Building an Application from Batch using Automake Make File (HC12)

Sometimes you may wish to build your application from batch (without using the IDE). In this case you have to create a make file, which will be used to build the whole application. This technical note explains how the make file from {Install}\(CodeWarrior_Examples)\MakeFileSample can be used to rebuild any application and explains:

- How you can use the automake file for building
- How you can adjust this automake make file to your own project.

The sample make file provided here is updating dependencies for ANSI C and C++ source files automatically.

Using Automake make file to build an Application

Automake process can be started through a simple DOS command line. So you can start the automake from a DOS shell, from a batch file or from any location where you can start a DOS command.

Command Line

In order to perform an incremental build (rebuild only source files which have changed or including files which have changed), type following command:

```
%INSTALLPATH%\prog\piper.exe %INSTALLPATH%\prog\maker.exe -WmsgVrb default.mak
```

In order to rebuild the whole application enter following command:

```
%INSTALLPATH%\prog\piper.exe %INSTALLPATH%\prog\maker.exe -WmsgVrb default.mak
rebuild
```

Environment Variable

Some environment variables needs to be defined for the make to work properly. These environment variables are:

Macro	Usage
GENPATH	Contains paths to all directories application source files are stored. Usually contains al least following CodeWarrior directories: `${INSTALLPATH}\lib\HC12c\src;${INSTALLPATH}\lib\HC12c\include;${INSTALLPATH}\lib\HC12c\lib` Add the paths to your application source files to this variable.
OBJPATH	Contain path to location where object files will be generated
ABSPATH	Contain path to location where executable file will be generated
TEXTPATH	Contain path to location where map, listing, dependency files will be generated

These environment variables can be defined:

1. On the Maker command line using –Env options
   In this case add following options to your command line:

```
-Env"GENPATH=%INSTALLPATH%\lib\hc12c\src;%INSTALLPATH%\lib\hc12c\lib"
-ENV"OBJPATH=.\bin" -ENV"ABSPATH=.\bin" -ENV"TEXTPATH=.\bin"
```

Where INSTALLPATH refers to the CodeWarrior installation path. All paths where application source files are stored must be specified in GENPATH environment variable.

2. At the DOS level
   Define the environment variable using SET commands:

```
SET INSTALLPATH=C:\Freescale
SET GENPATH=%INSTALLPATH%\lib\hc12c\src;%INSTALLPATH%\lib\hc12c\lib
SET OBJPATH=.\bin
SET ABSPATH=.\bin
SET TEXTPATH=.\bin
```

Adjusting Automake make file to your Project

In order to adjust the attached make file to your own project you have to adjust some macros to your own project. Each macro, which needs to be adjusted is described below.

Macro	Usage
PROJECT	Project name. The project name specified here is used as base name for the generated executable file, S record File or library file. Specified project name should not include any path information. Specified project name should not include spaces.

SOURCE_FILES	List of source files. This macro contains the list of all source files (C, C++, Assembly) used to build the application. Source files enumerated here are separated with spaces.
	Source files can be specified on one single line as follows:
	`SOURCE_FILES=Cpu.c IO_Map.c Vectors.c Bit1.c Events.c MyPrjt.c`
	Alternatively you can specify them on several lines using line continuation as follows:
	`SOURCE_FILES= Cpu.c \`
	`IO_Map.c \`
	`Vectors.c \`
	`Bit1.c \`
	`Events.c \`
	`Start12.c \`
	`MyPrjt.c`
	Make sure to specify the source file extensions with lowercase, otherwise the automake will not work appropriately.

ADD_OBJECTS	Additional binary files. This macro contains the list of all additional binary files (object files, library) used to build the application. Binary files enumerated here are separated with spaces. Additional binary files can be specified on one single lines or on several lines using line continuation. Usually contains at least the ANSI library file to link to the application. If you have changed the compiler options, refer to $(INSTALLPATH)\lib\hc12c\readme.txt to detect which ANSI library file should be used.

Macro	Usage
INSTALLPATH	Installation directory. This macro points to your CodeWarrior root installation directory.
LINKFILE	Linker file used to link the application.
MAKEFILE	Name of the make file used for building.
BUILD_TARGET_EXT	Indicates the target to generate. Set "**`BUILD_TARGET_EXT=abs`**" if you want to generate an executable file. Set "**`BUILD_TARGET_EXT=lib`**" if you wish to generate a library.
GENPATH	Contains paths to all directories where build tools will look for source files, library file, PRM file, and so on. Usually contains al least following CodeWarrior directories: `${INSTALLPATH}\lib\HC12c\src;${INSTALLPATH}\lib\HC12c\include;${INSTALLPATH}\lib\HC12c\lib` Add the paths to your application source files to this environment variable.
INCLUDEPATH	Contains paths to system library and application include files. Usually contains at least following CodeWarrior directory: `${INSTALLPATH}\lib\HC12c\include`
OBJPATH	Contain path to location where object files will be generated.
ABSPATH	Contain path to location where executable file will be generated.
TEXTPATH	Contain path to location where map, listing, dependency files will be generated.
CCFLAGS_COMMON	Compiler options. Add all options you wish to use to compile your C and C++ files. Common options applying to both C and C++ files should b specified here. If you have an existing CodeWarrior project build with the IDE, add options specified in the compiler panel here
CCFLAGS	Compiler options. Add all options you wish to use to compile your C files only.
CPPFLAGS	Compiler options. Add all options you wish to use to compile your C++ files only
AAFLAGS	Assembler options. Add all options you wish to use to assemble your assembly files. If you have an existing CodeWarrior project build with the IDE, add options specified in the assembler panel here.
LLFLAGS	Linker options. Add all options you wish to use to link your application. . If you have an existing CodeWarrior project build with the IDE, add options specified in the linker panel here.

