MediaTek

MAUI Make/Build Environment and Procedures Design Document

Documents Number:

Preliminary(Released) Information

Revision: 1.4

Release Date: Sep, 30, 2004

Revision History

Revision	Date	Author	Comments
1.00	07/23/2002	Rex Luo	Create initial version 1.0
1.10	09/29/2002	Rex Luo	Release for Pixtel MMi
1.11	10/18/2002	Rex Luo	Add custom release procedure
1.12	12/18/2002	Rex Luo	Modify according to modified procedure
1.20	11/26/2003	Sherman Wang	Release for customers
1.21	12/3/2003	Sherman Wang	Update environment requirements
1.30	12/29/2003	Sherman Wang	Add 6. Description of Options
1.31	03/30/2004	Sherman Wang	Customer release
1.40	09/30/2004	Sherman Wang	Update for make utility changed from PVCS Configuration Builder to GNU make

Table of contents

1 Introduction

MAUI make/build environment and procedures utilize GNU make for building project executable binaries. The actions include **new**, **update**, **remake**, **clean all**, **clean modules, codegen**. Detailed terminology and features can be referenced in later sections.

2 Terminology and Features

- **Terminology**
  - **Customer –** Released customer
  - **Project –** Combination of independent sources, header files, and created image binary, objects, libraries etcsuch as, gprs, gsm.
  - **Action –** Behaviors can be executed by the build script.
  - **Component/Module** – A project decomposition unit which can be created as a library.
  - **New, Update, Remake –** Able to build all, rebuild only changed files based on generated dependency.
  - **Clean** – Able to clean generated objects, libraries, and logs.

- **Features**
  - Able to easily add or delete files from the build
  - Able to handle include files included in other directories
  - Able to build based on relative paths
  - Able to build different projects which have private source trees.
  - Other than public build options, able to allow modules to specify private options.
  - Able to clean dedicated modules or all.
  - Able to debug easily.

3 Environment Requirements and Limitations

3.1 Environment Requirements

- OS
  Windows 2000, WinXP. The recommended OS is Windows 2000 with SP2 or later.

- Compiler
  ADS (Arm Developer Suite) v1.2. The build version should be equal to or greater than 842. The recommended build version is build 842.

- Perl interpretor
  ActivePerl. The recommended version is ActivePerl 5.6.1.
  It can be downloaded from http://www.activestate.com/Products/Download/Get.plex?id=ActivePerl.

3.2 Environment Limitations

- Suggest to expand OS's environment space to 1024.
- Limit clean/update/remake modules to at most 7 modules.
- Make sure "sh.exe" is not in the directories defined in DOS environment variable PATH

4 File Architecture and Directories

4.1 Directory Architecture

Project Name : MAUI

Root Directory -

[D:\pvcs\maui\]
[mcu]
Make.bat
M_*.bat
[build]
[make]
[nucleus]
[init]
[inc]
[l1]
[drv]
[ps]
[custom]
[Fast_DL]
[tools]
[mtk_libs]
[tst]
[verno]
…

[mcu]
Description:
MCU part source codes are placed here which manages protocol stack L1/L2/L3 and application layer issues, and major system boot, power-control management etc.

Make.bat
Description:
MAUI project make/build execution batch script.

[build]
Description:
Generated object, libraries, executable binary and log files directory. The directory will be created automatically. For details, see “Generated Objects, Libraries, Executable Binary and Log Files” sections.

[make]
Description:
Main build scripts and option file directory.

[nucleus]
Description:

Nucleus Plus RTOS source codes directory include C, Assembly, and included header files.

[init]
Description:
System boot and hardware dependent initialization directory. Meanwhile, exception handling, and interrupt service routine dispatcher are also placed here.

[inc]
Description:
System boot, initialization, layer1, and driver modules common included header files directory.

[l1]
Description:
Layer 1 source codes directory.

[drv]
Description:
Driver modules source codes directory.

[custom]
Description:
Custom's task/modules' sources and header files

[Fast_DL]
Description:
Cmm files for fast download

[tools]
Description:
Miscellaneous tools used in build/make and customer release procedures

[mtk_libs]
Description:
Component libraries provided my MediaTek.

[tst]
Description:
Database for trace

[verno]
Description:
Source code for keeping version information

Make/Build Script Directory –
[D:\pvcs\maui\mcu\make\]

[init]
[drv]

[nucleus_int]
[nucleus]
….
Comp.mak
Gsm2.mak
Monza_GPRS. mak
Option. mak
Verno_Monza.bld
Custom.bld

[init]
[drv]
[nucleus_int]
[nucleus]
…
Description:
Module's include path, source files (C, Assembly, Header Files) list, include path for dependency checking, optional definition files.
For example:
[D:\pvcs\maui\mcu\make\init\]
init.lis
init.def
init.pth
init.inc
init.lis contains init module source codes list; init.def contains init module's private compile predefinitions; init.pth contains init source codes' directory path, and init.inc contains init module's include header files directory path. All entities list in these files should be listed one entity a line. For example, if you add 3 compile predefinitions in one line of init.def, only the first compile predefinition will be considered. 3 lines should be occupied for these 3 compile predefinitions.

Wrong Usage:

```
__ABC_ENABLE__   __XYZ_ENABLE__   __MTK_SUPPORT__
```

Correct Usage:

```
__ABC_ENABLE__
__XYZ_ENABLE__
__MTK_SUPPORT__
```

Comp.mak
Description:
Component modules build script.

Gsm2. mak
Description:
Main make/build script.

Monza_GPRS. mak
Description:
List different configuration according to customer and project requirement

Option. mak
Description:
Project common option, and macro definition build script

Verno_Monza.bld
Version build script.

Custom.bld
Keep some variables used in custom release. It should not be modified.

