MediaTek

MAUI Make/Build Environment and Procedures Design Document

Document Number:

Preliminary (Released) Information

Revision: 1.43

Release Date: Sep. 04, 2005

Legal Disclaimer

BY OPENING OR USING THIS FILE, BUYER HEREBY UNEQUIVOCALLY ACKNOWLEDGES AND AGREES THAT THE SOFTWARE/FIRMWARE AND ITS DOCUMENTATIONS ("MEDIATEK SOFTWARE") RECEIVED FROM MEDIATEK AND/OR ITS REPRESENTATIVES ARE PROVIDED TO BUYER ON AN "AS-IS" BASIS ONLY. MEDIATEK EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. NEITHER DOES MEDIATEK PROVIDE ANY WARRANTY WHATSOEVER WITH RESPECT TO THE SOFTWARE OF ANY THIRD PARTY WHICH MAY BE USED BY, INCORPORATED IN, OR SUPPLIED WITH THE MEDIATEK SOFTWARE, AND BUYER AGREES TO LOOK ONLY TO SUCH THIRD PARTY FOR ANY WARRANTY CLAIM RELATING THERETO. MEDIATEK SHALL ALSO NOT BE RESPONSIBLE FOR ANY MEDIATEK SOFTWARE RELEASES MADE TO BUYER'S SPECIFICATION OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

BUYER'S SOLE AND EXCLUSIVE REMEDY AND MEDIATEK'S ENTIRE AND CUMULATIVE LIABILITY WITH RESPECT TO THE MEDIATEK SOFTWARE RELEASED HEREUNDER WILL BE, AT MEDIATEK'S OPTION, TO REVISE OR REPLACE THE MEDIATEK SOFTWARE AT ISSUE, OR REFUND ANY SOFTWARE LICENSE FEES OR SERVICE CHARGE PAID BY BUYER TO MEDIATEK FOR SUCH MEDIATEK SOFTWARE AT ISSUE.

THE TRANSACTION CONTEMPLATED HEREUNDER SHALL BE CONSTRUED IN ACCORDANCE WITH THE LAWS OF THE STATE OF CALIFORNIA, USA, EXCLUDING ITS CONFLICT OF LAWS PRINCIPLES.

MTK Confidential Release for GUOHONG
MTK Confidential Release for GUOHONG

Revision History

Revision	Date (mm/dd/yyyy)	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
1.41	01/05/2005	Joe Wang	Update for RVCT2.1.
1.42	08/14/2005	Joe Wang	Update for new make.bat and append info
1.43	09/04/2005	Naomi Ko	English review.

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 are described in the following sections.

1.1 Features

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

1.2 References

[1] GNU Make Manual, Version 3.80

1.3 Terms and Definitions

Term	Definition
Action	Behaviors that can be executed by the build script.
Clean	Ability to clean generated objects, libraries, and logs.
Component/Module	A project decomposition unit which can be created as a library.
Customer	Released customer.
New, Update, Remake	Ability to build all or rebuild only modified files based on generated dependencies.
Project	Combination of independent sources, header files, and created image binaries, objects, libraries, etc. such as GPRS, GSM.

2 Environment Requirements and Limitations

2.1 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 842 or greater. The recommended build version is build 842.
  - RVCT (RealView Compilation Tools) v2.1. The build version should be 498 or greater. The recommended build version is build 498.
    It can be downloaded from http://www.arm.com/support/downloads/

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

2.2 Environment Limitations

- Make sure “sh.exe” is not in the directories defined in DOS environment variable PATH

3 File Architecture and Directories

3.1 Directory Architecture

Project Name : MAUI

3.1.1 Root Directory

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

Make.bat, m*.pl	MAUI project make/build execution batch script.
[build]	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.
[custom]	Custom’s task/modules’ sources and header files
[drv]	Driver modules source codes directory.
[Fast_DL]	Cmm files for fast download
[inc]	System boot, initialization, layer1, and driver modules common included header files directory.
[init]	System boot and hardware dependent initialization directory. Meanwhile, exception handling, and interrupt service routine dispatcher are also placed here.
[l1]	Layer 1 source codes directory.

[make]	Main build scripts and option file directory.
[mcu]	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.
[mtk_lib]	Component libraries provided my MediaTek.
[nucleus]	Nucleus Plus RTOS source codes directory include C, Assembly, and included header files.
[tools]	Miscellaneous tools used in build/make and customer release procedures
[tst]	Database for trace
[verno]	Source code for keeping version information

3.1.2 Make/Build Script Directory

[D:\pvcs\maui\mcu\make\]
- Custom.bld
- Comp.mak
- Gsm2.mak
- Monza_GPRS.mak
- Option.mak
- Verno_Monza.bld
- [drv]
- [init]
- [nucleus]
- [nucleus_int]
- …

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 listed in these files should be listed one entity per line: if multiple compile predefinitions are added in one line of init.def, only the first compile predefinition is considered. Three compile predefinitions must occupy three separate lines.

