Silicon Graphics Inc
Common Makefile include files

NOTES

Publication Date: 5/6/93

Overview

- Overview
  - Why are they useful?
  - What is common?
  - Where do I go for more information?
  - What common actions are expected of you?
- **commondefs and commonrules** (page 6)
  - Used implicitly almost everywhere
  - Used in source directories (both parent and leaf directories)
  - Most typical include files

Why are they useful?

- Files to include in your makefiles
- Simplify and shorten Makefiles
- Provide common targets for common actions
- Provide Tool abbreviations and option management
- Support cross compilation
  - Run ppropriate tools from $TOOLROOT
  - Access appropriate header files and libraries from $ROOT
- NOTE: this document applies to Sherwood (5.0), or subsequent, releases

What is common?

- Some common targets are typically provided
  - **clean** remove TARGET by-products (*.o files, etc.)
  - **clobber** remove all but source
  - **rmtargets** just remove $(TARGETS)
- Other {defs/rules} files tend to extend common{defs/rules}

What common actions are expected of you?

- A common Makefile structure, namely:
  - Include the ***defs*** file
  - Add your local definitions and overrides
  - Add your default Makefile target
  - Include the ***rules*** file
  - Add your other Makefile targets
- Some common targets

– **default**	produce the targets of this directory only and pass the target to a make in appropriate subdirectories. manrules and relnoterules are an exception, they provide this target.
– **install**	same as default plus invoke $(INSTALL) on the targets, with appropriate idb tag info manrules and relnoterules are an exception, they provide this target.

Where do I go for more information?

- Use the handbook commands to access the following chapters:

– **makeconv**	SGI Makefile Conventions
– **ism**	ISM Class notes (Independent Software Module)
– **buildman**	Building Manual Pages and Release Notes
– **SGIintro**	Pointers to Engineering Resources at SGI. It has information on some of the code names for releases and products at SGI.

commondefs and commonrules

- base set of macros and targets used in most SGI Makefiles, primarily focused on producing executables
- Macros and Flags Provided
  - Compiler and Loader option flags (page 7)
  - Flag Defaults (page 9)
  - Tool Abbreviations (via macros) (page 10)
  - Combined Compiler macro name and flags (page 11)
  - Other macros (page 12)
- Other features
- Common Targets
- Makefile implications
  - Makefile Structure (page 14)
  - Overriding the default rules (page 16)
  - Sample Makefile (page 17)

Compiler and Loader option flags

- Three sources of definitions
  - **G** global, defined in commondefs
  - **L** local, defined in Makefile
  - **V** variable, defined on Makefile command line
  - V overrides L which overrides G
- Three types of compiler flags
  - **DEFS** Mechanism to pass environment variables to source
  - **INCS** Directories to search to satisfy #includes
  - **OPTS** all other compiler options
- Four types of compilers
  - **C** C
  - **C++** C++
  - **F77** FORTRAN 77
  - **P** Pascal
- Three types of loader flags
  - **LIBS** directories to search to satisfy external references
  - **DSO** for making Dynamic Shared Objects (with lib rules)
  - **OPTS** all other loader options

Compiler and Loader option flags (continued)

- Order of components to form flag name
  - source
  - compiler letters or ‘LD’ for loader
  - type of flag
  - Example (of C Defs local to your Makefile)
    - LCDEFS =- -DCPUTYPE
- Flags are combined for you (based on the option flags you provide)
  - **CFLAGS**
  - **C++FLAGS**
  - **FFLAGS**
  - **PFLAGS**
  - **LDFLAGS**
- **C++FLAGS** are used with YACC and LEX compilations

Flag Defaults

- use of $(ROOT)/usr/include instead of /usr/include
- for C, C++, P, F77, YACC and LEX
  - Automatic Makefile dependencies to file Makedepend (MKDEPOPT)
- for C, C++, P, and F77
  - Optimization turned on (OPTIMIZER)
  - Correct endian (based on ***PRODUCT***defs file) (ENDIAN)
- for C
  - Some warnings turned off (WOFF)
- for C++
  - use of $(ROOT)/usr/include/cc instead of /usr/include/cc
  - C++ defines:

    _MODERN_C
    _SVR4_SOURCE
    _SGI_SOURCE
- for LD
  - use of $(ROOT)/{lib, usr/lib} instead of {/lib, /usr/lib}

Flag Overrides

- OPTIMIZER, WOFF can be overridden by unsetting the macro
- To select different dependency file, set macro MKDEPFILE

Tool Abbreviations (via macros)

