# CMakeLists.txt

# Copyright (C) 2007,2009-2017 Glenn Randers-Pehrson

# Written by Christian Ehrlicher, 2007

# Revised by Roger Lowman, 2009-2010

# Revised by Clifford Yapp, 2011-2012

# Revised by Roger Leigh, 2016

# Revised by Andreas Franek, 2016

# This code is released under the libpng license.

# For conditions of distribution and use, see the disclaimer

# and license in png.h

cmake\_minimum\_required(VERSION 3.0.2)

cmake\_policy(VERSION 3.0.2)

# Set MacOSX @rpath usage globally.

if (POLICY CMP0020)

cmake\_policy(SET CMP0020 NEW)

endif(POLICY CMP0020)

if (POLICY CMP0042)

cmake\_policy(SET CMP0042 NEW)

endif(POLICY CMP0042)

# Use new variable expansion policy.

if (POLICY CMP0053)

cmake\_policy(SET CMP0053 NEW)

endif(POLICY CMP0053)

if (POLICY CMP0054)

cmake\_policy(SET CMP0054 NEW)

endif(POLICY CMP0054)

set(CMAKE\_CONFIGURATION\_TYPES "Release;Debug;MinSizeRel;RelWithDebInfo")

project(libpng ASM C)

enable\_testing()

set(PNGLIB\_MAJOR 1)

set(PNGLIB\_MINOR 6)

set(PNGLIB\_RELEASE 34)

set(PNGLIB\_NAME libpng${PNGLIB\_MAJOR}${PNGLIB\_MINOR})

set(PNGLIB\_VERSION ${PNGLIB\_MAJOR}.${PNGLIB\_MINOR}.${PNGLIB\_RELEASE})

include(GNUInstallDirs)

# needed packages

#Allow users to specify location of Zlib,

# Useful if zlib is being built alongside this as a sub-project

option(PNG\_BUILD\_ZLIB "Custom zlib Location, else find\_package is used" OFF)

IF(NOT PNG\_BUILD\_ZLIB)

find\_package(ZLIB REQUIRED)

include\_directories(${ZLIB\_INCLUDE\_DIR})

ENDIF(NOT PNG\_BUILD\_ZLIB)

if(NOT WIN32)

find\_library(M\_LIBRARY

NAMES m

PATHS /usr/lib /usr/local/lib

)

if(NOT M\_LIBRARY)

message(STATUS "math lib 'libm' not found; floating point support disabled")

endif()

else()

# not needed on windows

set(M\_LIBRARY "")

endif()

# COMMAND LINE OPTIONS

option(PNG\_SHARED "Build shared lib" ON)

option(PNG\_STATIC "Build static lib" ON)

option(PNG\_TESTS "Build libpng tests" ON)

# Many more configuration options could be added here

option(PNG\_FRAMEWORK "Build OS X framework" OFF)

option(PNG\_DEBUG "Build with debug output" OFF)

option(PNGARG "Disable ANSI-C prototypes" OFF)

option(PNG\_HARDWARE\_OPTIMIZATIONS "Enable Hardware Optimizations" ON)

set(PNG\_PREFIX "" CACHE STRING "Prefix to add to the API function names")

set(DFA\_XTRA "" CACHE FILEPATH "File containing extra configuration settings")

if(PNG\_HARDWARE\_OPTIMIZATIONS)

# set definitions and sources for arm

if(CMAKE\_SYSTEM\_PROCESSOR MATCHES "^arm" OR

CMAKE\_SYSTEM\_PROCESSOR MATCHES "^aarch64")

set(PNG\_ARM\_NEON\_POSSIBLE\_VALUES check on off)

set(PNG\_ARM\_NEON "check" CACHE STRING "Enable ARM NEON optimizations:

check: (default) use internal checking code;

off: disable the optimizations;

on: turn on unconditionally.")

