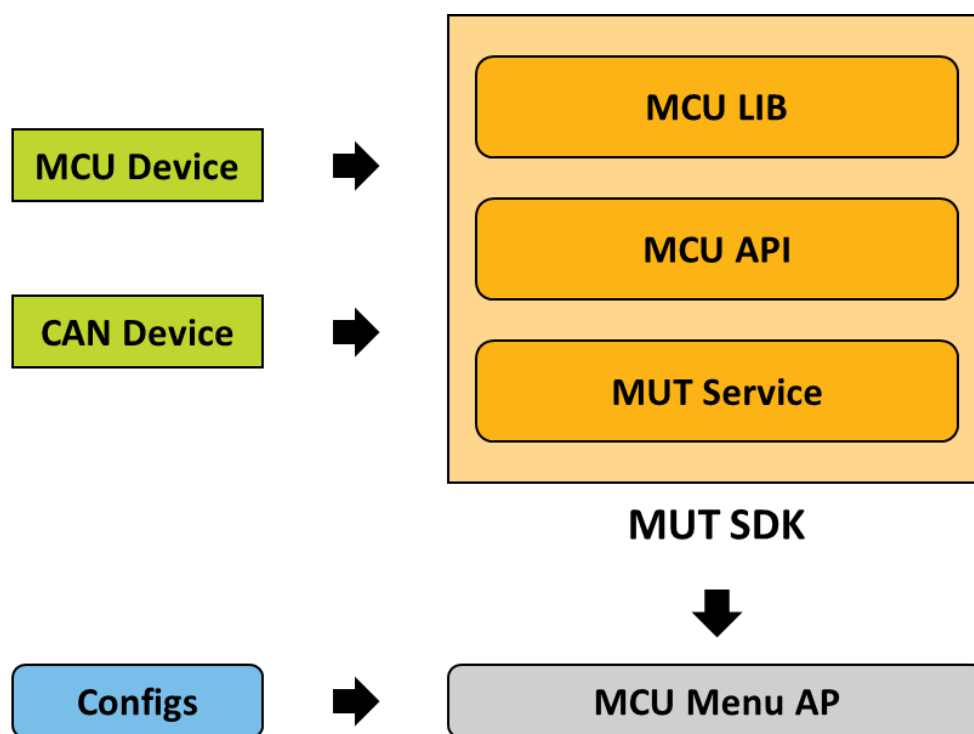
You can get MCU and CAN info by MUT SDK while connect corresponding device.
(see [picture 2](#))



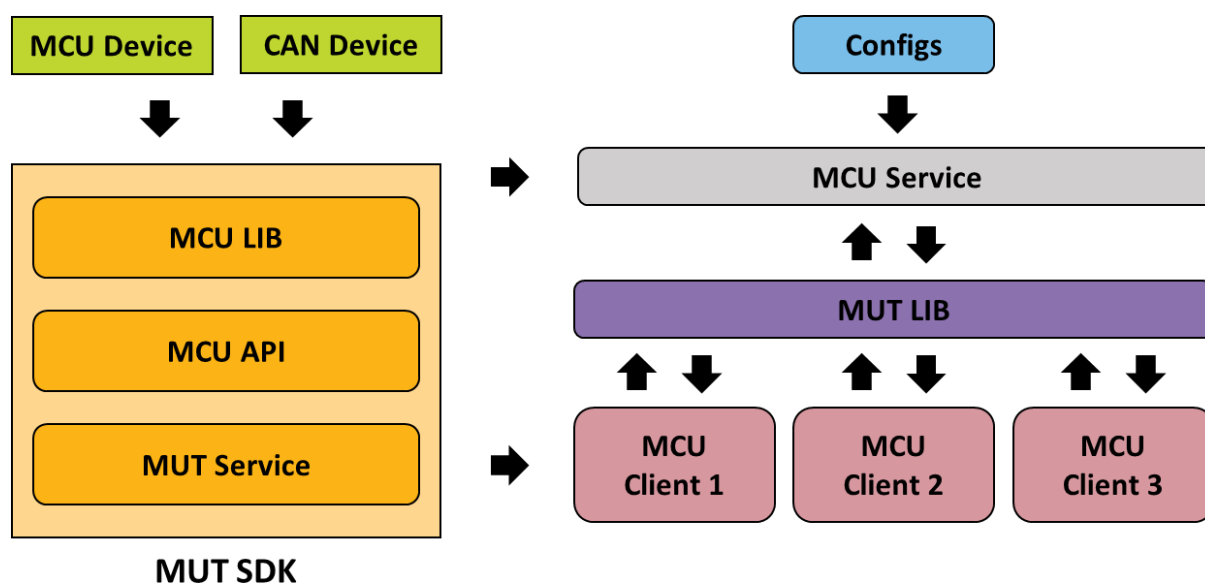
Picture 2 - SDK with Hardware Diagram

2.3 SDK Software Architecture

MUT SDK provides two software architecture for development. One is **standalone program** (see [picture 3](#)), the other is **Client/Server Framework program** (see [picture 4](#)). Developer can use suitable software architecture for the application.