- Compilers
  - CC
  - C++ (or C++C)
  - PC
  - F77 (or FC)
  - HOST_CC
  - HOST_C++
- Compiler Related
  - AS
  - LEX
  - LINT
  - MKF2C
  - YACC
  - NM
- Linking and Loading
  - AR
  - LD
  - LIBSPEC
  - LORDER
  - MKSHLIB
  - SIZE
  - STRIP
  - RANLIB
- Miscellaneous
  - INSTALL
  - SHELL
- Supported, but not recommended (i.e. reference the command directly, since its function is independant of specific release or hardware platform)
  - AWK
  - NAWK
  - ECHO
  - M4
  - MKF2C

Combined Compiler macro name and flags

- $(C++F) is equivalent to $(C++) $(C++FLAGS)
- Similarly (i.e. substitute new letter in three places for C++)
  - PC for Pascal
  - LD for loader
  - AS for assembler
  - LEX for Lex
  - YACC for YACC
- Exceptions
  - $(CCF) is equivalent to $(CC) $(CFLAGS)
  - $(F77F) is equivalent to $(F77) $(FFLAGS)

Other macros

- DIRT macros -> {V,L,G,X}DIRT
  - **{V,L,G}DIRT** define target by-products to be removed by the clean target
  - **XDIRT** for use of any enclosing set of commondefs, e.g. ismcommondefs
- subdirectory support - **SUBDIRS_MAKERULE**
  - **SUBDIRS_MAKERULE** as a command for a target, will execute the same target in each subdirectory of the macro SUBDIRS (echoing what it is going to do to stdout)
  - can override the target passed to the subdirectories by setting the envariable ROOT to be the desired target
  - can override the default action for each directory by resetting the macro SUBDIR_MAKERULE
  - set macro NOSUBMESG to yes and any non-existent directory of the SUBDIRS macro will not be reported as an error.
- macros for other common Makefile include files
  - library DSO support
  - **HEADERS_SUBDIRS_MAKERULE** identical to SUBDIRS_MAKERULE except operates on HEADERS_SUBDIRS instead of SUBDIRS
  - **EXPORTS_SUBDIRS_MAKERULE** identical to SUBDIRS_MAKERULE except operates on EXPORTS_SUBDIRS instead of SUBDIRS

Other Features

- local{defs,rules}
  - included at end of corresponding common file, if present in the current directory
  - useful for defining things part of your build process, but not part of build groups build process, e.g. setting DSOREGFILE
  - CAUTION: in general do not want any differences between your build process and that of build groups build process
  - can override default file name via macros LOCALDEFS and LOCALRULES.
  - To access such a file located outside of the current directory, then set LOCALDEFS or LOCALRULES on the command line. You can also just include the file from within a local LOCALDEFS or LOCALRULES file.

Makefile Structure

- You define your source files via the macros:

– **CFILES**	for C code
– **C++FILES**	for C++ code
– **ASFILES**	for Assembly code
– **FFILES**	for FORTRAN code
– **RFILES**	for RATFOR code (see f77(1))
– **EFILES**	for RATFOR code (see f77(1))
– **PFILES**	for Pascal code
– **LFILES**	for Lex code implicit rules of ***name***.l into ***name***.o rather than lex.yy.o
– **YFILES**	for YACC code implicit rules of ***name***.y into ***name***.o rather than y.tab.o
– **SHFILES**	for a shell scripts
– **HFILES**	for header files

- You define the list of targets to be made by the Makefile
  - **TARGETS**
- You define the default rule (typically just =>
  default: $(TARGETS))

Makefile Structure (continued)

- You are provided:
  - OBJECTS macro, the list of objects produced by the makefile, derived from the FILES macros you define.
  - default targets:
  - **clean** (Standard SGI makefile target)
  - **clobber** (Standard SGI makefile target)
  - **rmtargets** (Standard SGI makefile target)
  - **fluff** run lint on source files
  - **tags** form ctags file from sources for use with vi and other editors

Overriding the default rules

- by renaming them, as well as,
- via setting macro COMMONPREF to a unique value
- For instance,

  COMMONPREF = xyzzy

  makes commonrules define

  xyzzyclean, xyzzyclobber, etc.

  instead of

  clean, clobber, etc

Sample Makefile

```
include $(ROOT)/usr/include/make/commondefs
# <definitions of this Makefile, last macro definition wins>
CFILES = foo.c bar.c
TARGETS = foo
default: $(TARGETS)
include $(COMMONRULES)
install: default
          $(INSTALL) ...
foo: $(OBJECTS)
          $(CCF) $(OBJECTS) $(LDFLAGS) -o $@
# <rules of this Makefile>
```