set\_property(CACHE PNG\_ARM\_NEON PROPERTY STRINGS

${PNG\_ARM\_NEON\_POSSIBLE\_VALUES})

list(FIND PNG\_ARM\_NEON\_POSSIBLE\_VALUES ${PNG\_ARM\_NEON} index)

if(index EQUAL -1)

message(FATAL\_ERROR

" PNG\_ARM\_NEON must be one of [${PNG\_ARM\_NEON\_POSSIBLE\_VALUES}]")

elseif(NOT ${PNG\_ARM\_NEON} STREQUAL "no")

set(libpng\_arm\_sources

arm/arm\_init.c

arm/filter\_neon.S

arm/filter\_neon\_intrinsics.c)

if(${PNG\_ARM\_NEON} STREQUAL "on")

add\_definitions(-DPNG\_ARM\_NEON\_OPT=2)

elseif(${PNG\_ARM\_NEON} STREQUAL "check")

add\_definitions(-DPNG\_ARM\_NEON\_CHECK\_SUPPORTED)

endif()

else()

add\_definitions(-DPNG\_ARM\_NEON\_OPT=0)

endif()

endif()

# set definitions and sources for powerpc

if(CMAKE\_SYSTEM\_PROCESSOR MATCHES "^powerpc\*" OR

CMAKE\_SYSTEM\_PROCESSOR MATCHES "^ppc64\*" )

set(PNG\_POWERPC\_VSX\_POSSIBLE\_VALUES on off)

set(PNG\_POWERPC\_VSX "on" CACHE STRING "Enable POWERPC VSX optimizations:

off: disable the optimizations.")

set\_property(CACHE PNG\_POWERPC\_VSX PROPERTY STRINGS

${PNG\_POWERPC\_VSX\_POSSIBLE\_VALUES})

list(FIND PNG\_POWERPC\_VSX\_POSSIBLE\_VALUES ${PNG\_POWERPC\_VSX} index)

if(index EQUAL -1)

message(FATAL\_ERROR

" PNG\_POWERPC\_VSX must be one of [${PNG\_POWERPC\_VSX\_POSSIBLE\_VALUES}]")

elseif(NOT ${PNG\_POWERPC\_VSX} STREQUAL "no")

set(libpng\_powerpc\_sources

powerpc/powerpc\_init.c

powerpc/filter\_vsx\_intrinsics.c)

if(${PNG\_POWERPC\_VSX} STREQUAL "on")

add\_definitions(-DPNG\_POWERPC\_VSX\_OPT=2)

endif()

else()

add\_definitions(-DPNG\_POWERPC\_VSX\_OPT=0)

endif()

endif()

# set definitions and sources for intel

if(CMAKE\_SYSTEM\_PROCESSOR MATCHES "^i?86" OR

CMAKE\_SYSTEM\_PROCESSOR MATCHES "^x86\_64\*" )

set(PNG\_INTEL\_SSE\_POSSIBLE\_VALUES on off)

set(PNG\_INTEL\_SSE "on" CACHE STRING "Enable INTEL\_SSE optimizations:

off: disable the optimizations")

set\_property(CACHE PNG\_INTEL\_SSE PROPERTY STRINGS

${PNG\_INTEL\_SSE\_POSSIBLE\_VALUES})

list(FIND PNG\_INTEL\_SSE\_POSSIBLE\_VALUES ${PNG\_INTEL\_SSE} index)

if(index EQUAL -1)

message(FATAL\_ERROR

" PNG\_INTEL\_SSE must be one of [${PNG\_INTEL\_SSE\_POSSIBLE\_VALUES}]")

elseif(NOT ${PNG\_INTEL\_SSE} STREQUAL "no")

set(libpng\_intel\_sources

intel/intel\_init.c

intel/filter\_sse2\_intrinsics.c)

if(${PNG\_INTEL\_SSE} STREQUAL "on")

add\_definitions(-DPNG\_INTEL\_SSE\_OPT=1)

endif()

else()

add\_definitions(-DPNG\_INTEL\_SSE\_OPT=0)

endif()

endif()