Wrong Usage:

```
__ABC_ENABLE__   __XYZ_ENABLE__   __MTK_SUPPORT__
```

Correct Usage:

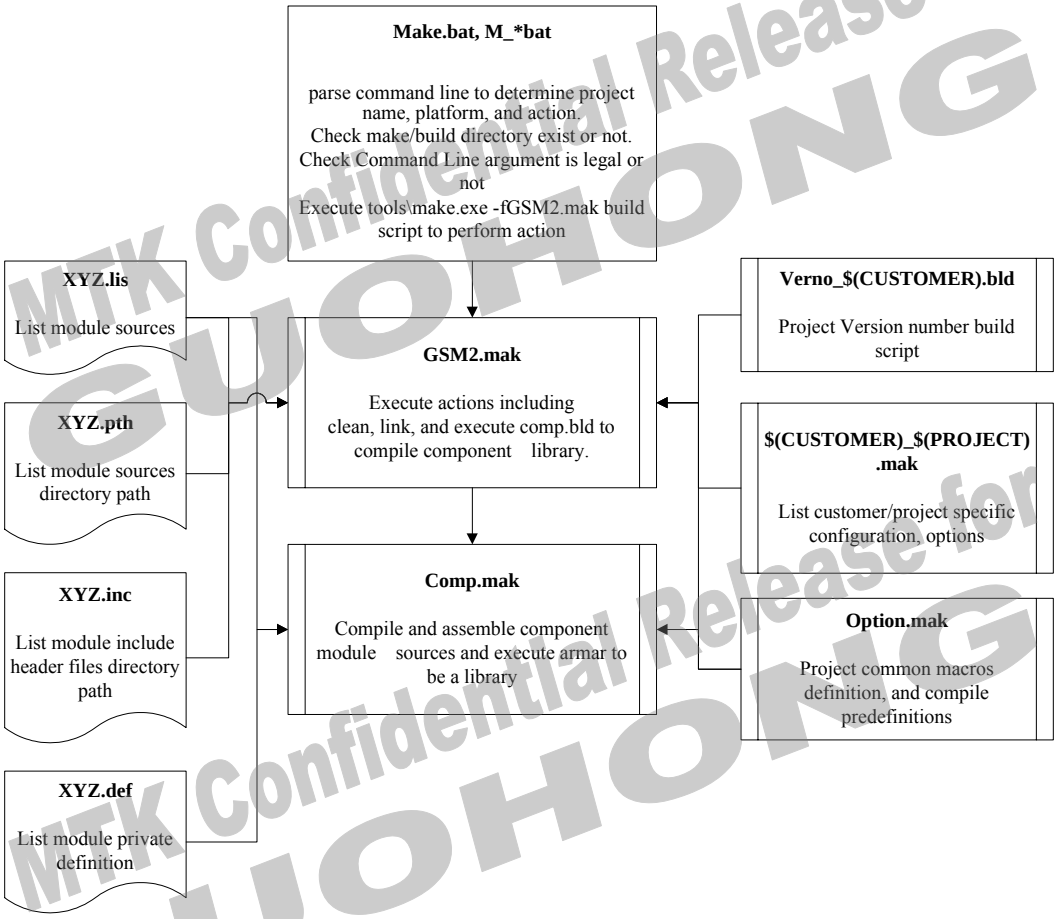
```
__ABC_ENABLE__
__XYZ_ENABLE__
__MTK_SUPPORT__
```

Comp.mak	Component modules build script.
Gsm2. mak	Main make/build script.
Monza_GPRS.mak	List different configuration according to customer and project requirement
Option.mak	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.

3.2 Build Scripts

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

Figure 1: Architecture of MAUI Make/Build Script



Make.bat, M*.pl Parses the command line to determine project name, platform, and action. The existence of a \make directory and the validity of the command line arguments are verified. After verification is complete, the build script Gsm2.mak is executed to perform other actions.