Limitation

Note around assembly source files

The automake make file uses the ability of the compiler to automatically generate dependency list for ANSI C and C++ source files.
If you have assembly source file in your application, you have to insert the dependency list for each assembly source file in the make file manually.
A typical dependency list looks as follows:

```
target file: {dependency file}
```

For example if you have an assembly source file called MyFile.asm, which includes file MyInclude.h and derivative .h, the dependency list for generating the corresponding object file looks like:

```
MyFile.o: MyFile.asm MyInclude.h derivative .h
```

Note around file extension

The attached automake make file works fine for standard file extensions .c, .cpp, .asm and .s.

Note around burner

The attached sample make files are using burner options to build the s record file. If you need to get more control over the generation of the S19 file, you can use a so called burner command file (.bbl file) to generate the S record.
In that case replace the macro:

```
BBFLAGS=   $(ENVOPTIONS) \
  OPENFILE "$(PROJECT).s19" \
  format = motorola \
  origin = 0x0000 \
  len = 0xffffffff \
  busWidth = 1 \
  SENDBYTE 1 "$(PROJECT).abs"
```

by

```
BBFLAGS=   $(ENVOPTIONS) -F=default.bbl
```

Please refer to the Burner reference manual for more info on burner command file syntax.

Appendix: default.mak Make File

```
##############################
# Auto-make file: default.mak
##############################

# Following variables should be updated to reflect you current project settings.
# PROJECT:              Project name. Name of the final absolute/S19/library file,
#                      without path information
PROJECT=MyProject

# SOURCE_FILES:                              All C/C++/ASM files (without path)
SOURCE_FILES=datapage.c \
             Start12.c \
             main.c

# ADD_OBJECTS:                         Additional object/library files to link with
ADD_OBJECTS=ansisi.lib

# INSTALLPATH:          Path where tools are installed (root installation
#                       directory). If called from build.bat, INSTALLPATH
#                       is through -D option.
#                       Remove comment below and adjust path if INSTALLPATH is not
#                       set on make invocation line
#INSTALLPATH=C:\Freescale\CodeWarrior_HC12
# LINKFILE:               Linker parameter file to be used
LINKFILE=.\prm\project.prm
```

```
# MAKEFILE:               Name of this make file
MAKEFILE=default.mak

# define here if you want to build a absolute file (abs) or a library (lib)
BUILD_TARGET_EXT=abs
#BUILD_TARGET_EXT=lib

#INCLUDEPATH:           Paths where application or system include files can be
#                       found.
INCLUDEPATH="$(INSTALLPATH)\lib\hc12c\include"

#GENPATH:                Pathes where application source and prm files can be
#                        found
LIB=$(INSTALLPATH)\lib\hc12c
GENPATH=.\Sources;$(LIB)\src;$(LIB)\lib\hc12c\include;$(LIB)\lib\hc12c\lib;

#OBJPATH:                Path where compiler and assembler will place object files
OBJPATH=.\bin
#ABSPATH:                Path where linker will place executable file
ABSPATH=.\bin
#TEXTPATH:               Path where linker will place map file and compiler
#                        disassembly listing file and dependency file.
TEXTPATH=.\bin

# The name of the dependency list file. Here the compiler itself updates the
# make file!
DEP_LIST_FILE = $(PROJECT).dep

#ENVOPTIONS:             Command line options to set environment variables.
ENVOPTIONS=-ENV"GENPATH=$(GENPATH)" -ENV"OBJPATH=$(OBJPATH)" \
  -ENV"ABSPATH=$(ABSPATH)" -ENV"TEXTPATH=$(TEXTPATH)"

# Here there are some special options to select the object file format for
# compiler/assembler/linker
CC_OBJ_FORMAT=-F2
AA_OBJ_FORMAT=-F2
LL_OBJ_FORMAT=-Fe

# Compiler settings, C and C++ options.
# Short description about the options used (you may adapt this options to your
# own need):
#  -Lasm:    Produce listing file
#  -Lm:      Generate dependency information
#  -LmCfg:   Configuration options for -Lm: m=write path of main file, i=write
#            path of included files, u=update information
#  -D:       Add preprocessor define, here it defines the target derivative name
#  -WmsgFob: Option to configure the message output format. Here we changed the
#            format so that the column number is written too
#  -C++f:    Use full C++ mode for C++ files
CC=$(INSTALLPATH)\prog\cHC12.exe