4.2 Build Scripts

Project make/build procedure flow and relationship can be seen in Fig 1:

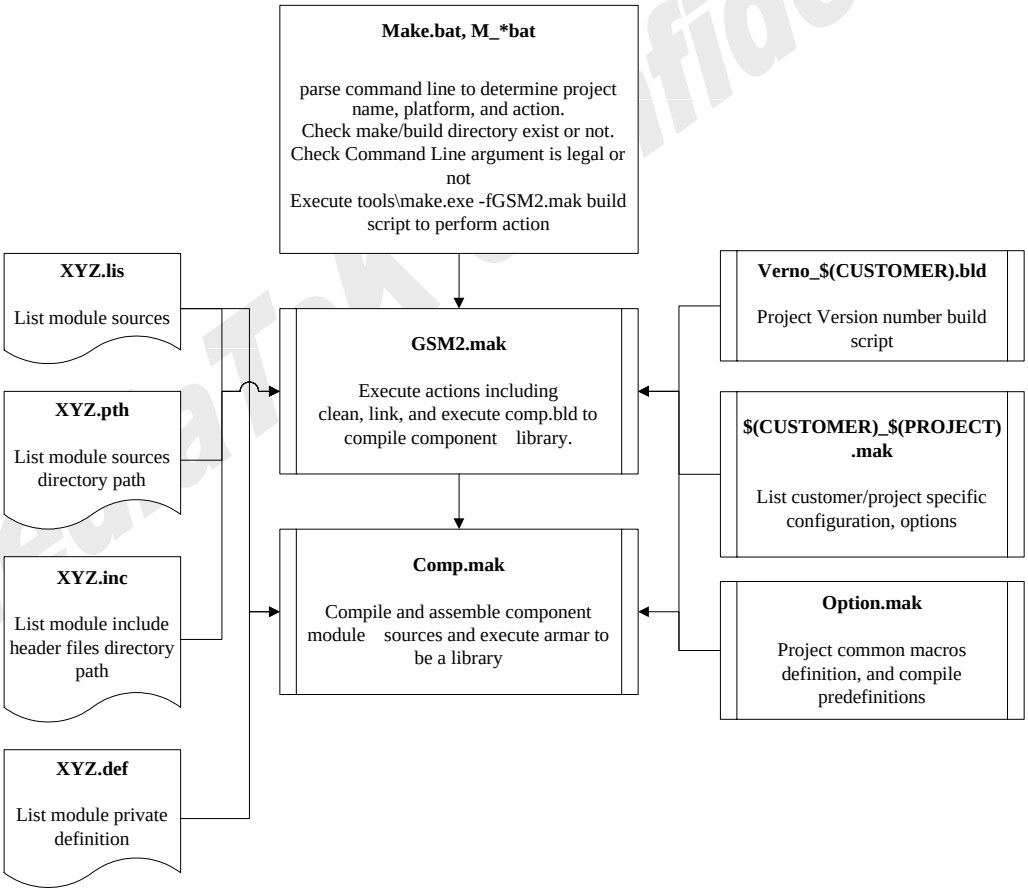

Fig 1. Architecture of MAUI Make/Build Script

Make.bat, M_*.bat
Description:

It will parse command line to determine project name, platform, and action. Meanwhile, checking \make directory exist or not and checking command line argument is legal or not. After checking, to execute build -script GSM2.mak build script to perform action.

Usage: Make [custom=customername] <project> <platform> <action> [module]"

custom	=	Monza	
project	=	GPRS	(GSM only)
action	=	new	(clean, scan, compile, link) (default)
	=	update	(scan, compile, link)
	=	remake	(compile, link)
	=	clean	(clean)
module	=	component module name (nucleus, l1, ...)	

Example:

To make/build new GPRS project, clean all old objects, libraries, and log files etc., the **new** action also creates necessary directories and removes all temporary files, and flushes log files automatically.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS **new**

To update project dependency, and compile changed modules, link. Notice that, update and remake action won't remove temporary files, and flush log file. Build results will be append after last log file.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS **update**

To recompile changed files, and link

d:>\pvcs\maui\mcu\Make custom=Monza GPRS **remake**

To clean all objects, temporary files, libraries, and executable binaries. Meanwhile log file will also be flushed.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS **clean**

To clean dedicated init modules' objects libraries. Meanwhile log file will also be flushed.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS **clean init**

Gsm2.mak

Description:

Main make/build script which executes actions including clean, retrieve, scan, link, and executes comp.mak to compile component's library.

USAGE:
build <CUSTOMER> <PROJECT> [build_flag] [build script] <ACTION>
CUSTOMER = Monza …

PROJECT = GPRS

ACTION = new update remake clean

Example:
build CUSTOMER=Monza PROJECT=GPRS -nologo -script make\gsm2.mak new

Comp.mak
Description:

Component modules build script compiles and assembles component module's sources and executes armar.exe to generate libraries

$(CUSTOMER)_$(PROJECT).mak (Monza_GPRS.mak)
Description:
Customer-project private configuration, including pre-processor definition, include path, modules etc..

Option.mak
Description:
Project common option, and macro definition build script

Verno_Monza.mak
Version number build script.

4.3 Module Option Files

Module option files includes module's include path, source files (C, Assembly) list, include path for dependency checking, optional definition files.

For example:
[D:\pvcs\maui\mcu\make\init\]
init.lis
init.def
init.pth
init..inc
init.lis contains init module source codes list; init.def contains init module's private compile predefinitions; init.pth contains init source codes' directory path, and init.inc contains init module's include header files directory path.

init.lis – List all init module's source codes including C sources, and Assembly sources.

Example content:

init\bootarm.s

init\ex_item.c
init\idma.c
init\init.c
init\intrCtrl.c
init\isrentry.c
init\pdn.c
init\regioninit_ads.s
…

init.def – List all init module's private compile predefinition for C sources.

NOTICE:
- **Current implementation doesn't consider assembly pre-definitions**
- **APCS_INTWORK is a special keyword to identify the module is compiled or assembled with APCS Interwork Specification (-apcs /interwork).**

Example content:

APCS_INTWORK
MTK_SUPPORT

init.pth – List all init module's source codes directory from mcu root directory.

Example content:

init
init\src

init.inc – List all init module's included header files directory path from mcu root directory.