Picture 3 - Standalone Architecture Program Diagram



Picture 4 - Client/Server Framework Architecture Program Diagram

3. MUT_SDK

In this chapter, you will know how to use **MUT_SDK** step by step. Here are some tips to help developers get started quickly. In chapter 3.1, you should do some operation before using **MUT_SDK**. In chapter 3.2, you will learn about how to use our **MUT_SDK** by **standalone architecture**. In chapter 3.3, you will learn about how to use our **MUT_SDK** by **client/server framework architecture**.

3.1 Software Operation

3.1.1 Decompress the MUT_SDK

```
$tar -Jxf MUT_SDK-linux_vX.X.X.tar.xz #(X.X.X is MUT SDK version)
```

3.1.2 Source Code Structure

After decompress **MUT_SDK**, the contents of **MUT_SDK** are as below:

File Path		Description
“client” folder	Bin	Two executable demo programs “demoCAN_AP” and “demoMCU_AP” included.
	configs	The configuration files for a lot of products included. (for initiating some settings)
	include	There are some of necessary header files needed when developing your program.
	LIB_x86_64	The shared library “libMUT.so” and “libPMUT.so(for python used)” included. (You could copy shared library to system lib path)
	Src	Two demo sources named “demoCAN_AP.c” and “demoMCU_AP.c” included.
	demoMCU_AP.py	Used libPMUT.so to access MCU
	demoCAN_AP.py	Used libPMUT.so to access CAN
	Makefile	The make file building demo source code.
“service” folder	Bin	The “mcu_service” ran as MUT service (you can use launch.sh to start/stop MUT service)
	configs	The configuration files for a lot of products included. (for initiating some settings)
	include	There are some of necessary header files needed when developing your program.
	LIB_x86_64	There are some of necessary shared libraries included. (You should copy shared libraries to system lib path)
	Src	Two demo sources named “mcu_ap.c” and “mcu_menu_ap.c” included.
	launch.sh	You can use it to run MUT service. The usage is as below: bash launch.sh { start stop } [fifo name] or input “bash launch.sh” to show the detail of usage
releases		The release information of MUT SDK

3.2 Standalone Program

In Standalone architecture, you should follow the API of server side and develop the program to access MCU if you just have one program to access MCU. Please reference the demo program sources (included `mcu_ap.c` and `mcu_menu_ap.c` in `MUT_SDK-linux_vX.X.X/service/src`) to develop your own program or try running the demo program introduced under the section 3.2.1 and 3.2.2.

There is little difference between `mcu_ap.c` and `mcu_menu_ap.c`. Description is as follows:

- `mcu_ap.c`: Provide CLI Interface to use MUT SDK.
- `mcu_menu_ap.c`: Provide menu in CLI interface to use MUT SDK.

[NOTE]: Before launch any MCU application, you should check the path which has “**LIB_x86_64**”, **MUT LIB document**, and set **Environment Variable** for library path to be used by applications.

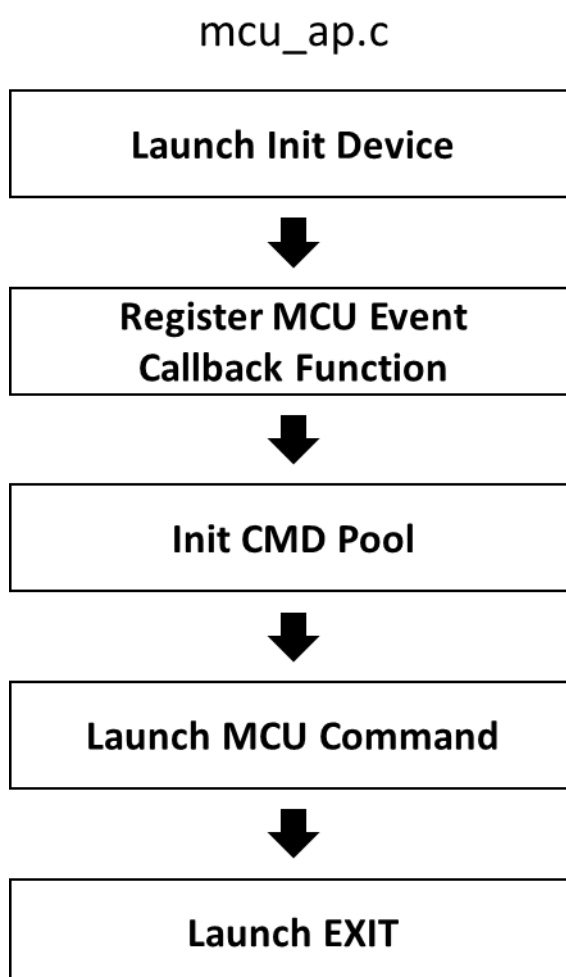
Example:

```
$cd MUT_SDK-linux_vX.X.X/service/      # go to the path
$export                                  # set environment variable
LD_LIBRARY_PATH=LIB_x86_64              # launch application
$./bin/mcu_ap
```

3.2.1 mcu.ap

In the mcu.ap, there are 5 step as below (see picture 5).

- Step 1: Launch Init Device
 In this step, you can initialize MCU or CAN device by parameter.
- Step 2: Register MCU Event Callback Function
 In this step, you can register callback function to MCU.
 MCU will send the message with corresponding data for user.
- Step 3: Init CMD Pool
 Initialize CMD pool to prepare for next step.
- Step 4: Launch MCU Command
 In this step, you can launch MCU command. All command functions can
 find in the file, “**Commands List**”.
- Step 5: Launch EXIT
 Exit the program.