CCFLAGS_COMMON= $(CC_OBJ_FORMAT) -Lasm -Lm=$(DEP_LIST_FILE) -WmsgNu=abcet \
                -I$(INCLUDEPATH) -LmCfg=ilmu $(ENVOPTIONS)
CCFLAGS=$(CCFLAGS_COMMON)
CPPFLAGS= $(CCFLAGS_COMMON) -C++f

# Assembler settings
# Short description about the options used (you may adapt this options to your
# own need):
AA=$(INSTALLPATH)\prog\aHC12.exe  $(AA_OBJ_FORMAT) -WmsgNu=abcet $(ENVOPTIONS)
AAFLAGS=

# linker settings
# Short description about the options used (you may adapt this options to your
# own need):
#  -M:       Generate map file (overwrites the optional MAPFILE command in the
#            linker parameter file
#  -O:       Name of the output file. Overwrites the optional one specified in the
#            linker parameter file
```

```
#  -Add:     The list of object/library files specified here is linked to the
#            application (together with the ones specified in the linker parameter
#            file)
LL=$(INSTALLPATH)\prog\linker.exe
LLFLAGS=  -M -O$(PROJECT).abs -WmsgNu=abcet -Add($(ALL_OBJ_FILES) $(ADD_OBJECTS)) \
          $(LL_OBJ_FORMAT) $(ENVOPTIONS)

# burner settings, only used for S-Record, Intel HEX or binary output files
# Short description about the options used (you may adapt this options to your
# own need):
#  OPENFILE: Creates/opens the destination file
#  format:   Format of the file, here it is motorola S-Record
#  origin:   Start address from where to copy, here begin of memory (address zero)
#  len:      Set to maximum, that is the whole application
#  busWidth: Set to one, thus no splitting between different eeproms
#  SENDBYTE: Starts generation of the output
BB=$(INSTALLPATH)\prog\burner.exe
BBFLAGS=   $(ENVOPTIONS) \
  OPENFILE "$(PROJECT).s19" \
  format = motorola \
  origin = 0x0000 \
  len = 0xffffffff \
  busWidth = 1 \
  SENDBYTE 1 "$(PROJECT).abs"

# libmaker settings, only used for libraries
# Short description about the options used (you may adapt this options to your
# own need):
LM=$(INSTALLPATH)\prog\libmaker.exe
LMFLAGS= $(ENVOPTIONS) -NoPath -Cmd"$(ALL_OBJ_FILES) $(ADD_OBJECTS) = $(PROJECT).lib"

################################################################################
# from here you don't need to adapt the automake file
################################################################################
# build object file list with search and replace operation
CPP_OBJ_FILES = $(SOURCE_FILES:.cpp=.o)
CXX_OBJ_FILES = $(CPP_OBJ_FILES:.cxx=.o)
C_OBJ_FILES   = $(CXX_OBJ_FILES:.c=.o)
S_OBJ_FILES   = $(C_OBJ_FILES:.s=.o)
ALL_OBJ_FILES = $(S_OBJ_FILES:.asm=.o)

#####################################
# rules
#####################################
# top target
all: $(PROJECT).$(BUILD_TARGET_EXT)
  echo *** $(PROJECT) is up to date. ***

# this targets enforces a rebuild:
# 1) delete the dependency info file (may contain outdated information)
# 2) call a second maker process to build the target again, with following options:
#    $(MAKEFILE):  Make file, which is this file
#     -Prod:        Project file with all other settings
#     -WmsgVrb:     Use verbose mode, so we know what is going on
#     -D:           Pass through macro variables we got from the invocation line
#                   directly (INSTALLPATH)
#     -ViewMin:     Run as iconified application
#     -Mkall:       Make the target without checking for dependency dates
rebuild:
  -del $(DEP_LIST_FILE)
  -del $(TEXTPATH)\$(DEP_LIST_FILE)
  $(INSTALLPATH)\prog\maker.exe $(MAKEFILE) $(ENVOPTIONS) -WmsgVrb \
              -D(INSTALLPATH=$(INSTALLPATH)) -ViewMin -mkall
  echo "*** rebuild done ***"

$(PROJECT).abs: $(LINKFILE) $(MAKEFILE) $(ALL_OBJ_FILES) $(ADD_OBJECTS)
  $(LL) $(LLFLAGS) $(LINKFILE)
  $(BB) $(BBFLAGS)
```

```
$(PROJECT).lib: $(MAKEFILE) $(ALL_OBJ_FILES) $(ADD_OBJECTS)
  $(LM) $(LMFLAGS)

# various rules how to build an object file from a source file
.c.o:
  $(CC) $(CCFLAGS) $*.c

.cpp.o:
  $(CC) $(CPPFLAGS) $*.cpp

.cxx.o:
  $(CC) $(CPPFLAGS) $*.cxx

.asm.o:
  $(AA) $(AAFLAGS) $*.asm

.s.o:
  $(AA) $(AAFLAGS) $*.s

########################################################
#dependency section, generated/updated by the compiler
########################################################

INCLUDE $(DEP_LIST_FILE)
```