Example content:

init
inc
inc\hwdrv
l1\common

NOTICE:
- **Some modules, for example, nucleus which contain interwork and non-interwork sources must be split to different modules to build. Nucleus is split to nucleus for non-interwork and nucleus_int for interwork. The advantage to do that is performance consideration, while disadvantage is split 2 libraries.**

4.4 Intermediate Build Scripts and Log File

There are several PVCS Configuration Build internal temporary files. Furthermore, main build script will generate several intermediate build scripts to transfer needed information. They are all ~*.tmp in make directory.

[D:\pvcs\maui\mcu\make\]

Comp.mak
Gsm2.mak
~buildinfo.tmp
Option.mak
~compbld.tmp
Verno_Monza.mak

~buildinfo.tmp
Description:
The file contains project name, and platform definition for Gsm2.mak and Option.mak to reference.

```
PROJECT=GPRS
PLATFORM=MT6218B
```

~compbld.tmp
Description:
Component modules needed build information.

```
FIXPATH    = D:\pvcs\maui\mcu
OBJSDIR    = D:\pvcs\maui\mcu\build\Monza\GPRS\MT6218Bo
RULESDIR   = D:\pvcs\maui\mcu\build\Monza\GPRS\MT6218Br
SRCPATH    =  D:\pvcs\maui\mcu\init\src
COMPONENT  = adaptation
LISFILE    = D:\pvcs\maui\mcu\make\init\init.lis
TARGDIR    = D:\pvcs\maui\mcu\build\Monza
INCDIRS    = D:\pvcs\maui\mcu\nucleus\inc                D:\pvcs\maui\mcu\kal\include
D:\pvcs\maui\mcu\kal\common\include D:\pvcs\maui\mcu\ps\adaptation\include
D:\pvcs\maui\mcu\ps\stacklib\include D:\pvcs\maui\mcu\ps\interfaces\include D:\pvcs\maui\mcu\ps\init\include
D:\pvcs\maui\mcu\ps\config\include D:\pvcs\maui\mcu\ps\ith\sme\include
D:\pvcs\maui\mcu\ps\ith\sme_stack\include D:\pvcs\maui\mcu\ps\ith\sme_tt\include
D:\pvcs\maui\mcu\ps\gen\sme_tt D:\pvcs\maui\mcu\ps\gen\sme D:\pvcs\maui\mcu\ps\gen\sme_stack
D:\pvcs\maui\mcu\ps\mm\include
PROJDIR    = D:\pvcs\maui\mcu\build\Monza\GPRS
PLATFORM   = MT6218B
DEFINES    = IDLE_TASK _DEBUG IDMA_DOWNLOAD   MTK_KAL RELEASE_KAL KAL_ON_NUCLEUS
MT6218B
INTWORK    = FALSE
```

4.5 Generated Objects, Libraries, Executable Binary and Log Files

Build script will generate log, object build directory according to customer, project, and platform.

For example:
[D:\pvcs\maui\mcu\build\]
[Monza]
MT6218B.log

[log]
[GPRS]
[MT6218Br]
[MT6218Bo]
[nucleus]
[nucleus_int]
[init]
[drv]
[lib]

MT6218B.log
Description:
Gsm2.mak will generate the log. The log will record the overall building process.

[log]
Description:
Comp.mak will generate the log when building each component. The log will record the compiling, archiving process of each component.

Build script will generate and flush above log files if script execute **clean** action, or actions which depend on **clean** action.

[MT6218Br]
Description:
Project module dependency rules directory. These .dep are generated by PVCS scandep.exe.

For example:
[D:\pvcs\maui\mcu\build\Monza\GPRS\MT6218Br\]
nucleus.dep
init.dep
nucleus_int.dep
drv.dep

[MT6218Bo]
Description:
Project generated objects, libraries directory.

[nucleus]
[nucleus_int]
[init]
[drv]
…
Description:
Component modules generated object directory.

[lib]
Description:
Component modules generated library directory.

For example:
[D:\pvcs\maui\mcu\build\Monza\GPRS\MT6218Bo\lib\]
nucleus.lib
nucleus_int.lib
init.lib
drv.lib
…

5 Procedures and Functionality

Build/make procedures and functionality are listed as below:

1. Create project root directory for project sources, master build batch script and make scripts.

2. To make/build new GPRS project, clean all old objects, libraries, and log files etc., the **new** action also creates necessary directories and removes all temporary files, and flushes log files automatically.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS new

new action depends on **cleanall asngen codegen asnregen update**, therefore, will clean all intermediate scripts, log file, and call scandep.exe to scan header file dependency, and build component module's libraries, link, and transform to executable image file.

3. To clean all objects, temporary files, libraries, and executable binaries. Meanwhile log file will also be flushed.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS clean

clean action will check object directories, and create them automatically if they are not existed. If you didn't assign any modules to clean, it will clean all modules by default.

4. To clean dedicated init modules' objects libraries. Meanwhile log file will also be flushed.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS clean init
d:>\pvcs\maui\mcu\Make custom=Monza GPRS clean init drv

5. To update project dependency, and compile changed modules, link. Notice that, **update** and **remake** action won't remove temporary files, and flush log file. Build results will be append after last log file.

d:>\pvcs\maui\mcu\Make custom=Monza GPRS update

update action depends on **cleanlog codegen genverno gencustominfo resgen scan remake**, therefore, won't clean module's objects, and library. However, update action will check dependency and remake modified sources, link to binary. update and remake actions don't depend on **cleanall** action, therefore, will not create object directory, and flush log file. You must execute new or clean action to flush log file.

6. To recompile changed files, and link

d:>\pvcs\maui\mcu\Make custom=Monza GPRS remake

remake action depends on **cleanlog libs $(BIN_FILE)**. **libs** will compose the libraries of each components and **$(BIN_FILE)** will link them to binary.

6 Description of Options

This section describes the frequently used options for configuring the flavor of a build.