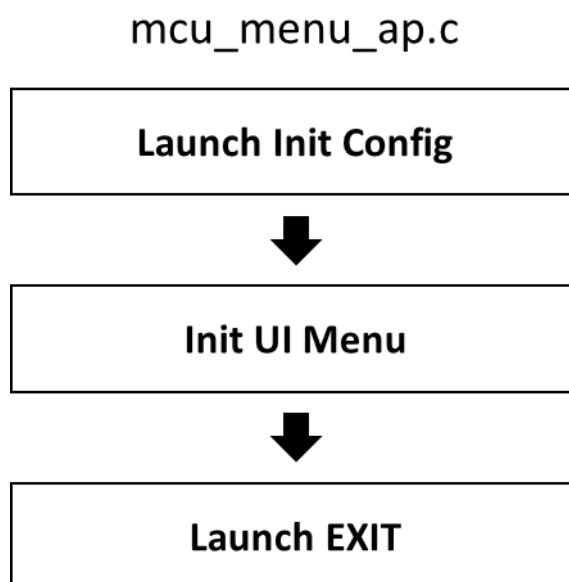


Picture 5 – mcu.ap Flow Chart

3.2.2 mcu_menu_ap

In the mcu_menu.ap, there are 3 step as below (see picture 6). Without redundant operation, you can use all MCU functions by this mcu_menu_ap. Follow the tips on cmd, you will know what MCU function can do something. Get and set information more convenience.

- Step 1: Launch Init Config
 This block will read configs base on each device.
- Step 2: Init UI Menu
 Launch UI menu. You can get/set what information by following the tips on cmd.
- Step 3: Launch EXIT
 Exit the program.



Picture 6 – mcu_menu.ap Flow Chart

3.2.3 API Function

There are API function table corresponding steps with mcu_ap and mcu_menu_ap.

More information can find from **mcu_ap.c** and **mcu_menu_ap.c** in **MUT_SDK-linux_vX.X.X/service/src**.

mcu_ap

Step		API Function
1	Launch Init Device	int LAUNCH_INIT_DEV(char *machineNm, int devType, char *dev, int baudR, int canType);
2	Register MCU Event Callback Function	int Reg_Event_CF(void (*callback)(int, BYTE, BYTE *));
3	Init CMD Pool	int INIT_CMD_POOL();
4	Launch MCU Command	int LAUNCH_MCU_CMD(char *cmdName, BYTE *dataIn, int dataInLen, BYTE *dataOut, int *dataOutLen);
5	Launch EXIT	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);

mcu_menu_ap

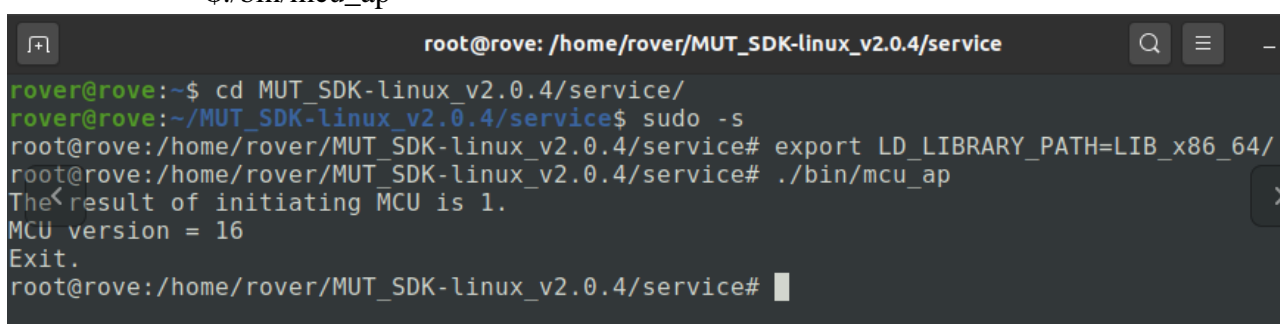
Step		API Function
1	Launch Init Config	FILE * LAUNCH_INIT_CONFIG(int argc ,char *argv[]);
2	Init UI Menu	int INIT_UI_MENU(FILE *fConfig, int (*findIdx)(char *, int *), handleFun (*findFunPtr)(char *));
3	Launch EXIT	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);

3.2.4 Demo Screenshot

You can run demo program (see picture 7 ~ 8) or your own program used the MCU API. Besides, you can modify and compile (used Make file in MUT_SDK-linux_vX.X.X/service, the command is make {mcu_ap | mcu_menu_ap}-A to compile) demo program to test and try. The screen shot as below demo the MCU client program to access MCU directly.