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

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

Example 1
To make/build a new GPRS project, clean all old objects, libraries, and log files etc. The **new** action also creates the necessary directories, removes all temporary files, and flushes log files automatically.

```
d:>\pvcs\maui\mcu\Make  Monza  GPRS  new
```

Example 2
To update project dependency, compile changed modules, and link. Note that **update** and **remake** actions do not remove temporary files, nor do they flush the log file. Build results are appended after last entry in the log file.

```
d:>\pvcs\maui\mcu\Make  Monza  GPRS update
```

Example 3
To recompile changed files only and link:

```
d:>\pvcs\maui\mcu\Make  Monza  GPRS remake
```

Example 4
To clean all objects, temporary files, libraries, and executable binaries. The log file is also flushed.

```
d:>\pvcs\maui\mcu\Make  Monza  GPRS clean
```

Example 5
To clean dedicated init module's object library. The log file is also flushed.

```
d:>\pvcs\maui\mcu\Make  Monza  GPRS clean init
```

Gsm2.mak Main make/build script which executes actions including clean, retrieve, scan, link; and executes comp.mak to compile a component's library.

Comp.mak Build script which compiles and assembles individual component module sources and executes armar.exe to generate libraries.

<customer>_<project>.mak (e.g. Monza_GPRS.mak)
Customer-project private configuration, including pre-processor definition, include path, modules etc.

Option.mak Project common options and macro definition build script.

Verno_Monza.mak Version number build script.

3.3 Module Option Files

The module option files include a module's include path, source files (C, Assembly) list, the include path for dependency checking, and optional definition files.

For example:

[D:\pvcs\maui\mcu\make\init\]
init.def
init.inc
init.lis
init.pth

init.lis Lists all the init module's source code files, including C and assembly sources.

Sample 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 Lists all the init module's private compile predefinition for C source code files.

Note:

- The current implementation does not consider assembly predefinitions.
- APCS_INTWORK is a special keyword to indicate that the module is compiled or assembled with APCS Interwork Specification (-apcs /interwork).

Sample content:

APCS_INTWORK
MTK_SUPPORT

init.pth Lists all the init module's source code directory from the mcu root directory.

Sample content:

init
init\src

init.inc Lists all the init module's included header files directory path from mcu root directory.

Sample content:

```
inc
inc\hwdrv
init
l1\common
```

Note: Some modules (e.g. nucleus) which contain Interwork and non-Interwork sources must be split into different modules for building. The nucleus module is split into nucleus for non-Interwork sources and nucleus_int for Interwork sources. Splitting a module improves performance; however the library is split into 2 libraries.

3.4 Intermediate Build Scripts and Log File

In addition to the PVCS Configuration Build internal temporary files, the main build script generates several intermediate build scripts to transfer needed information. All temporary files are named ~*.tmp in the make directory.

```
[D:\pvcs\maui\mcu\make\]
    ~buildinfo.tmp
    ~compbld.tmp
    Comp.mak
    Gsm2.mak
    Option.mak
    Verno_Monza.mak
```

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

Sample content:

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

~compbld.tmp Component modules needed build information.