6.1 Core Software

Option	Location	Description
L1_CATCHER	Monza_GPRS.mak	L1 Catcher Support; TRUE | FALSE
SPLIT_SYSTEM	Monza_GPRS.mak	Split system feature; TRUE | FALSE
NU_DEBUG	Monza_GPRS.mak	Nucleus Plus debug support; TRUE | FALSE
NU_NO_ERROR_CHECKING	Monza_GPRS.mak	No Nucleus Plus debug support. ; TRUE | FALSE
IRDA_SUPPORT	Monza_GPRS.mak	Used to enable/disable the TST dump via IRDA; TRUE | FALSE
MTK_SLEEP_ENABLE	Monza_GPRS.mak	Sleep Mode Support; TRUE | FALSE
RF_MODULE	Monza_GPRS.mak	BRIGHT2 | BRIGHT4 | MT6119 | AERO | FOUNTAIN | FOUNTAIN2 | SPRING | KLM2003_FOUNTAIN2 | KLM2003_SPRING | CHICAGO2003_FOUNTAIN2 | CHICAGO2003_AERO | CANNON_FOUNTAIN2
L1_GPRS	Monza_GPRS.mak	L1 GPRS Function, Notice: MT6205 don't support that
MTK_DSP_DEBUG	Monza_GPRS.mak	DSP Debugging Feature; TRUE | FALSE
CSD_SUPPORT	Monza_GPRS.mak	CSD Feature; TRUE | FALSE
PMIC_PRESENT	Monza_GPRS.mak	This option is for PMIC support; TRUE | FALSE
PLATFORM	Monza_GPRS.mak	Hardware Platform, MT6205 | MT6208 | FPGA | MT6218 ...etc.
BOARD_VER	Monza_GPRS.mak	Baseband main board description, SHOULD BE ONE OF THE FOLLOWINGS: MT6208_EVB | MT6208_CEVB | MT6205_CEVB | ORDNANCE | KLM2003_BB | CHICAGO2003_BB | MT6218_MW001 | CANNON
SUB_BOARD_VER	Monza_GPRS.mak	CANNON Baseband main board subversion, SHOULD BE ONE OF THE FOLLOWINGS PCB01
LCD_MODULE	Monza_GPRS.mak	These options are to support different kinds of panels. S6B1713 | MTKLCM | S1D15G00 | COLOR_LCD | MTKLCM_COLOR | SSD1815B | ORDNANCELCM | DUAL_LCD | KLMLCM | UC1687 | INFOLCM
MCU_CLOCK	Monza_GPRS.mak	MCU clock setting | MCU_13M | MCU_26M | MCU_39M | MCU_52M

Option	Location	Description
EXT_CLOCK	Monza_GPRS.mak	External clock source setting, EXT_13M | EXT_26M
PLUTO_25KEYS	Monza_GPRS.mak	to use the extend keypad for PLUTO; TRUE | FALSE
AFC_TC	Monza_GPRS.mak	AFC temperature compensation, FALSE is used for VCTCXO and low angle XO device, TRUE is used for XO device
SW_FLASHDOWNLOAD	Monza_GPRS.mak	Use software flash download agent; TRUE | FALSE
MCD_SUPPORT	Monza_GPRS.mak	MCD support feature; TRUE | FALSE
TST_SUPPORT	Monza_GPRS.mak	TST Catcher Support; TRUE | FALSE
TCPIP_SUPPORT	Monza_GPRS.mak	TCPIP support feature; UDP_TCP, UDP, TCP, or NONE
TELECA_FEATURE	Monza_GPRS.mak	WAP support feature; WAP, WAP2, WAP_MMS, WAP2_MMS or NONE
FAST_UART	Monza_GPRS.mak	support 921600bps fast uart; TRUE | FALSE
MSDC_CARD_SUPPORT_TYPE	Monza_GPRS.mak	MSDC_SD_MMC for SD/MMC card support MSDC_MS for MS card support MSDC_MSPRO for MS-PRO card support NONE
FM_RADIO_CHIP	Monza_GPRS.mak	FM radio support; NONE|TEA5767HN
NAND_SUPPORT	Monza_GPRS.mak	NAND flash support; TRUE | FALSE
USB_SUPPORT	Monza_GPRS.mak	USB support; TRUE | FALSE
J2ME_SUPPORT	Monza_GPRS.mak	J2ME support: NONE|MTK_J2ME|J2ME_LIB
AMRWB_CODEC	Monza_GPRS.mak	AMR codec support; TRUE | FALSE
JPG_DECODE	Monza_GPRS.mak	JPEG decode support; TRUE | FALSE
JPG_ENCODE	Monza_GPRS.mak	JPEG encode support; TRUE | FALSE
GIF_DECODE	Monza_GPRS.mak	GIF decode support; TRUE | FALSE
DAF_DECODE	Monza_GPRS.mak	Digital audio format decode support; TRUE | FALSE
MP4_CODEC	Monza_GPRS.mak	Mpeg4 codec support; TRUE | FALSE
ISP_SUPPORT	Monza_GPRS.mak	Image signal processor support; TRUE | FALSE
PHB_SIM_ENTRY	Monza_GPRS.mak	Phonebook Entry Number in SIM: 100|200|300