Script:

```
$cd MUT_SDK-linux_v2.0.4/service/
$sudo -s
$export LD_LIBRARY_PATH=LIB_x86_64/
$./bin/mcu_ap
```

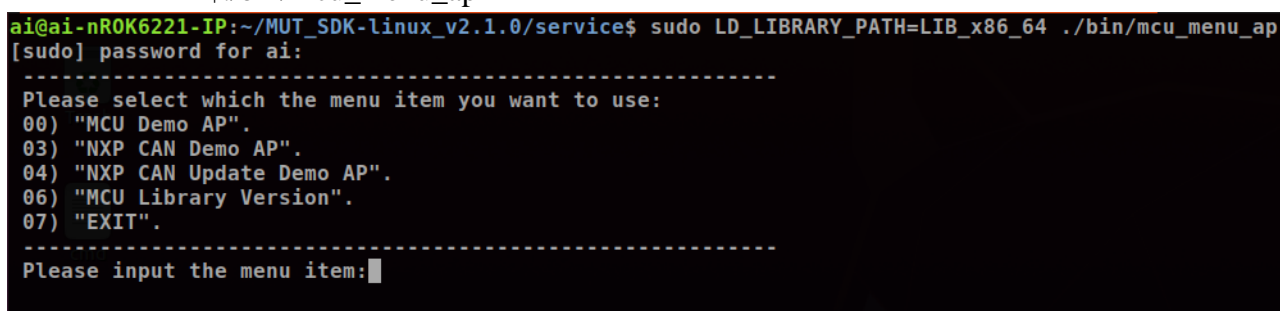


```
root@rove: /home/rover/MUT_SDK-linux_v2.0.4/service
root@rove:~$ cd MUT_SDK-linux_v2.0.4/service/
root@rove:~/MUT_SDK-linux_v2.0.4/service$ sudo -s
root@rove:/home/rover/MUT_SDK-linux_v2.0.4/service# export LD_LIBRARY_PATH=LIB_x86_64/
root@rove:/home/rover/MUT_SDK-linux_v2.0.4/service# ./bin/mcu_ap
The result of initiating MCU is 1.
MCU version = 16
Exit.
root@rove:/home/rover/MUT_SDK-linux_v2.0.4/service#
```

Picture 7 – mcu_ap demo screenshot

Script:

```
$cd MUT_SDK-linux_v2.1.0/service/
$sudo -s
$export LD_LIBRARY_PATH=LIB_x86_64/
$./bin/mcu_menu_ap
```



```
ai@ai-nR0K6221-IP:~/MUT_SDK-linux_v2.1.0/service$ sudo LD_LIBRARY_PATH=LIB_x86_64 ./bin/mcu_menu_ap
[sudo] password for ai:
-----
Please select which the menu item you want to use:
00) "MCU Demo AP".
03) "NXP CAN Demo AP".
04) "NXP CAN Update Demo AP".
06) "MCU Library Version".
07) "EXIT".
-----
Please input the menu item:
```

Picture 8 – mcu_menu_ap demo screenshot

3.3 Client/Server Framework Program

In client/server framework architecture, you should follow the API of client side and develop the client program to access MCU remotely if you have many programs to access MCU at the same time. Please reference the demo program sources (included **demoMCU_AP.c** and **demoCAN_AP.c** in **MUT_SDK-linux_vX.X.X/client/src**) to develop your own program or try running the demo program introduced under the section 3.3.1 and 3.3.2.

[NOTE]: Before launch any MCU application, you should check the path which has “**LIB_x86_64**”, **MUT LIB document**, and set **Environment Variable** for library path to be used by applications.

Example:

```
$cd MUT_SDK-linux_vX.X.X/service/      # go to the path
$export                                  # set environment variable
LD_LIBRARY_PATH=LIB_x86_64              # launch server
$./bin/mcu_service                       # go to the path
$cd MUT_SDK-linux_vX.X.X/client/        # set environment variable
$export                                  # launch client
LD_LIBRARY_PATH=LIB_x86_64
$./bin/demoMCU_AP
```

3.3.1 Server

We provide “**mcu_service**” bin file for user to launch server side service. The file is in **MUT_SDK-linux_vX.X.X/service/bin**.

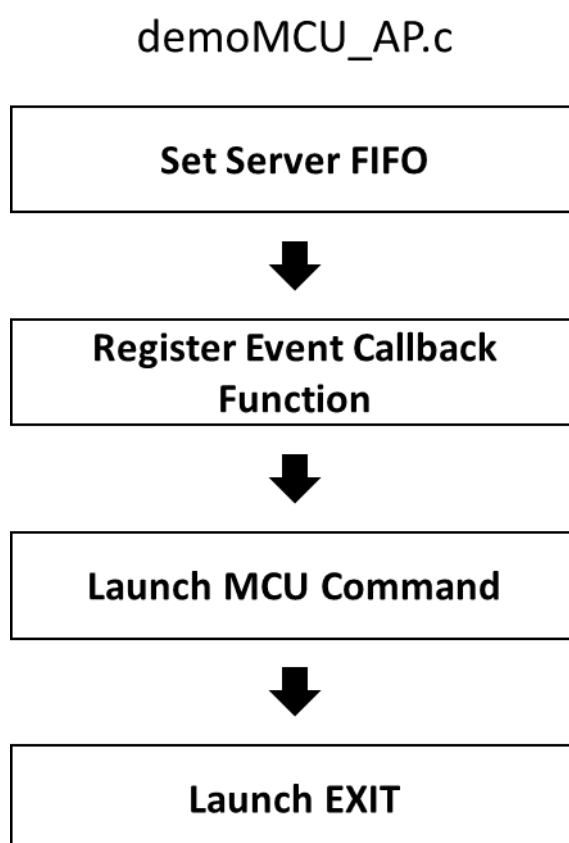
3.3.2 Client

There are two kind of methods to get data. One is for getting MCU data and the other is for getting CAN data. As below descriptions about **demoMCU_AP.c** and **demoCAN_AP.c**, you will learn how to get MCU data and CAN data in client side.