# set definitions and sources for MIPS

if(CMAKE\_SYSTEM\_PROCESSOR MATCHES "mipsel\*" OR

CMAKE\_SYSTEM\_PROCESSOR MATCHES "mips64el\*" )

set(PNG\_MIPS\_MSA\_POSSIBLE\_VALUES on off)

set(PNG\_MIPS\_MSA "on" CACHE STRING "Enable MIPS\_MSA optimizations:

off: disable the optimizations")

set\_property(CACHE PNG\_MIPS\_MSA PROPERTY STRINGS

${PNG\_MIPS\_MSA\_POSSIBLE\_VALUES})

list(FIND PNG\_MIPS\_MSA\_POSSIBLE\_VALUES ${PNG\_MIPS\_MSA} index)

if(index EQUAL -1)

message(FATAL\_ERROR

" PNG\_MIPS\_MSA must be one of [${PNG\_MIPS\_MSA\_POSSIBLE\_VALUES}]")

elseif(NOT ${PNG\_MIPS\_MSA} STREQUAL "no")

set(libpng\_mips\_sources

mips/mips\_init.c

mips/filter\_msa\_intrinsics.c)

if(${PNG\_MIPS\_MSA} STREQUAL "on")

add\_definitions(-DPNG\_MIPS\_MSA\_OPT=2)

endif()

else()

add\_definitions(-DPNG\_MIPS\_MSA\_OPT=0)

endif()

endif()

endif(PNG\_HARDWARE\_OPTIMIZATIONS)

# SET LIBNAME

set(PNG\_LIB\_NAME png${PNGLIB\_MAJOR}${PNGLIB\_MINOR})

# to distinguish between debug and release lib

set(CMAKE\_DEBUG\_POSTFIX "d")

include(CheckCSourceCompiles)

option(ld-version-script "Enable linker version script" ON)

if(ld-version-script AND NOT APPLE)

# Check if LD supports linker scripts.

file(WRITE "${CMAKE\_CURRENT\_BINARY\_DIR}/conftest.map" "VERS\_1 {

global: sym;

local: \*;

};

VERS\_2 {

global: sym2;

main;

} VERS\_1;

")

set(CMAKE\_REQUIRED\_FLAGS\_SAVE ${CMAKE\_REQUIRED\_FLAGS})

set(CMAKE\_REQUIRED\_FLAGS ${CMAKE\_REQUIRED\_FLAGS} "-Wl,--version-script='${CMAKE\_CURRENT\_BINARY\_DIR}/conftest.map'")

check\_c\_source\_compiles("void sym(void) {}

void sym2(void) {}

int main(void) {return 0;}

" HAVE\_LD\_VERSION\_SCRIPT)

if(NOT HAVE\_LD\_VERSION\_SCRIPT)

set(CMAKE\_REQUIRED\_FLAGS ${CMAKE\_REQUIRED\_FLAGS\_SAVE} "-Wl,-M -Wl,${CMAKE\_CURRENT\_BINARY\_DIR}/conftest.map")

check\_c\_source\_compiles("void sym(void) {}

void sym2(void) {}

int main(void) {return 0;}

" HAVE\_SOLARIS\_LD\_VERSION\_SCRIPT)

endif()

set(CMAKE\_REQUIRED\_FLAGS ${CMAKE\_REQUIRED\_FLAGS\_SAVE})

file(REMOVE "${CMAKE\_CURRENT\_BINARY\_DIR}/conftest.map")

endif()

# Find symbol prefix. Likely obsolete and unnecessary with recent

# toolchains (it's not done in many other projects).

function(symbol\_prefix)

set(SYMBOL\_PREFIX)

execute\_process(COMMAND "${CMAKE\_C\_COMPILER}" "-E" "-"

INPUT\_FILE /dev/null

OUTPUT\_VARIABLE OUT

RESULT\_VARIABLE STATUS)

if(CPP\_FAIL)

message(WARNING "Failed to run the C preprocessor")

endif()

string(REPLACE "\n" ";" OUT "${OUT}")

foreach(line ${OUT})

string(REGEX MATCH "^PREFIX=" found\_match "${line}")

if(found\_match)

STRING(REGEX REPLACE "^PREFIX=(.\*\)" "\\1" prefix "${line}")

string(REGEX MATCH "\_\_USER\_LABEL\_PREFIX\_\_" found\_match "${prefix}")

if(found\_match)