Option	Location	Description
PHB_PHONE_ENTRY	Monza_GPRS.mak	Phonebook Entry Number in NVRAM: 100|200|300
EMAIL_SUPPORT	Monza_GPRS.mak	Email support; TRUE | FALSE
SW_CHANGE_BLOCKING	Monza_GPRS.mak	TRUE is used to enforce backup on s set of important data items.
MELODY_VER	Monza_GPRS.mak	SW_SYN_8K | YAMAHA_MA3 | ROHM_8788 | SIN_WAV_SYN | DSP_WT_SYN; SW_SYN_8K supported since MT6205B, that means it is not valid on MT6208, MT6205, DSP_WT_SYN supported since MT6218B, that means it is not valid on MT6208, MT6205, MT6205B and MT6218
BAND_SUPPORT	Monza_GPRS.mak	support of designated band: PGSM900 | EGSM900 | RGSM900 | DCS1800 | PCS1900 | GSM850 | GSM450 | GSM480 | DUAL900 | TRIPLE | QUAD | DUAL850
PRODUCTION_RELEASE	Monza_GPRS.mak	Production release feature includes auto-reset when system hang; TRUE | FALSE
__CPHS__	Monza_GPRS.mak	Enable the CPHS function of the MS.
__SAT__	Monza_GPRS.mak	Enable the SIM Application Toolkit function of the MS.
__CSD_T__	Monza_GPRS.mak	Enable the transparent data capability of MS if CSD_SUPPORT is TRUE.
__CSD_NT__	Monza_GPRS.mak	Enable the non-transparent data capability of MS if CSD_SUPPORT is TRUE.
_DEBUG	Option.mak	This option is for the debugging feature.
MTK_KAL	Option.mak	If this option is defined, error management of Nucleus+ is combined with KAL's error management.
KAL_ON_NUCLEUS	Option.mak	Some defines in KAL common include files are dependent on the using OS. If it is defined, it means that Nucleus+ is the using OS.
MTK_TARGET	Option.mak	This option is for code running on the target side.
IDMA_DOWNLOAD	Option.mak	This option is for idma.
DEBUG_KAL	Option.mak	This option is to enable debug functions of KAL. If it is defined, extra code and data structure are included for debugging.
__SYS_INTERN_RAM__	Option.mak	This option is for internal SRAM. If it is defined, a memory pool, internal_ram_pool_g, is defined. And the stack of one task can be built on the internal ram pool.
MTK_NEW_API	Option.mak	1. If this option is defined, msg_send_ext_queue() API has only one parameter- ilm_ptr. Otherwise, it has two parameters. 2. receive_msg_int_q() API is declared and defined only when this option is defined. 3. Its original use is for compatibility. Now it must be defined.

Option	Location	Description
DEBUG_SAVE_CUR_THREAD	Option.mak	1. This option is for the feature of debugging of the current running thread. 2. If it is defined, the current thread debug information will be saved when context switch.
DEBUG_ITC	Option.mak	This option is to enable debug functions of Inter Task Communication. If it is defined, extra code and data structure are included for debugging.
DEBUG_BUF	Option.mak	This option is to enable debug functions of buffer management. If it is defined, extra code and data structure are included for debugging.
DEBUG_BUF2	Option.mak	This option is to enable advanced debug functions of buffer management. If it is defined, extra code and data structure are included for debugging.
STDC_HEADERS	Option.mak	This option is for using of C runtime library, such as “stdlib.h”.
TARGET_BUILD	Option.mak	If this option is defined, code is built for running on the board. Otherwise, code is built for running on the phone.

6.2 MMI

Most MMI configuration is used by compile options or specific hard code in MMI source code. If a customer has licensed MMI source code, the customer can modify it. If we use compile options, you can find it out in MAK or MMI_featuresXXXXX.h. [XXXXX means project name]

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
Phonebook records	Custom can configurable SIM/Phone max. numbers of phonebook entities	PHONEBOOK	Compile Options	#define MAX_PB_SIM_ENTRIES 200 #define MAX_PB_PHONE_ENTRIES 100
Input method	Choose different input method core engine	Input method	Compile options	MMI_T9 to enable T9 MMI_ZI to enable ZI Notes: 1. Only one of them can be turned on. 2. Configure in XXX_GSM.mak file

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
Prefered Input Method	Customer can configure prefered input method function or default input method functionality	Input method	Compile Options	#define __MMI_PREFER_INPUT_METHOD__ to enable prefered input method function. Notes: 1. If not defined __MMI_PREFER_INPUT_METHOD__, Editor will apply the default input method. English Lang==> ABC input mode Tra Chinese Lang==>BoPoMoFo input mode Sim Chinese Lang==>PinYin input mode 2. Configure in MMI_featuresXXX.h file
Input method-ZI	Customer can add related compile options for desired input methods	Input method	Compile options	__MMI_ZI_TR_CHINESE__ to enable Traditional Chinese input methods __MMI_ZI_SM_CHINESE__ to enable Simplified Chinese input methods __MMI_ZI_PRC_ENGLISH__ to enable English input methods __MMI_ZI_MULTITAP_PHONETIC_INPUT__ to enable multitap phonetic input methods __MMI_ZI_SMART_PHONETIC_INPUT__ to enable smart phonetic input methods Notes: 1. Configure in MMI_featuresXXX.h file
Input method-T9	Customer can add related compile options for desired input methods	Input method	Compile options	T9LANG_Chinese to enable all Chinese related input methods T9LANG_English to enable all English related input methods Notes: 1. Configure in MMI_featuresXXX.h file
Supported Languages	Customer can add related compile options for desired languages		Compile options	__MMI_LANGUAGE_TRADITIONAL_CHINESE__ __MMI_LANGUAGE_SIMPLIFIED_CHINESE__ __MMI_LANGUAGE_ENGLISH__ Notes: 1. Configure in MMI_featuresXXX.h file
Dialling Key map(*,+,P,W Key Map for phone number input)	Customer can configure related compile option for different multi tap key map of phone number input mode	Dialing Key map	Compile Options	#define __MMI_MULTITAP_KEY_0__ to map the "+, P, W" chars to multi tap key-0 Notes: 1. Configure in MMI_featuresXXX.h file 2. If not defined __MMI_MULTITAP_KEY_0__, the "*,+,P,W" is mapped to multi tap key-star