3.3.2.1 demoMCU_AP.c

In the demoMUC_AP.c, there are 4 step as below ([see picture 9](#)).

- Step 1: Set Server FIFO
In this step, you should set server FIFO to read data.
- Step 2: Register Event Callback Function
In this step, you can register callback function to MCU.
MCU will send the message with corresponding data for user.
- Step 3: Launch MCU Command
In this step, you can launch MCU command. All command functions can find in the file, “**Commands List**”.
- Step 4: Launch EXIT
Exit the program.

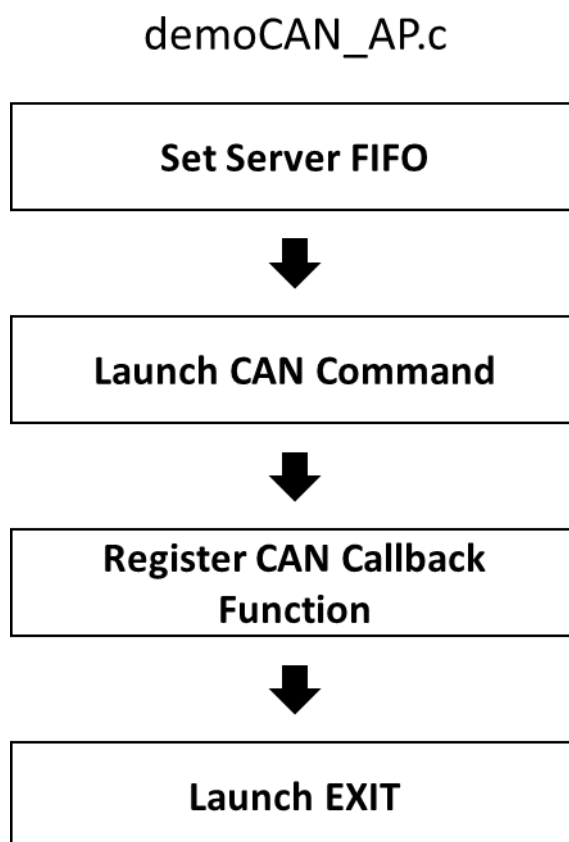


Picture 9 – Client MCU Flow Chart

3.3.2.2 demoCAN_AP.c

In the demoCAN_AP.c, there are 4 step as below (see picture 10).

- Step 1: Set Server FIFO
In this step, you should set server FIFO to read data.
- Step 2: Launch CAN Command
In this step, you can launch CAN command, and **set corresponding baud rate**. All command functions can find in the file, “**Commands List**”.
- Step 3: Register CAN Callback Function
In this step, you can register callback function to CAN.
CAN will send the message with corresponding data for user.
- Step 4: Launch EXIT
Exit the program.



Picture 10 - Client CAN Flow Chart

3.3.3 API Function

There are API function table corresponding steps with demoMCU_AP and demoCAN_AP. More information can find from **demoMCU_AP.c** and **demoCAN_AP.c** in **MUT_SDK-linux_vX.X.X/client/src**.

demoMCU_AP

Step		API Function
1	Set Server FIFO	int Set_Server_FIFO(char *FIFO_NM);
2	Register Event Callback Function	int Reg_Event_CF(void (*callback)(int, BYTE, BYTE *));
3	Launch MCU Command	int LAUNCH_MCU_CMD(char *cmdName, BYTE *dataIn, int dataInLen, BYTE *dataOut, int *dataOutLen);
4	Launch EXIT	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);

demoCAN_AP

Step		API Function
1	Set Server FIFO	int Set_Server_FIFO(char *FIFO_NM);
2	Launch CAN Command	int LAUNCH_CAN_CMD(char *cmdName, BYTE *dataIn, int dataInLen, BYTE *dataOut, int *dataOutLen);
3	Register CAN Callback Function	int Reg_CAN_CF(void (*callback)(int, short, short, short, BYTE[]));
4	Launch EXIT	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);

3.3.4 Demo Screenshot

You should start service program “mcu_service” (see picture 11) before you run demo client program (see picture 12) or your own program used the client API. Besides, you can modify and compile (used Make file in MUT_SDK-linux_vX.X.X/client) demo program to test and try. The screen shot as below just demo the MCU client program to access MCU by connecting MUT service. Maybe you can run “demoCAN_AP” to test by yourself if your machine had our CAN module.