STRING(REGEX REPLACE "(.\*)\_\_USER\_LABEL\_PREFIX\_\_(.\*)" "\\1\\2" prefix "${prefix}")

endif()

set(SYMBOL\_PREFIX "${prefix}")

endif()

endforeach()

message(STATUS "Symbol prefix: ${SYMBOL\_PREFIX}")

set(SYMBOL\_PREFIX "${SYMBOL\_PREFIX}" PARENT\_SCOPE)

endfunction()

if(UNIX)

symbol\_prefix()

endif()

find\_program(AWK NAMES gawk awk)

include\_directories(${CMAKE\_CURRENT\_BINARY\_DIR})

if(NOT AWK OR ANDROID)

# No awk available to generate sources; use pre-built pnglibconf.h

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/pnglibconf.h.prebuilt

${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.h)

add\_custom\_target(genfiles) # Dummy

else()

include(CMakeParseArguments)

# Generate .chk from .out with awk

# generate\_chk(INPUT inputfile OUTPUT outputfile [DEPENDS dep1 [dep2...]])

function(generate\_chk)

set(options)

set(oneValueArgs INPUT OUTPUT)

set(multiValueArgs DEPENDS)

cmake\_parse\_arguments(\_GC "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN})

if (NOT \_GC\_INPUT)

message(FATAL\_ERROR "Invalid arguments. generate\_out requires input.")

endif()

if (NOT \_GC\_OUTPUT)

message(FATAL\_ERROR "Invalid arguments. generate\_out requires output.")

endif()

add\_custom\_command(OUTPUT "${\_GC\_OUTPUT}"

COMMAND "${CMAKE\_COMMAND}"

"-DINPUT=${\_GC\_INPUT}"

"-DOUTPUT=${\_GC\_OUTPUT}"

-P "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/genchk.cmake"

DEPENDS "${\_GC\_INPUT}" ${\_GC\_DEPENDS}

WORKING\_DIRECTORY "${CMAKE\_CURRENT\_BINARY\_DIR}")

endfunction()

# Generate .out from .c with awk

# generate\_out(INPUT inputfile OUTPUT outputfile [DEPENDS dep1 [dep2...]])

function(generate\_out)

set(options)

set(oneValueArgs INPUT OUTPUT)

set(multiValueArgs DEPENDS)

cmake\_parse\_arguments(\_GO "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN})

if (NOT \_GO\_INPUT)

message(FATAL\_ERROR "Invalid arguments. generate\_out requires input.")

endif()

if (NOT \_GO\_OUTPUT)

message(FATAL\_ERROR "Invalid arguments. generate\_out requires output.")

endif()

add\_custom\_command(OUTPUT "${\_GO\_OUTPUT}"

COMMAND "${CMAKE\_COMMAND}"

"-DINPUT=${\_GO\_INPUT}"

"-DOUTPUT=${\_GO\_OUTPUT}"

-P "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/genout.cmake"

DEPENDS "${\_GO\_INPUT}" ${\_GO\_DEPENDS}

WORKING\_DIRECTORY "${CMAKE\_CURRENT\_BINARY\_DIR}")

endfunction()

# Generate specific source file with awk

# generate\_source(OUTPUT outputfile [DEPENDS dep1 [dep2...]])

function(generate\_source)

set(options)

set(oneValueArgs OUTPUT)

set(multiValueArgs DEPENDS)

cmake\_parse\_arguments(\_GSO "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN})

if (NOT \_GSO\_OUTPUT)

message(FATAL\_ERROR "Invalid arguments. generate\_source requires output.")

endif()

add\_custom\_command(OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/${\_GSO\_OUTPUT}"

COMMAND "${CMAKE\_COMMAND}"

"-DOUTPUT=${\_GSO\_OUTPUT}"

-P "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/gensrc.cmake"

DEPENDS ${\_GSO\_DEPENDS}

WORKING\_DIRECTORY "${CMAKE\_CURRENT\_BINARY\_DIR}")

endfunction()

# Copy file

function(generate\_copy source destination)

add\_custom\_command(OUTPUT "${destination}"

COMMAND "${CMAKE\_COMMAND}" -E remove "${destination}"

COMMAND "${CMAKE\_COMMAND}" -E copy "${source}"

"${destination}"

DEPENDS "${source}")

endfunction()

# Generate scripts/pnglibconf.h

generate\_source(OUTPUT "scripts/pnglibconf.c"

DEPENDS "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/pnglibconf.dfa"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/options.awk"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/pngconf.h")