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
Voice Memo	Customer can disable this function	Voice Memo	Compile options	__MMI_VOICE_MEMO__
Engineering Mode	Customer can disable this function	Engineering Mode	Compile options	__MMI_ENGINEER_MODE__
Factory Mode	Customer can disable this function	Factory Mode	Compile options	__MMI_FACTORY_MODE__
MAX. DIALED CALL Records	Customer can configure the max. number of records for dialed call log	CallHistory	Link Time	#define TOT_SIZE_OF_DIALED_LIST 10
MAX. MISSED CALL Records	Customer can configure the max. number of records for missed call log	CallHistory	Link Time	#define TOT_SIZE_OF_MISS_LIST 20
MAX. RECD CALL Records	Customer can configure the max. number of records for received call log	CallHistory	Link Time	#define TOT_SIZE_OF_RECV_LIST 20
MAX. Data Accounts	Customer can configure the max. number of data accounts	DataAccount	Link Time	#define MAX_DATA_ACCOUNT_LIMIT 5
NVRAM data items (A lot of configuration can be done)	Data items can be modified/added/removed in a customized way.	NVRAM	Link Time/NVRAM	Please refer the documents about NVRAM data item changes guide.
Main LCM contrast level	Main LCM contrast could be configure as 15 abstract level from driver	LCM	NVRAM	NVRAM_EF_CUST_HW_LEVEL_TBL_DEFAULT
SUB LCD	Manufactor can decide if SUBLCD exists	GUI	Compile Options	__MMI_SUBLCD__
Ring Tone Composer	Manufactor can enable/disable this function	Fun & Games	Compile Options	__MMI_RING_COMPOSER__ && __MMI_IMELODY_SUPPORT__
Themes	Manufactor can enable/disable this function	Fun & Games	Compile Options	__MMI_THEMES_APPLICATION__
Download	Manufactor can enable/disable this function	Fun & Games	Compile Options	((__MMI_EMS__) || defined(__MMI_WAP__)) && __DOWNLOAD__
To Do List	Manufactor can enable/disable this function	Organizer	Compile Options	__MMI_TODOLIST__

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
Calendar	Manufactor can enable/disable this function	Organizer	Compile Options	__MMI_CALENDAR__
Unit Converter	Manufactor can enable/disable this function	Organizer	Compile Options	__MMI_UNIT_CONVERTER__
Currency Converter	Manufactor can enable/disable this function	Organizer	Compile Options	__MMI_CURRENCY_CONVERTER__
World Clock	Manufactor can enable/disable this function	Organizer	Compile Options	__MMI_WORLD_CLOCK__

6.3 Applications

6.3.1 Socket & Data Account

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
MAX. Data Accounts	Customer can configure the max. Numbers of data accounts	Data Account	Link Time	#define MAX_DATA_ACCOUNT_LIMIT 5
connection long idle notification	When bearer (CSD or GPRS) is idle for a pre-configured time , Socket layer will notify upper applications (eg, WAP) so that applications can perform proper actions (eg, disconnect)	Socket	Compile options / Link Time	1. AUTO_DISCONNECT_BEARER 2. soc_auto_disc_sec: 120 sec

6.3.2 WAP

If you license Teleca source code, you can change these compile options.

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
Max. number of WAP profile	Customer can configure max. number of WAP profile	WAP1.2.1	Compile Options	BRA_CFG_N_PROFILES (3)

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
WAP profile default factory setting	Customer can pre-install default WAP profile content	WAP1.2.1	NVRAM	Refer section 2.8.2 - WAP Profile in FS_NVRAM_Description_Chicago.doc
Max. number of bookmark	Customer can configure max. number of bookmarks	WAP1.2.1	Compile Options	BRA_CFG_MAX_NBR_BOOKMARKS (20)
Bookmark default factory setting	Customer can pre-install default bookmarks	WAP1.2.1	NVRAM	Refer section 2.8.3 - WAP Bookmark in FS_NVRAM_Description_Chicago.doc
Default WTLS/X.509 Root CA factory setting	Customer can pre-install Security Root CA	WAP1.2.1	NVRAM	Refer section 2.9 - PRE-STORED ROOT CERTIFICATES in FS_NVRAM_Description_Chicago.doc
Max. cache size	Customer can configure max. size of cache	WAP1.2.1	Compile Options	BRA_CFG_MAX_CACHE_SIZE (20000)
WAP User-Agent name	Customer can configure WAP browser User-Agent name	WAP1.2.1	Compile Options	BRS_CFG_DEFAULT_USER_AGENT_HEADER (e.g. E.80)
WAP User-Agent Profile URL	Customer can configure User-Agent Profile URL for its product	WAP1.2.1	NVRAM	Refer section 2.8.1 - WAP Common setting in FS_NVRAM_Description_Chicago.doc
Max. number of PUSH message	Customer can configure max. number of push messages	WAP1.2.1	Compile Options	PUSH_MAX_NO_OF_MSG (15)
Max. number of WAP profile	Customer can configure max. number of WAP profile	WAP1.2.1	Compile Options	BRA_CFG_N_PROFILES (3)

7 Design and Implementation

This section lists the key sections in the build scripts:

7.1 Make.bat & M_*.bat – Main Build Batch File

Make.bat is implemented by DOS execution batch file, and will call PVCS Configuration Builder build command to execute build action.

Core section:

```
rem ************************************************************************
rem void PerformTheBuild(void)
rem ************************************************************************
:build
     echo DO_BUILD_LABEL=FALSE > make\~labelbuild.tmp
     echo CUSTOMER=%CUSTOMER% > make\~buildinfo.tmp
     echo PROJECT=%PROJECT% >> make\~buildinfo.tmp
     echo APLAT=%APLAT% >> make\~buildinfo.tmp
     .\tools\echo1 -n  BUILD_DATE_TIME= >> make\~buildinfo.tmp
     .\tools\time1 >> make\~buildinfo.tmp
     tools\make.exe -fmake\gsm2.mak -r -R CUSTOMER=%CUSTOMER% PROJECT=%PROJECT% %ACTION%
goto finish
```

Explain:

- :build is DOS execution label which execution flow can jump to.
- echo is DOS build-in command and echo string on standard output or assigned output stream.
- tools\echo1 and tools\time1 is used to print current time into `make\~buildinfo.tmp`is
- tools\make.exe is GNU make executable.