Script:

```
$cd MUT_SDK-linux_v2.1.0/service/
$sudo -s
$bash launch.sh start
Or
$cd MUT_SDK-linux_v2.1.0/service/
$sudo -s
$export LD_LIBRARY_PATH=LIB_x86_64/
$./bin/mcu_service&
```

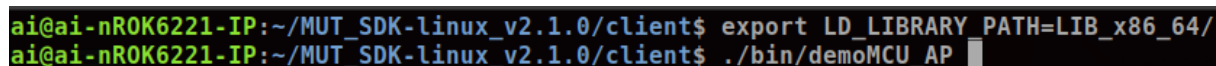


```
ai@ai-nR0K6221-IP:~$ cd MUT_SDK-linux_v2.1.0/service/
ai@ai-nR0K6221-IP:~/MUT_SDK-linux_v2.1.0/service$ bash launch.sh
[USAGE] {sh|bash} launch.sh {action} [fifo name]
{action}
    start, Start MCU/PoE board service
    stop, Stop MCU/PoE board service
[fifo name], If you start/stop PoE board service,
              you must provide customized fifo name to program.
ai@ai-nR0K6221-IP:~/MUT_SDK-linux_v2.1.0/service$ bash launch.sh start
[sudo] password for ai:
Starting MCU/CAN service...
ai@ai-nR0K6221-IP:~/MUT_SDK-linux_v2.1.0/service$
```

Picture 11

Script:

```
$cd MUT_SDK-linux_v2.1.0/service/
$sudo -s
$export LD_LIBRARY_PATH=LIB_x86_64/
$./bin/demoMCU_AP
```



```
ai@ai-nR0K6221-IP:~/MUT_SDK-linux_v2.1.0/client$ export LD_LIBRARY_PATH=LIB_x86_64/
ai@ai-nR0K6221-IP:~/MUT_SDK-linux_v2.1.0/client$ ./bin/demoMCU_AP
```

Picture 12

3.4 Common API Table

There are some of MCU or CAN API to use when you develop a client program. The API as below:

Source	API Function	Description
mcu_ap.c	int LAUNCH_INIT_DEV(char *machineNm, int devType, char *dev, int baudR, int canType);	To Initiate the MCU or CAN devices.
	void Register_MCU_EventCallBack(BYTE byByte1, BYTE byByte2, void (*callback)(CALLBACK_DATA));	To register callback function from library.
	int INIT_CMD_POOL();	To initiate the command pool within the structure of MCU content.
	int LAUNCH_MCU_CMD(char *cmdName, BYTE *dataIn, int dataInLen, BYTE *dataOut, int *dataOutLen);	Launching a command to MCU for service.
	int switchCAN(int canIdx);	To switch the CAN device you want to control if have multiple CAN devices.
mcu_menu_ap.c	FILE * LAUNCH_INIT_CONFIG(int argc ,char *argv[]);	To initiate the MCU or CAN devices with arguments and config file.
	int INIT_UI_MENU(FILE *fConfig, int (*findIdx)(char *, int *), handleFun (*findFunPtr)(char *));	To initiate UI Menu.
	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);	To exit the program.
demoMCU_AP.c	int Set_Server_FIFO(char *FIFO_NM);	For setting server's FIFO name.
	int Reg_Event_CF(void (*callback)(int, BYTE, BYTE *));	Register the callback function on MUT service to receive event.
	int LAUNCH_MCU_CMD(char *cmdName, BYTE *dataIn, int dataInLen, BYTE *dataOut, int *dataOutLen);	Launching a command to MCU for service.
	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);	To exit the program.
demoCAN_AP.c	int Set_Server_FIFO(char *FIFO_NM);	For setting server's FIFO name.
	int LAUNCH_CAN_CMD(char *cmdName, BYTE *dataIn, int dataInLen, BYTE *dataOut, int *dataOutLen);	Launching a command to CAN for service.
	int Reg_CAN_CF(void (*callback)(int, short, short, short, BYTE[]));	Register the callback function on MUT service to receive event.
	int LAUNCH_EXIT(char *cmdName, FILE *pfile, int uiBase);	To exit the program.

	int uiBase);	
--	--------------	--

Reference:

Source	File
MUT_SDK-linux_vX.X.X/service/src	mcu_ap.c mcu_menu_ap.c
MUT_SDK-linux_vX.X.X/client/include	mut.h
MUT_SDK-linux_vX.X.X/client/src	demoMCU_AP.c demoCAN_AP.c

4. Flash_MCU

In this chapter, you will know how to use **Flash_MCU** step by step. Here are some tips to help developers get started quickly. In chapter 4.1, you should do some operations before using **Flash_MCU**. In chapter 4.2, you can update your MCU/CAN firmware by **FLASH_MCU**.