# Generate pnglibconf.c

generate\_source(OUTPUT "pnglibconf.c"

DEPENDS "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/pnglibconf.dfa"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/options.awk"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/pngconf.h")

if(PNG\_PREFIX)

set(PNGLIBCONF\_H\_EXTRA\_DEPENDS

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/prefix.out"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/macro.lst")

set(PNGPREFIX\_H\_EXTRA\_DEPENDS

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/intprefix.out")

endif()

generate\_out(INPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.c"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.out")

# Generate pnglibconf.h

generate\_source(OUTPUT "pnglibconf.h"

DEPENDS "${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.out"

${PNGLIBCONF\_H\_EXTRA\_DEPENDS})

generate\_out(INPUT "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/intprefix.c"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/intprefix.out"

DEPENDS "${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.h")

generate\_out(INPUT "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/prefix.c"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/prefix.out"

DEPENDS "${CMAKE\_CURRENT\_SOURCE\_DIR}/png.h"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/pngconf.h"

"${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.out")

# Generate pngprefix.h

generate\_source(OUTPUT "pngprefix.h"

DEPENDS ${PNGPREFIX\_H\_EXTRA\_DEPENDS})

generate\_out(INPUT "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/sym.c"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/sym.out"

DEPENDS "${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.h")

generate\_out(INPUT "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/symbols.c"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/symbols.out"

DEPENDS "${CMAKE\_CURRENT\_SOURCE\_DIR}/png.h"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/pngconf.h"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/pnglibconf.h.prebuilt")

generate\_out(INPUT "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/vers.c"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/vers.out"

DEPENDS "${CMAKE\_CURRENT\_SOURCE\_DIR}/png.h"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/pngconf.h"

"${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.h")

generate\_chk(INPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/symbols.out"

OUTPUT "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/symbols.chk"

DEPENDS "${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/checksym.awk"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/symbols.def")

add\_custom\_target(symbol-check DEPENDS

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/symbols.chk")

generate\_copy("${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/sym.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.sym")

generate\_copy("${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/vers.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.vers")

add\_custom\_target(genvers DEPENDS "${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.vers")

add\_custom\_target(gensym DEPENDS "${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.sym")

add\_custom\_target("genprebuilt"

COMMAND "${CMAKE\_COMMAND}"

"-DOUTPUT=scripts/pnglibconf.h.prebuilt"

-P "${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/gensrc.cmake"

WORKING\_DIRECTORY "${CMAKE\_CURRENT\_BINARY\_DIR}")

# A single target handles generation of all generated files. If

# they are dependend upon separately by multiple targets, this

# confuses parallel make (it would require a separate top-level

# target for each file to track the dependencies properly).

add\_custom\_target(genfiles DEPENDS

"${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.sym"

"${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.vers"

"${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.c"

"${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.h"

"${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/pngprefix.h"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/intprefix.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/pnglibconf.c"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/prefix.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/sym.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/symbols.chk"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/symbols.out"

"${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/vers.out")

endif(NOT AWK OR ANDROID)

# OUR SOURCES

set(libpng\_public\_hdrs

png.h

pngconf.h

"${CMAKE\_CURRENT\_BINARY\_DIR}/pnglibconf.h"

)

set(libpng\_private\_hdrs

pngpriv.h

pngdebug.h

pnginfo.h

pngstruct.h

)

if(AWK AND NOT ANDROID)

list(APPEND libpng\_private\_hdrs "${CMAKE\_CURRENT\_BINARY\_DIR}/pngprefix.h")

endif()

set(libpng\_sources

${libpng\_public\_hdrs}

${libpng\_private\_hdrs}

png.c

pngerror.c

pngget.c

pngmem.c

pngpread.c

pngread.c

pngrio.c

pngrtran.c

pngrutil.c

pngset.c

pngtrans.c

pngwio.c

pngwrite.c

pngwtran.c

pngwutil.c

${libpng\_arm\_sources}

${libpng\_intel\_sources}

${libpng\_mips\_sources}

${libpng\_powerpc\_sources}

)

set(pngtest\_sources

pngtest.c

)

set(pngvalid\_sources

contrib/libtests/pngvalid.c

)

set(pngstest\_sources

contrib/libtests/pngstest.c

)

set(pngunknown\_sources

contrib/libtests/pngunknown.c

)

set(pngimage\_sources

contrib/libtests/pngimage.c

)

set(pngfix\_sources

contrib/tools/pngfix.c

)

set(png\_fix\_itxt\_sources

contrib/tools/png-fix-itxt.c

)

if(MSVC)

add\_definitions(-D\_CRT\_SECURE\_NO\_DEPRECATE)

endif(MSVC)

if(PNG\_DEBUG)

add\_definitions(-DPNG\_DEBUG)

endif()

