# Makefile

A makefile is a text file that is referenced by the make command that describes the building of targets, and contains information such as source-level dependencies and build-order dependencies.

The CDT can generate a makefile for you, such projects are called Managed Make projects. Some projects, known as Standard Make projects, allow you to define your own makefile.

## Sample Makefile

# A sample Makefile  
# This Makefile demonstrates and explains   
# Make Macros, Macro Expansions,  
# Rules, Targets, Dependencies, Commands, Goals  
# Artificial Targets, Pattern Rule, Dependency Rule.  
  
# Comments start with a # and go to the end of the line.  
  
# Here is a simple Make Macro.  
LINK\_TARGET = test\_me.exe  
  
# Here is a Make Macro that uses the backslash to extend to multiple lines.  
# This allows quick modification of more object files.  
OBJS = \  
 Test1.o \  
 Test2.o \  
 Main.o  
  
# Here is a Make Macro defined by two Macro Expansions.  
# A Macro Expansion may be treated as a textual replacement of the Make Macro.  
# Macro Expansions are introduced with $ and enclosed in (parentheses).  
REBUILDABLES = $(OBJS) $(LINK\_TARGET)  
  
# Make Macros do not need to be defined before their Macro Expansions,  
# but they normally should be defined before they appear in any Rules.  
# Consequently Make Macros often appear first in a Makefile.  
  
# Here is a simple Rule (used for "cleaning" your build environment).  
# It has a Target named "clean" (left of the colon ":" on the first line),  
# no Dependencies (right of the colon),  
# and two Commands (indented by tabs on the lines that follow).  
# The space before the colon is not required but added here for clarity.  
clean :   
 rm -f $(REBUILDABLES)  
 echo Clean done  
  
# There are two standard Targets your Makefile should probably have:  
# "all" and "clean", because they are often command-line Goals.  
# Also, these are both typically Artificial Targets, because they don't typically  
# correspond to real files named "all" or "clean".   
  
# The rule for "all" is used to incrementally build your system.  
# It does this by expressing a dependency on the results of that system,  
# which in turn have their own rules and dependencies.  
all : $(LINK\_TARGET)  
 echo All done  
  
# There is no required order to the list of rules as they appear in the Makefile.  
# Make will build its own dependency tree and only execute each rule only once  
# its dependencies' rules have been executed successfully.  
  
# Here is a Rule that uses some built-in Make Macros in its command:  
# $@ expands to the rule's target, in this case "test\_me.exe".  
# $^ expands to the rule's dependencies, in this case the three files  
# main.o, test1.o, and test2.o.  
$(LINK\_TARGET) : $(OBJS)  
 g++ -g -o $@ $^  
  
# Here is a Pattern Rule, often used for compile-line.  
# It says how to create a file with a .o suffix, given a file with a .cpp suffix.  
# The rule's command uses some built-in Make Macros:  
# $@ for the pattern-matched target  
# $lt; for the pattern-matched dependency  
%.o : %.cpp  
 g++ -g -o $@ -c $<  
  
# These are Dependency Rules, which are rules without any command.  
# Dependency Rules indicate that if any file to the right of the colon changes,  
# the target to the left of the colon should be considered out-of-date.  
# The commands for making an out-of-date target up-to-date may be found elsewhere  
# (in this case, by the Pattern Rule above).  
# Dependency Rules are often used to capture header file dependencies.  
Main.o : Main.h Test1.h Test2.h  
Test1.o : Test1.h Test2.h  
Test2.o : Test2.h  
  
# Alternatively to manually capturing dependencies, several automated  
# dependency generators exist. Here is one possibility (commented out)...  
# %.dep : %.cpp  
# g++ -M $(FLAGS) $< > $@  
# include $(OBJS:.o=.dep)

## Frequently Asked Questions:

Your Console view can be very useful for debugging a build.

##### **Q1**. My Console view says **"Error launching builder"**. What does that mean?

Error launching builder (make -k clean all )  
(Exec error:Launching failed)

Most probably, the build command (by default "make") is not on your path. You can put it on your path and restart Eclipse.

You can also change the build command to something that is on your path. If you are using MinGW tools to compile, you should replace the build command with "mingw32-make".

##### **Q2**. My Console view says **"No rule to make target 'X'"**.

make -k clean all   
make: \*\*\* No rule to make target 'clean'.  
make: \*\*\* No rule to make target 'all'.

By default, the make program looks for a file most commonly called "Makefile" or "makefile". If it cannot find such a file in the working directory, or if that file is empty or the file does not contain rules for the command line goals ("clean" and "all" in this case), it will normally fail with an error message similar to those shown.

If you already have a valid Makefile, you may need to change the working directory of your build. The default working directory for the build command is the project's root directory. You can change this by specifying an alternate Build Directory in the Make Project properties. Or, if your Makefile is named something else (eg. buildFile.mk), you can specify the name by setting the default Build command to make -f buildFile.mk.

If you do not have a valid Makefile, create a new file named Makefile in the root directory. You can then add the contents of the sample Makefile (above), and modify it as appropriate.

##### **Q3**. My Console view says **"missing separator"**.

make -k clean all   
makefile:12: \*\*\* missing separator. Stop.

The standard syntax of Makefiles dictates that every line in a build rule must be preceded by a Tab character. This Tab character is often accidentally replaced with spaces, and because both result in white-space indentation, this problem is easily overlooked. In the sample provided, the error message can be pinpointed to line 12 of the file "makefile"; to fix the problem, insert a tab at the beginning of that line.

##### **Q4**. My Console view says **"Target 'all' not remade because of errors"**.

make -k clean all   
make: \*\*\* [clean] Error 255  
rm -f Test1.o Test2.o Main.o test\_me.exe  
g++ -g -o Test1.o -c Test1.cpp  
make: \*\*\* [Test1.o] Error 255  
make: \*\*\* [Test2.o] Error 255  
make: \*\*\* [Main.o] Error 255  
g++ -g -o Test2.o -c Test2.cpp  
g++ -g -o Main.o -c Main.cpp  
make: Target 'all' not remade because of errors.

The likely culprit here is that g++ is not on your Path.

The Error 255 is produced by make as a result of its command shell not being able to find a command for a particular rule.

Messages from the standard error stream (the lines saying Error 255) and standard output stream (all the other lines) are merged in the Console view here.

##### **Q5**. What's with the -k flag?

The -k flag tells make to continue making other independent rules even when one rule fails. This is helpful for build large projects.

You can remove the -k flag by turning on Project Properties > C/C++ Make Project > Make Builder > Stop on first build error

##### **Q6**. My Console view looks like:

mingw32-make clean all   
process\_begin: CreateProcess((null), rm -f Test1.o Test2.o Main.o test\_me.exe, ...) failed.  
make (e=2): The system cannot find the file specified.  
  
mingw32-make: \*\*\* [clean] Error 2  
rm -f Test1.o Test2.o Main.o test\_me.exe

This means that mingw32-make was unable to find the utility "rm". Unfortunately, MinGW does not come with "rm". To correct this, replace the clean rule in your Makefile with:

clean :   
 -del $(REBUILDABLES)  
 echo Clean done

The leading minus sign tells make to consider the clean rule to be successful even if the del command returns failure. This may be acceptable since the del command will fail if the specified files to be deleted do not exist yet (or anymore).