4.1 Software Operation

4.1.1 Decompress the Flash_MCU

`$tar -Jxf FLASH_MCU-linux_vX.X.X.tar.xz (X.X.X is flash tool version)`

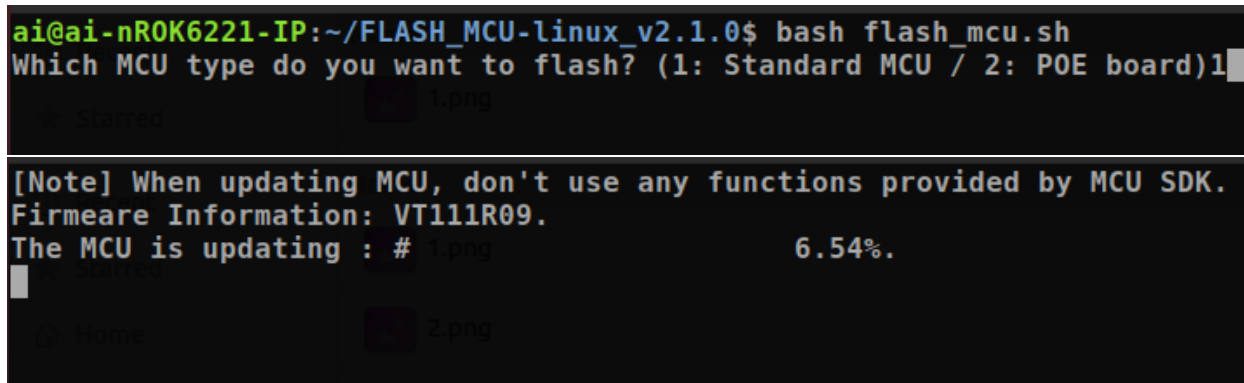
4.1.2 Source Code Structure

After decompress **Flash_MCU**, the contents of **Flash_MCU** are as below:

File Path	Description
bin	The executable program “mcu_upd” included.
configs	The configuration files for a lot of products included.
FW	Firmware files.
LIB_x86_64	There are some of necessary shared libraries included.
flash_mcu.sh	You can use it to flash MCU/CAN firmware. The usage is as below: bash flash_mcu.sh

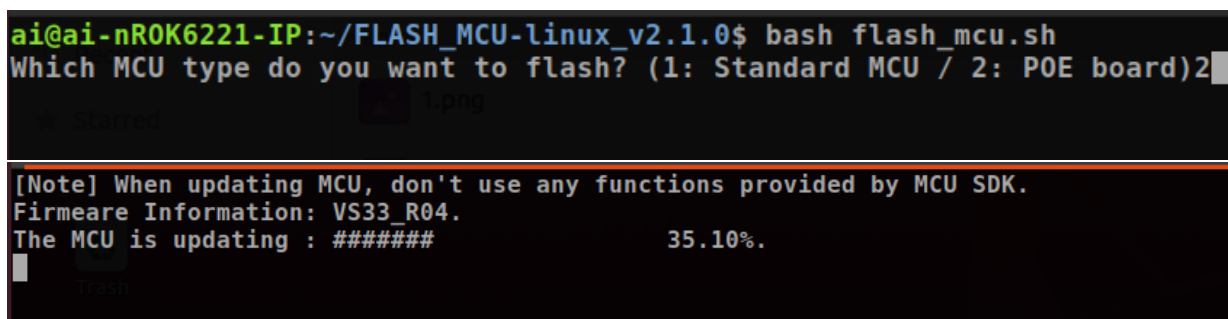
4.2 Flash MCU/CAN

You can run the flash script (in `FLASH_MCU-linux_vX.X.X`) if you need to flash MCU. Follow the steps as below (see picture 13 ~ 14) to flash.



```
ai@ai-nR0K6221-IP:~/FLASH_MCU-linux_v2.1.0$ bash flash_mcu.sh
Which MCU type do you want to flash? (1: Standard MCU / 2: POE board)1
[Note] When updating MCU, don't use any functions provided by MCU SDK.
Firmeare Information: VT111R09.
The MCU is updating : # 6.54%.
```

Picture 13



```
ai@ai-nR0K6221-IP:~/FLASH_MCU-linux_v2.1.0$ bash flash_mcu.sh
Which MCU type do you want to flash? (1: Standard MCU / 2: POE board)2
[Note] When updating MCU, don't use any functions provided by MCU SDK.
Firmeare Information: VS33_R04.
The MCU is updating : ##### 35.10%.
```

Picture 14

- **It will auto detect your machine and update your firmware of MCU.**

Appendix A – Histories

Version	Description
V2.2.38	Update bin file for ATC750 Modify ATC750 config – Added get system parameter.
V2.2.37	Supported VTC6232
V2.2.36	Remove ATC3750 series get Wi-Fi status connector hint.
V2.2.35	Supported nROK6231-ST
V2.2.34	Fix the error preventing demoMCU_AP.py from running.
V2.2.33	Add nROK6231 config
V2.2.32	Debug demoMCU_AP.py
V2.2.31	Update bin file for ATC3530 Modify ATC3530 config – remove set power type
V2.2.29	Supported VMC1110
V2.2.28	Update bin file for ATC3750-6C Fixed "GET_WWAN_STATUS" print incorrect interface
V2.2.27	Supported VTC1920
V2.2.26	Supported VTKSCAP Update bin file for ATC3750-6C Set ATC series default debug level = 0 Added new functions(Input Frequency & Pulse Count) for ATC3750
V2.2.25	Modify configuration file(enable battery MUT) for nROK7251 Debug MCU update program
V2.2.24	Supported VTC6221-NMD for ODM Update bin file for VTC6221-NMD
V2.2.23	Supported X80
V2.2.22	Add bin file for nROK6221-IP & VTC6221 Modify the nROK6221-IP config file
V2.2.21	Update bin file for VTC7270