# NOW BUILD OUR TARGET

include\_directories(${CMAKE\_CURRENT\_SOURCE\_DIR} ${ZLIB\_INCLUDE\_DIR})

unset(PNG\_LIB\_TARGETS)

if(PNG\_SHARED)

add\_library(png SHARED ${libpng\_sources})

set(PNG\_LIB\_TARGETS png)

set\_target\_properties(png PROPERTIES OUTPUT\_NAME ${PNG\_LIB\_NAME})

add\_dependencies(png genfiles)

if(MSVC)

# msvc does not append 'lib' - do it here to have consistent name

set\_target\_properties(png PROPERTIES PREFIX "lib")

set\_target\_properties(png PROPERTIES IMPORT\_PREFIX "lib")

endif()

target\_link\_libraries(png ${ZLIB\_LIBRARY} ${M\_LIBRARY})

if(UNIX AND AWK)

if(HAVE\_LD\_VERSION\_SCRIPT)

set\_target\_properties(png PROPERTIES LINK\_FLAGS

"-Wl,--version-script='${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.vers'")

elseif(HAVE\_SOLARIS\_LD\_VERSION\_SCRIPT)

set\_target\_properties(png PROPERTIES LINK\_FLAGS

"-Wl,-M -Wl,'${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.vers'")

endif()

endif()

endif()

if(PNG\_STATIC)

# does not work without changing name

set(PNG\_LIB\_NAME\_STATIC png\_static)

add\_library(png\_static STATIC ${libpng\_sources})

add\_dependencies(png\_static genfiles)

# MSVC doesn't use a different file extension for shared vs. static

# libs. We are able to change OUTPUT\_NAME to remove the \_static

# for all other platforms.

if(NOT MSVC)

set\_target\_properties(png\_static PROPERTIES

OUTPUT\_NAME "${PNG\_LIB\_NAME}"

CLEAN\_DIRECT\_OUTPUT 1)

else()

set\_target\_properties(png\_static PROPERTIES

OUTPUT\_NAME "${PNG\_LIB\_NAME}\_static"

CLEAN\_DIRECT\_OUTPUT 1)

endif()

list(APPEND PNG\_LIB\_TARGETS png\_static)

if(MSVC)

# msvc does not append 'lib' - do it here to have consistent name

set\_target\_properties(png\_static PROPERTIES PREFIX "lib")

endif()

target\_link\_libraries(png\_static ${ZLIB\_LIBRARY} ${M\_LIBRARY})

endif()

if(PNG\_FRAMEWORK)

set(PNG\_LIB\_NAME\_FRAMEWORK png\_framework)

add\_library(png\_framework SHARED ${libpng\_sources})

add\_dependencies(png\_framework genfiles)

list(APPEND PNG\_LIB\_TARGETS png\_framework)

set\_target\_properties(png\_framework PROPERTIES

FRAMEWORK TRUE

FRAMEWORK\_VERSION ${PNGLIB\_VERSION}

MACOSX\_FRAMEWORK\_SHORT\_VERSION\_STRING ${PNGLIB\_MAJOR}.${PNGLIB\_MINOR}

MACOSX\_FRAMEWORK\_BUNDLE\_VERSION ${PNGLIB\_VERSION}

MACOSX\_FRAMEWORK\_IDENTIFIER org.libpng.libpng

XCODE\_ATTRIBUTE\_INSTALL\_PATH "@rpath"

PUBLIC\_HEADER "${libpng\_public\_hdrs}"