7.2 GSM2.mak – Main Build Script

Core sections of GSM2.mak are listed as below:

Major actions in GSM2.mak:

```
# ****************************************************************************
# New Build
# ****************************************************************************
new : cleanall cmmgen asngen codegen asnregen update


# ****************************************************************************
#  Update Build
# ****************************************************************************
update : cleanlog codegen genverno gencustominfo resgen remake
```

```
# ************************************************************************
#  Remake Build
# ************************************************************************
remake : cleanlog libs $(BIN_FILE) done
```

- Action “new” depends on cleanall, cmmgen, asngen, codegen, asnregen and update
- Action “update” depends on cleanlog, codegen, genverno, gencustominfo, resgen and remake
- Action “remake” depends on cleanlog, libs, and $(BIN_FILE)

library creation:

```
# ************************************************************************
# Library Targets
# ************************************************************************
libs: cleanlib $(COMPLIBLIST)
     @echo Library build finished.

cleanlib:
     ……

%.lib:
     echo Beginning $* component build process... >> $(LOG)
     @echo tools\make.exe -fmake\comp.mak -r -R COMPONENT=$* ... $(strip $(COMPLOGDIR))\$*.log
     ……
          (tools\make.exe -fmake\comp.mak -r -R COMPONENT=$* > $(strip $(COMPLOGDIR))\$*.log 2>&1)
     ……
        @type $(strip $(COMPLOGDIR))\$*.log >> $(LOG)
```

- Action “libs” depends on cleanlib and *.lib in $(COMPLIBLIST)
- Action “cleanlib” will clean previous output files including *.log, *.lib
- “tools\make.exe -fmake\comp.mak -r -R COMPONENT=$*” is invoked to create a library

binary creation:

```
# ************************************************************************
#  Executable Targets
# ************************************************************************
$(BIN_FILE):
     ……
       ($(LINK) -via make\~libs.tmp > $(LOG) 2>&1)
     ……
     # ----------------------------
     # The size of the binary image depends on the available memory on the target
     # platform.
     # ----------------------------
     @echo Creating binary file $(BIN_FILE)
```

```
    $(BIN_CREATE) $(strip $(TARGDIR))\$(IMG_FILE) $(BIN_FORMAT) -output $(TARGDIR)\$(BIN_FILE)
    ……
```

- armlink ($(LINK)) will link libraries, and fromelf (BIN_CREATE) will create binary here.

7.3 Monza_GPRS.mak – Customer-Project Specific Build Script

$(COMPLIST):

```
# COMPLIST(during CUSTOM_RELEASE)      = CUS_REL_SRC_COMP + CUS_REL_PAR_SRC_COMP
ifeq ($(strip $(CUSTOM_RELEASE)),TRUE)
    COMPLIST =     $(strip $(CUS_REL_SRC_COMP))
    COMPLIST +=    $(strip $(CUS_REL_PAR_SRC_COMP))
endif
```

- COMPLIST lists all components to be built in customer release. Adding “COMPLIST += xxx” after “COMPLIST += $(CUS_REL_PAR_SRC_COMP)” will add xxx into building modules.

$(CUSTOM_COMMINC):

```
# ****************************************************************************
# Common include path
# ****************************************************************************
CUSTOM_COMMINC    =  init\include stacklib\include adaptation\include config\include \
                     sst\include inc fdd\include \
                     interface\l1interface interface\hwdrv interface\os interface\l1audio \
                     tst\database_classb\pstrace_db tst\include \
                     l1\common dummyps\include

CUSTOM_COMMINC    += ……
```

- CUSTOM_COMMINC lists all including paths for building all modules. In “make\xxx\xxx.inc”, it lists the including paths only for building module xxx instead. Adding “CUSTOM_COMMINC += xxx/yyy” will add xxx/yyy into including paths for building all modules.

$(CUSTOM_OPTION):

```
# ****************************************************************************
# Common preprocessor definitions
# ****************************************************************************
CUSTOM_OPTION = __GSM_MODE__ __GPRS_MODE__ \
                __MOD_L4C__ __MOD_CSM__ __MOD_RAC__ __MOD_SMU__ __MOD_SMSAL__ \
                __MOD_PHB__ __MOD_UEM__ __MOD_CC__ __MOD_CISS__ __MOD_SMS__ \
                __MOD_MM__ __MOD_NVRAM__ __MOD_SIM__ __MOD_TCM__ \
                __SAT__ __EM_MODE__ __CPHS__ __MULTI_BOOT__ __FS_ON__ __BMT_CHECK_CHARGER__ \
                $(MELODY_VER) __18V_30V_ME__  __PHB_COMPARE_NUMBER_9_DIGIT__
```

```
CUSTOM_OPTION += ……
……
```

- CUSTOM_OPTION lists all compile options for building all modules. In “make\xxx\xxx.def”, it lists the compile options only for building module xxx instead. Adding “CUSTOM_OPTION += XXX” will add XXX into compile options for building all modules.

$(CUS_REL_MTK_COMP):

```
CUS_REL_MTK_COMP  +=  adaptation config interface_classb kal \
     nucleus nucleus_int nucleus_debug stacklib fs \
     cc ciss data flow_ctrl l4_classb llc mm_classb ppp psconfig \
     rr_classb sim sm sms sndcp mmi \
     mtkdebug amr515 ft l1audio l1audio32 sst fdd

CUS_REL_MTK_COMP    += ……
```

- CUS_REL_MTK_COMP lists all components provided my MediaTek with .lib only. These .lib are put in \mcu\mtk_libs. Besides, the components listed in CUS_REL_PAR_SRC_COMP also have .lib in \mcu\mtk_libs.

7.4 Comp.mak – Component Module Build Script

Building objects:

```
# ********************************************************************************
# Component Targets
# ********************************************************************************
# ----------------------------
# C Objects
# ----------------------------
.c.obj:
     @echo Compiling $< ...
     @if exist tmp0.bat del /f /q tmp0.bat
     @tools\strcmpex.exe $(ACTION) remake e tmp0.txt  $(CINTWORK) $(CFLAGS) $(CDEFS) $(CINCDIRS)
     -o $(COMPOBJS_DIR)/$@ $<
     @tools\strcmpex.exe $(ACTION) remake n tmp0.txt  $(CINTWORK) $(CFLAGS) $(CDEFS) $(CINCDIRS)
     -MD -o $(COMPOBJS_DIR)/$@ $<
     @if exist tmp0.txt tools\warp.exe tmp0.txt
     @if exist tmp0.txt $(CMPLR) -via tmp0.txt
     ……

# ----------------------------
# Assembly Objects
# ----------------------------
.s.obj:
     @echo Compiling $< ..
     @$(ASM) $(AINTWORK) $(AFLAGS) $(ADEFS) $< -o $(COMPOBJS_DIR)/$@
```