V2.2.20	Update bin file for ATC3750-8M
V2.2.19	Modify function (SIM Card Set & Get)for VTC6231
V2.2.18	Supported VTC6231
V2.2.16	Supported ATC3750-8M
V2.2.15	Modify configuration file(enable battery MUT) for VMC320 c& Debug MCU update program
V2.2.14	Added function (KEY LED STATUS)for VMC320c
V2.2.12	Supported nROK7270/nROK7271
V2.2.11	Supported ATC3540/ATC3520
V2.2.10	Add Event key(button break)
V2.2.9	1. Supported VMC320(arm)
V2.2.8	Supported VTC1031/nROK1031
V2.2.7	1. Fix send command failed due to parse MCU response error with long data.
V2.2.6	Supported ATC3750
V2.2.5	Supported aROK8110
V2.2.4	Supported VTK62B protocol v1.5
V2.2.3	Supported VTC7270
v2.2.1	Supported nROK6222
v2.2.0	1. Optimized stability of UART and Client/Server.
v2.2.0 v2.1.9	2.Added timeout function to Client/Server.
	1.Supported VTC-7260

v2.1.8	1.Supported vROK3030
v2.1.7	1.Supported VTC1030
v2.1.6	1.Supported VTC7252.
v2.1.5	1.Optimized MUT codes
v2.1.4	1.Optimized the procedure of MCU-flashing
v2.1.3	1.Supported ATC3530.
v2.1.3 v2.1.2	2.Read the update address map from the MCU.
	1.Modified client library to let the python function invoked.
v2.1.1	1.Modified the MUT service & client library & demoMCU_AP.py
v2.1.1 v2.1.0	2.Added the demo python program named demoCAN_AP.py
	1.Supported VIOB-POE2-01(PoE board).
v2.1.0 v2.0.14	2.Added PoE board service/client
	3.Optimized the flow of flashing
	1.Supported nROK6221-IP.
v2.0.13	1.Added the ODM function for VTC-1011
v2.0.13 v2.0.12	2.Modified the update issue of flashing MCU
	1.Support TX64-R131.
v2.0.11	1.Refined the functions of ATC3200
v2.0.11 v2.0.10	2.Add the ATC3200 new MCU bin file to MUT
	1.MUT SDK support ARM platform.

v2.0.9	1.Add the functions for ATC3200
v2.0.9 v2.0.8	2.Modified the flash address map of ATC3200
	1.Support aROK5510.
v2.0.7	1.Fix issue that ST CAN can't be used.
v2.0.6	1.Open the CAN message for mcu_menu_ap.
v2.0.5	1.Refined the functions of nROK-7251
v2.0.5 v2.0.4	2.Refined the functions of ATC3200
	3.Added nROK7252 、VMC-2020
	1.Modified the issue of CAN
v2.0.4 v2.0.3	2.Added MCU demo ap with menu into MUT SDK
	3.Added MCU demo ap without menu
	4.Removed MCU Menu AP tool
	5.Added new functions for VTC-7251
	1.Modified the issue of CAN
v2.0.3 v2.0.2	2.Optimized the log mechanism
	3.Based on MCU Base Library v3.2.4
	1.Modified log level within codes
v2.0.2 v2.0.1	2.Modified the issue of disconnect FIFO
	3.Removed DEBUG statement
	4.Based on MCU Base Library v3.2.3

	1.Supported the python call client shared library of MUT
v2.0.1 v2.0.0	2.Added the flash update info of ATC3200 in function_UPD.c
	3.Added ATC3200 、 nROK7251
	4.Based on MCU Base Library v3.2.2
	1.Added the service/client framework for MUT SDK
v2.0.0 v1.4.6	2.Separated and tag the functions of CAN and MCU
	3.Separated MENU framework from MCU library and MUT SDK
	4.Fine tune MCU library and MUT SDK
	5.Modified the callback function of event
	6.Based on MCU Base Library v3.2.1
	1.Supported VTC6221
v1.4.5	1.Supported VTC-6222
v1.4.4	1.Fixed timeout issue when sending command continuous
v1.4.4 v1.4.3	2.Adjusted update config information for VTC-7251
	1.Supported ATC8110
v1.4.2	1.Supported VTK62B
v1.4.1	For VTC-7251
v1.4.1 v1.4.0	1.Fixed timeout of getting SIM card status
	2.Disabled PCIe/SATA control
	1.Supported VTC-7251

v1.3.9	1.Changed odometer from input frequency to pulse count for VTC1020
v1.3.8	1.Supported VMD3002
v1.3.7	1.Supported VTC-7250
v1.3.6	1.Resolved using API commands without UI that will cause system getting slowing down
v1.3.5	1.Support VMC3020 、 VMC4020 、 MVS5200 、 MVS5600 、 MVS5603 、 MVS2623
v1.3.4	1.Supported VTC1910
v1.3.4 v1.3.3	2.Fixed getting POE status error issue
	1.Supported VTC-1011
v1.3.2	1.Enabled getting CAN data of the VTC1020
v1.3.2 v1.3.1	2.Removed WIFI power setting item
	1.Supported VTC1020
v1.3.1 v1.3.0	2.Fixed ATC-8010 bug issue
	3.Based on MCU Base Library v3.1.9
	1.First released
v1.3.0	2.Based on MCU Base Library v3.1.8
	3.Supported VTC1021 、 VTC1911 、 VTC6220 、 VMC3021 、 ATC-8010 、 ALITS-2121ES