A. <u>Makefile.uc.mk **Local Variables**</u>
Local variables are only visible during compilation of current module.

1. INCLUDE_THIS_COMPONENT.
   If this value is equal to ‘y’ (or array of y’s: “y y y”) then this module will be included in build.
   All other values will be interpreted as ‘no’ and module will not be included in build.
   example: INCLUDE_THIS_COMPONENT := y

2. SRC.
   Add the files for compilations to this variable (for example: SRC += foo.c) .
   Supported types
   .c   - C compiler will be used.
   .cc, .cpp   - C++ compiler will be used.
   .s, .S   - Assembler will be used.

3. SPEED_CRITICAL_FILES.
   In supported architecture, files listed in this variable, will be optimized for speed.
   For example, in architecture with slow flash and fast RAM, functions/variable from these files
   will be located in RAM.
   sample usage: SPEED_CRITICAL_FILE += foo.c

4. INCLUDE_DIR.
   Add additional path for looking after included files.
   sample usage: INCLUDE_DIR += /lib/include   or   INCLUDE_DIR += ../foo_dir

5. DEFINES.
   Add additional defines.
   sample usage: DEFINES += USE_FIXED_POINT   or   DEFINES += VER_STR=”ver0.1"

6. CFLAGS.
   Additional flags for C compiler.
   example: CFLAGS += -Wpedantic

7. CPPFLAGS.
   Additional flags for C++ compiler.
   example: CPPFLAGS += -fno-nonansi-builtins

8. ASMFLAGS.
   Additional flags for assembler compiler.
   example: ASMFLAGS += -x assembler-with-cpp

9. ADDITIONAL_DEPS.
   Additional dependencies files.
   These files will be added to list of dependencies for *make* utility. These files are checked for changes
   before deciding whether perform compilation or skip it.
   example: ADDITIONAL_DEPS += src/file.c

10. VPATH.
    This is built-in (in make utility) variable to add search path for looking after files.
    example: `VPATH += src/commands/`

B. Makefile.uc.mk **Global Variables**

Global variables updated and visible for all compiled modules.

1. COMPILER_HOST_OS.
   Read only variable. Contain the name of host OS (windows, linux, …).

2. DATE_STR.
   Read only variable. Contain the current date.

3. TIME_STR.
   Read only variable. Contain the current time.

4. APP_ROOT_DIR.
   Read only variable. Contain the path to main folder of project.

5. OUT_DIR.
   Read only variable. Contain the path to the folder of compilation outputs.

6. PROJECT_NAME.
   Read only variable. Contain the name of project.

7. OUTPUT_APP_NAME.
   Read only variable. Contain the main part of output name.

8. COMMON_PUBLIC_DIR.
   Read only variable. Contain the path to common_public repository.

9. PARENT_OF_COMMON_PUBLIC_DIR.
   Read only variable. Contain the path to folder that contains common_public repository.

9. EXTERNAL_SOURCE_ROOT_DIR.
   Read only variable. Contain the path to folder that contains third party projects and repositories.

C. Makefile.uc.mk **Useful Functions**

Global variables updated and visible for all compiled modules.

1. ADD_TO_GLOBAL_DEFINES()

```
usage:  DUMMY := $(call ADD_TO_GLOBAL_DEFINES, your_define)
        DUMMY :=  $(call ADD_TO_GLOBAL_DEFINES, your_define=xx)
```

   This function will add ‘define’ (ex. : -D, #define in C/C++) variable to compilation line for all compiled files.

2. ADD_TO_GLOBAL_INCLUDE_PATH()
   usage: DUMMY := $(call ADD_TO_GLOBAL_INCLUDE_PATH, your/include/path)
   This function will add search path for includes (ex. : -I in C/C++) to compilation line for all compiled files.

3. ADD_TO_GLOBAL_CFLAGS()
   usage: DUMMY := $(call ADD_TO_GLOBAL_CFLAGS, flag)
   This function will add additional C flag to all compilation of all c files.

4. ADD_TO_GLOBAL_LDLAGS()
   usage: DUMMY := $(call ADD_TO_GLOBAL_LDFLAGS, flag)
   This function will add additional linker flag.

5. ADD_TO_GLOBAL_LIBS()
   usage: DUMMY := $(call ADD_TO_GLOBAL_LIBS, your_lib)
   This function will add additional library file for linkage. The filename will be formatted automatically, according to linker syntax. For example: for GCC use libfoo.a, it will be added to linker as -lfoo flag.

6. ADD_TO_GLOBAL_WHOLE_LIBS()
   usage: DUMMY := $(call ADD_TO_GLOBAL_LIBS, your_lib)
   This function will add additional library file for linkage. The library added as whole, without removing unused symbols. This is useful if some symbols are needed to be kept but are not referenced explicitly. The filename will be formatted automatically, according to linker syntax.
   For example: for GCC use libfoo.a, it will be added to linker as -lfoo flag.

7. ADD_TO_GLOBAL_LIBS_PATH()
   usage: DUMMY := $(call ADD_TO_GLOBAL_LIBS_PATH, path/to/lib/folder)
   This function will add additional search path for libraries.

8. sync additional GIT repository to specific hash commit:
   steps:
   a) create variable that will contain requested commit
   b) populate CURR_GIT_COMMIT_HASH_VARIABLE with name of variable from (a)
   c) populate CURR_GIT_REPO_DIR with path to requested repository
   d) include $(MAKEFILES_ROOT_DIR)/_include_functions/git_prebuild_repo_check.mk

   for example:

```
MY_HASH :="4fe2cfe3ff7a6f4a7083fa98f1ff07005d226ec8"
CURR_GIT_COMMIT_HASH_VARIABLE :=MY_HASH
CURR_GIT_REPO_DIR :=$( EXTERNAL_SOURCE_ROOT_DIR)/my_git_repo
include $(MAKEFILES_ROOT_DIR)/_include_functions/git_prebuild_repo_check.mk
```