Components.

1. Each component is folder with Makefile.uc.mk file that describe the component.
   Components can be located inside other components.

2. When build system start build process, it will perform the scan for all existing components.
   For C/C++ compilation following folders and their sub-folders are scanned:
   a) application folder
   b) common_public/c/sw_packages folder
   c) common_public/c/hw_drivers folder
   d) common_private/c folder

3. The list of all found components will be created in application folder in:
   z_auto_generated_files/**all_found_components.mk**
   this file has two main sections:
   a) first section with all found component.
   b) second section is used to decide what components will be included in build process.

   this file is build in makefile format to be processed later in the pre-build step.

4. Build process will decide to include or not to include the component according to
   $(INCLUDE_THIS_COMPONENT) or **$(INCLUDE_THIS_FOR_H_FILES_PATH)** variables in Makefile.uc.mk file.
   a) INCLUDE_THIS_COMPONENT := y - will include the component, the Makefile.uc.mk will be called by 'make' utility and component folder will be added in search path for include files.
   b) INCLUDE_THIS_FOR_H_FILES_PATH := y - will add component folder only in search path for include files.

5. The list of components that will be included in the build is created in application folder in:
   z_auto_generated_files/**include_components.mk** in $(SUBDIRS) variable
   this file is built in makefile format to be processed later in during the build.

6. Useful variables.
   main:
   a) **$(SRC)** - list of source files to compile. Can be C files (.c), C++ files (.cc, .cpp), assembler files (.S, .s) .
   b) **$(INCLUDE_DIR)** – list of additional search path for include files that will be passed to compiler (/I, -I ...). Valid only for building this component.
   c) ***$(DEFINES)*** – list of additional defines that will be passed to compiler (/D, -D ...). Valid only for building this component.
   d) **$(CFLAGS)** - list of additional flags that will be passed to C compiler. Valid only in this component.
   e) **$(CPPFLAGS)** - list of additional flags that will be passed to C++ compiler. Valid only in this component.
   f) **$(ASMFLAGS)** - list of additional flags that will be passed to ASM compiler. Valid only in this component.

   predefined variables:

a) **$(APP_ROOT_DIR)** – path to application folder. Can be used as a starting point to find additional files.
b) **$(COMMON_PUBLIC_DIR)** – path to common_public folder. Can be used as a starting point to find additional files.
c) **$(PARENT_OF_COMMON_PUBLIC_DIR)** – path to parent folder of common_public. Can be used as a starting point to find additional files.
d) **$(EXTERNAL_SOURCE_ROOT_DIR)** – path to folder with third party projects. Can be used as a starting point to find additional files.
e) **$(OUTPUT_APP_NAME)** – main application name for output binary.
f) **$(PROJECT_NAME)** – project name.
g) **$(COMPILER_HOST_OS)** – host OS detected by build system (Windows, Linux, ...).
h) **$(DATE_STR)** – date of current build.
j) **$(TTIME_STR)** – time of current build.

7. Useful functions.
a) **ADD_TO_GLOBAL_INCLUDE_PATH** – will add folder to global include path (/I, -I, ...).
usage:
DUMMY := $(call ADD_TO_GLOBAL_INCLUDE_PATH , your/include/path)

b) **ADD_TO_GLOBAL_DEFINES** – will add defines to global defines list (/D, -D, ...).
usage:
DUMMY := $(call ADD_TO_GLOBAL_DEFINES , your_define=xx)

c) **ADD_TO_GLOBAL_LIBS_PATH** – will add folder to search path of libraries (/L).
usage:
DUMMY := $(call ADD_TO_GLOBAL_LIBS_PATH , your/library/path)

d) **ADD_TO_GLOBAL_LIBS** – will add library to library list (gcc - .a files, armcc - .lib files, ...).
usage:
DUMMY := $(call ADD_TO_GLOBAL_LIBS , your_library)

e) **ADD_TO_GLOBAL_WHOLE_LIBS** – will add whole library to list of whole libraries.
(gcc - .a files, armcc - .lib files, ...). All object in these libraries will be included during link process, rather than searching the archive for the required object files.
(gcc example: -Wl,--whole-archive $(LIB_LIST) -Wl,--no-whole-archive)
usage:
DUMMY := $(call ADD_TO_GLOBAL_WHOLE_LIBS , your_library)

d) **ADD_TO_GLOBAL_CFLAGS** – will add C flags to be passed to compiler.
usage:
DUMMY := $(call ADD_TO_GLOBAL_CFLAGS , your_cpflags)

d) **ADD_TO_GLOBAL_LDFLAGS** – will add C flags to be passed to compiler.
usage:
DUMMY := $(call ADD_TO_GLOBAL_LDFLAGS , your_ldflags)

e) **$(call exit,1)** - function to stop build process. Good for debugging of build process.

f) **$(call get_curr_component_dir)** - function to calculate folder of current component.
Important!! - Should be used before any 'include' statement in makefile.

8. Creating new basic component in application folder.
   a) add new folder.
   b) add Makefile.uc.mk in folder from previous step.
   c) add INCLUDE_THIS_COMPONENT := y in Makefile.uc.mk from previous step.
   d) add include $(COMMON_CC) at the end of Makefile.uc.mk, this line should be always at the end of file.
   e) add source files into folder.
      Remember: folder will be added in search path for include files, so in the root folder of component is recommended to keep only API header files.
   f) add files you want to compile to $(SRC) variable. (Example: SRC += src/foo.c src/bar.c)
   g) run ‘make clean’ and then ‘make all’ (then build system will find new component and compile it)