Sample content:

```
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
COMPILER = RVCT

3.5 Generated Objects, Libraries, Executable Binary and Log Files

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

For example:

[D:\pvcs\maui\mcu\]
[build]
[Monza]
MT6218B.log
[gprs]
[MT6218Bo]
[MT6218Br]
[log]
…
[drv]
[init]
[nucleus]
[nucleus_int]

MT6218B.log Gsm2.mak generate this log to record the overall building process.

[log] Comp.mak generates a log file when building each component. The log records the compiling and archiving process of each component, and is placed in the log directory.

Build script generates and flushes the above log files if the script executes a **clean** action, or actions which depend on the **clean** action.

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

For example:

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

[MT6218Bo] Project generated objects, libraries directory.

[drv]
[init]
[nucleus]
[nucleus_int]
…

[lib] Component modules generated library directory.

For example:

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

4 Procedures and Functionality

Build/make procedures and functionality are listed below:

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

2. Make/build new GPRS project, clean all old objects, libraries, and log files etc.

   The **new** action also creates necessary directories, removes all temporary files, and flushes log files automatically.

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

   The **new** action depends on the **cleanall, asngen, codegen, asnregen, update** commands. Therefore, it cleans all intermediate scripts and log files, and calls scandep.exe to scan header file dependency, and builds component module's libraries, links, and generates an executable image file.

3. Clean all objects, temporary files, libraries, and executable binaries. The log file is also flushed.

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

   The **clean** action checks object directories, and creates them automatically if they do not exist. If no modules are indicated for cleaning, then by default, all modules are cleaned.

4. Clean dedicated init modules' object libraries. The log file is also flushed.

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

5. Update project dependency, and compile changed modules, link. Note that, the **update** and **remake** actions do not remove temporary files, nor do they flush the log file. Build results are appended after last log file.

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

   The **update** action depends on **cleanlog codegen genverno gencustominfo resgen scan remake**. Therefore, the module's objects and library are not cleaned. However, the **update** action checks dependency and remakes modified sources, links to binary. Neither **update** nor **remake** depends on **cleanall**; therefore, an object directory is not created, and the log file is not flushed. The new or clean action must be executed to flush the log file.

6. Recompile changed files, and link:

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

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

5 Description of Options

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

5.1 Core Software

Table 1: Core Software Options

Option	Location	Description
COMPILER	Monza_GPRS.mak	Set compiler; ADS | RVCT
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
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

Option	Location	Description
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
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.
__RVCT__	Option.mak	This define is for porting RVCT compiler.
__ADS__	Option.mak	This define is for porting ADS compiler.
_DEBUG	Option.mak	This option is for the debugging feature.

Option	Location	Description
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.
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.

5.2 MMI

Most MMI configuration is achieved via compile options or specific hard code in the MMI source code. If a customer has licensed MMI source code, the customer can modify it. By using compile options, configuration parameters can be found in MAK or MMI_featuresXXXXX.h. (where XXXXX is the project name).

Table 2: MMI Options

Configurable Items	Description	Entity Name	Configuration 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
Preferred Input Method	Customer can configure preferred input method function or default input method functionality	Input method	Compile Options	#define __MMI_PREFER_INPUT_METHOD__ to enable preferred 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

Configurable Items	Description	Entity Name	Configuration Method	Detail Configure methods
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
Dialing 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
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

Configurable Items	Description	Entity Name	Configuration Method	Detail Configure methods
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	Manufacturer can decide if SUBLCD exists	GUI	Compile Options	__MMI_SUBLCD__
Ring Tone Composer	Manufacturer can enable/disable this function	Fun & Games	Compile Options	__MMI_RING_COMPOSER__ && __MMI_IMELODY_SUPPORT__
Themes	Manufacturer can enable/disable this function	Fun & Games	Compile Options	__MMI_THEMES_APPLICATION__
Download	Manufacturer can enable/disable this function	Fun & Games	Compile Options	((__MMI_EMS__) || defined(__MMI_WAP__)) && __DOWNLOAD__
To Do List	Manufacturer can enable/disable this function	Organizer	Compile Options	__MMI_TODOLIST__
Calendar	Manufacturer can enable/disable this function	Organizer	Compile Options	__MMI_CALENDAR__
Unit Converter	Manufacturer can enable/disable this function	Organizer	Compile Options	__MMI_UNIT_CONVERTER__
Currency Converter	Manufacturer can enable/disable this function	Organizer	Compile Options	__MMI_CURRENCY_CONVERTER__
World Clock	Manufacturer can enable/disable this function	Organizer	Compile Options	__MMI_WORLD_CLOCK__

5.3 Applications

5.3.1 Socket and Data Account

Table 3: Socket and Data Account Options

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
Maximum Number of Data Accounts	Customer can set the maximum number 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 (e.g. WAP) so that applications can perform proper actions (e.g. disconnect)	Socket	Compile options / Link Time	1. AUTO_DISCONNECT_BEARER 2. soc_auto_disc_sec: 120 sec

5.3.2 WAP

Customers with licensed Teleca source code may configure the following compile options.

Table 4: WAP 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)
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

Configurable Items	Description	Entity Name	Configure method	Detail Configure methods
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)

6 Design and Implementation

This section lists the key sections in the build scripts.

6.1 Make.bat and M*.pl – Main Build Batch File

Make2.pl is implemented by perl script, and will call GNU make command to execute build action.

Core section:

```
    system("echo CUSTOMER=$custom > make\\~buildinfo.tmp");
    system("echo PROJECT=$project >> make\\~buildinfo.tmp");
    system("echo APLAT=$plat >> make\\~buildinfo.tmp");
    $timeStr = &CurrTimeStr;
    system("echo BUILD_DATE_TIME=$timeStr >> make\\~buildinfo.tmp");
    system("tools\\make.exe -fmake\\${myMF} -r -R CUSTOMER=$custom PROJECT=$project $action");
```

Explanation:

- echo is DOS build-in command and echo string on standard output or assigned output stream.
- tools\make.exe is GNU make executable.

6.2 GSM2.mak – Main Build Script

Core sections of GSM2.mak are listed 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 $(COMPLIBLIST).
- Action “cleanlib” cleans previous output files including *.log, *.lib.
- “tools\make.exe -fmake\comp.mak -r -R COMPONENT=$*” 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)
     @if exist $(strip $(TARGDIR))\$(strip $(BIN_FILE))\ROM \
      (@if exist $(strip $(FIXPATH))\custom\flash\$(strip $(BOARD_VER))\FlashConf.c \
        (perl .\tools\append.pl -c $(strip $(FIXPATH))\custom\flash\$(strip
     $(BOARD_VER))\FlashConf.c $(strip $(TARGDIR))\$(strip $(BIN_FILE))\ROM MTK_ROM_INFO_v01
     $(call Upper,$(strip $(BIN_FILE))) $(VERNO)) \
     ……
```