- .c.obj: part is responsible for compiling C code
- "@tools\strcmpex.exe $(ACTION) remake e tmp0.txt $(CINTWORK) $(CFLAGS) $(CDEFS) $(CINCDIRS) -o $(COMPOBJS_DIR)/$@ $<" is used to output a long line into tmp0.txt to avoid the DOS "command line too long" error. It will echo options for compiling .c into tmp0.txt and then "$(CMPLR) -via tmp0.txt" will execute the compiler.
- .s.obj: part is responsible for compiling assembly code

Building a library:

```
# ****************************************************************************
# Library Targets
# ****************************************************************************
update_lib: obj_dir $(TARGLIB)

obj_dir:
     @if not exist $(OBJSDIR)\$(COMPONENT) \
        (md $(OBJSDIR)\$(COMPONENT))
     @if exist $(RULESDIR)\$(COMPONENT).dep del /q /f $(RULESDIR)\$(COMPONENT).dep

$(TARGLIB) : $(COBJS) $(AOBJS)

     # If library for customer release exists.
     # Copy and update it or create a new one
     @if exist $(FIXPATH)\mtk_lib\$(COMPONENT).lib  \
        (copy /z $(FIXPATH)\mtk_lib\$(COMPONENT).lib $(subst /,\,$(TARGLIB))) & \
        ($(LIB) -r $(TARGLIB) $(COMPOBJS_DIR)/*.obj) \
     else \
        ($(LIB) -create $(TARGLIB) $(COMPOBJS_DIR)/*.obj)
```

- "$(TARGLIB) : $(COBJS) $(AOBJS)" part is responsible for archiving a library from some objects
- For library is existed in \mcu\mtk_libs, "armar –r …" is invoked for replacing newly generated objects. It's used in creating libraries listed in $(CUS_REL_PAR_SRC_COMP)
- For other libraries, "armar –c …" is invoked to create a new library.

8 Error Messages

In general, the error message is logged on \build\Monza\MT6218B.log and \build\Monza\log*.log. While you have compile or linking errors, you can find the \build\Monza\log*.log or \build\Monza\MT6218B.log respectively for detail.

- Environment not installed well or environment space not enough

Message:

Bad command or filename

- Source path wrong:

Message:

c:\progra~1\arm\adsv1_1\bin\armar.exe -create -c -via C:\WINDOWS\TEMP\lis_0028.tmp
Warning: L6875W: Archive D:\pvcs\maui\mcu\build\MTK\gprs\mt6208o\lib\data.lib is not an ELF Object Library

c:\progra~1\arm\adsv1_1\bin\armar.exe -create -r -via C:\WINDOWS\TEMP\lis_0050.tmp
Error: L6833E: File 'D:\pvcs\maui\mcu\build\MTK\gprs\mt6208o\data\data_deinit.obj' does not exist

9 How to customize your build environment

Monza_GPRS.mak is the customer-project specific build script. The customer can customize the configurations in this file.
The following demo some scenarios of the customization:

9.1 To add some modules into or remove some modules from the building procedure.

To complete this kind of configuration, it is necessary to understand the following variables in the make file Monza_GPRS.mak

- COMPLIST: list all source code modules can be built to .lib. In initial custom release, COMPLIST should be the sum of CUS_REL_SRC_COMP and CUS_REL_PAR_SRC_COMP. The following is the initial setting in custom release.

```
ifeq ($(strip $(CUSTOM_RELEASE)),TRUE)
   COMPLIST =    $(strip $(CUS_REL_SRC_COMP))
   COMPLIST +=   $(strip $(CUS_REL_PAR_SRC_COMP))
endif
```

- CUS_REL_MTK_COMP: list all modules provided with .lib only. These .lib are put in \mcu\mtk_libs.

- **If you want to add a source module**

1. add the module "xyz" (in lower case) into COMPLIST.

```
ifeq ($(strip $(CUSTOM_RELEASE)),TRUE)
   COMPLIST =    $(strip $(CUS_REL_SRC_COMP))
   COMPLIST +=   $(strip $(CUS_REL_PAR_SRC_COMP))
   COMPLIST +=   xyz
endif
```

2. add a folder “mcu\make\xyz” for xyz.lis, xyz.inc, xyz.pth, xyz.def.

- **If you want to remove a source module**

1. remove the module, for example "custom", from COMPLIST. Maybe the module is defined in CUS_REL_SRC_COMP or CUS_REL_PAR_SRC_COMP, instead of defined in COMPLIST directly.

```
CUS_REL_SRC_COMP += verno ~~custom~~
……
ifeq ($(strip $(CUSTOM_RELEASE)),TRUE)
   COMPLIST =    $(strip $(CUS_REL_SRC_COMP))
   COMPLIST +=   $(strip $(CUS_REL_PAR_SRC_COMP))
endif
```

- **If you want to move a source module to a .lib module:**

1. remove the module , for example "media", from COMPLIST. Maybe the module is defined in CUS_REL_SRC_COMP or CUS_REL_PAR_SRC_COMP, instead of defined in COMPLIST directly.
2. add the module "media " (in lower case) into CUS_REL_MTK_COMP.

```
CUS_REL_PAR_SRC_COMP +=  l1_classb init ~~media~~
……
```

```
ifeq ($(strip $(CUSTOM_RELEASE)),TRUE)
   COMPLIST =    $(strip $(CUS_REL_SRC_COMP))
   COMPLIST +=   $(strip $(CUS_REL_PAR_SRC_COMP))
endif
……
CUS_REL_MTK_COMP  +=  adaptation config interface_classb  …… \
sst  fdd ppp media
```

3. copy \mcu\build\Monza\GPRS\MT6218Bo\lib\media.lib to \mcu\mtk_libs, even the media.lib is already existed in \mcu\mtk_libs.
4. delete the folder mcu\make\ media

10 Reference

1. GNU Make Manual, Version 3.80