- armlink ($(LINK)) links libraries, and fromelf (BIN_CREATE) creates the binary.
- Append.pl appends the following information to ROM (multi-bin) or bin file for the use by flash tool.

A string noting the version of this format (16 chars)	17 bytes ("MTK_ROM_INFO_v01" with 0x00 ending)
Bin file name	64 bytes (with 0x00 ending)
Project id	64 bytes (with 0x00 ending)
Flash device count (FC)	1 byte
Flash info a. Manufacture id (2bytes) b. Device id (2bytes) c. Extended device code 1 (2bytes) d. Extended device code 2 (2bytes) e. FAT start address (4bytes) f. FAT length (4bytes)	16 bytes x FC
NFB (0xffff) or 0x0000	2 bytes
Bin file identifier	8 bytes ("MTK_BIN" with 0x00 ending)
The total size of above items	4 bytes (Integer)

6.3 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” adds xxx/yyy to the 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” adds XXX to the 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 by MediaTek with .lib only. These .lib are put in \mcu\mtk_lib. Components listed in CUS_REL_PAR_SRC_COMP also have libraries in \mcu\mtk_lib.

6.4 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
     ……

%.obj : %.cpp
     @echo Compiling $< ...
     @if exist tmp0.bat del /f /q tmp0.bat
     @tools\strcmpex.exe $(ACTION) remake e tmp0.txt  $(CINTWORK) $(CPLUSFLAGS) $(CDEFS)
     $(CINCDIRS) -o $(COMPOBJS_DIR)/$@ $<
     @tools\strcmpex.exe $(ACTION) remake n tmp0.txt  $(CINTWORK) $(CPLUSFLAGS) $(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 $(COMPOBJS_DIR)/$@ $<" is used to output a long line into tmp0.txt to avoid the DOS "command line too long" error. It echoes options for compiling .c into tmp0.txt and then "$(CMPLR) -via tmp0.txt" executes the compiler.
- `%.obj : %.cpp`: part is responsible for compiling C++ code.
- .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) $(CPPOBJS) $(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)

    @echo $(TARGLIB) is updated
```

- "$(TARGLIB) : $(COBJS) $(CPPOBJS) $(AOBJS)" is responsible for archiving a library from some objects.
- For existing library in \mcu\mtk_lib, "armar –r …" is invoked to replace newly generated objects. The command is used to create libraries listed in $(CUS_REL_PAR_SRC_COMP).
- For other libraries, "armar –c …" is invoked to create a new library.

7 Error Messages

In general, the error message is logged in a \build\Monza\MT6218B.log and \build\Monza\log*.log file. When compile or linking errors occur, details are recorded in the \build\Monza\log*.log or \build\Monza\MT6218B.log file, respectively.

- Environment not installed properly or not enough environment space.

  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

8 How to Customize the Build Environment

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

8.1 Add Modules to or Remove Modules from the Build Procedure

To complete this kind of configuration, the customer must understand the following variables in the make file Monza_GPRS.mak:

- COMPLIST: lists all source code modules that can be built into .lib. In an initial custom release, COMPLIST is the sum of CUS_REL_SRC_COMP and CUS_REL_PAR_SRC_COMP. The following is the initial setting in a 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: lists all modules provided with .lib only. These .lib are put in \mcu\mtk_lib.

8.1.1 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.

8.1.2 Remove a Source Module

1. Remove the module, for example "custom", from COMPLIST. Note that the module may be defined in CUS_REL_SRC_COMP or CUS_REL_PAR_SRC_COMP, instead of 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
```

8.1.3 Move a Source Module to a .lib Module

1. Remove the module, for example "media", from COMPLIST. Note that the module may be defined in CUS_REL_SRC_COMP or CUS_REL_PAR_SRC_COMP, instead of 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_lib, even if the media.lib already exists in \mcu\mtk_lib.

4. Delete the folder mcu\make\media.

Index of Tables