OUTPUT\_NAME png)

target\_link\_libraries(png\_framework ${ZLIB\_LIBRARY} ${M\_LIBRARY})

endif()

if(NOT PNG\_LIB\_TARGETS)

message(SEND\_ERROR

"No library variant selected to build. "

"Please enable at least one of the following options: "

" PNG\_STATIC, PNG\_SHARED, PNG\_FRAMEWORK")

endif()

if(PNG\_SHARED AND WIN32)

set\_target\_properties(png PROPERTIES DEFINE\_SYMBOL PNG\_BUILD\_DLL)

endif()

function(png\_add\_test)

set(options)

set(oneValueArgs NAME COMMAND)

set(multiValueArgs OPTIONS FILES)

cmake\_parse\_arguments(\_PAT "${options}" "${oneValueArgs}" "${multiValueArgs}" ${ARGN})

if (NOT \_PAT\_NAME)

message(FATAL\_ERROR "Invalid arguments. png\_add\_test requires name.")

endif()

if (NOT \_PAT\_COMMAND)

message(FATAL\_ERROR "Invalid arguments. png\_add\_test requires command.")

endif()

set(TEST\_OPTIONS "${\_PAT\_OPTIONS}")

set(TEST\_FILES "${\_PAT\_FILES}")

configure\_file("${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/test.cmake.in"

"${CMAKE\_CURRENT\_BINARY\_DIR}/tests/${\_PAT\_NAME}.cmake" @ONLY)

if(CMAKE\_MAJOR\_VERSION GREATER 2) # have generator expressions

add\_test(NAME "${\_PAT\_NAME}"

COMMAND "${CMAKE\_COMMAND}"

"-DLIBPNG=$<TARGET\_FILE:png>"

"-DTEST\_COMMAND=$<TARGET\_FILE:${\_PAT\_COMMAND}>"

-P "${CMAKE\_CURRENT\_BINARY\_DIR}/tests/${\_PAT\_NAME}.cmake")

else() # old 2.x add\_test; limited and won't work well on Windows

# Note LIBPNG is a dummy value as there are no generator expressions

add\_test("${\_PAT\_NAME}" "${CMAKE\_COMMAND}"

"-DLIBPNG=${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.so"

"-DTEST\_COMMAND=./${\_PAT\_COMMAND}"

-P "${CMAKE\_CURRENT\_BINARY\_DIR}/tests/${\_PAT\_NAME}.cmake")

endif()

endfunction()

if(PNG\_TESTS AND PNG\_SHARED)

# Find test PNG files by globbing, but sort lists to ensure

# consistency between different filesystems.

file(GLOB PNGSUITE\_PNGS "${CMAKE\_CURRENT\_SOURCE\_DIR}/contrib/pngsuite/\*.png")

list(SORT PNGSUITE\_PNGS)

file(GLOB TEST\_PNGS "${CMAKE\_CURRENT\_SOURCE\_DIR}/contrib/testpngs/\*.png")

list(SORT TEST\_PNGS)

set(PNGTEST\_PNG "${CMAKE\_CURRENT\_SOURCE\_DIR}/pngtest.png")

add\_executable(pngtest ${pngtest\_sources})

target\_link\_libraries(pngtest png)

png\_add\_test(NAME pngtest COMMAND pngtest FILES "${PNGTEST\_PNG}")

add\_executable(pngvalid ${pngvalid\_sources})

target\_link\_libraries(pngvalid png)

png\_add\_test(NAME pngvalid-gamma-16-to-8

COMMAND pngvalid OPTIONS --gamma-16-to-8)

png\_add\_test(NAME pngvalid-gamma-alpha-mode

COMMAND pngvalid OPTIONS --gamma-alpha-mode)

png\_add\_test(NAME pngvalid-gamma-background

COMMAND pngvalid OPTIONS --gamma-background)

png\_add\_test(NAME pngvalid-gamma-expand16-alpha-mode

COMMAND pngvalid OPTIONS --gamma-alpha-mode --expand16)

png\_add\_test(NAME pngvalid-gamma-expand16-background

COMMAND pngvalid OPTIONS --gamma-background --expand16)

png\_add\_test(NAME pngvalid-gamma-expand16-transform

COMMAND pngvalid OPTIONS --gamma-transform --expand16)

png\_add\_test(NAME pngvalid-gamma-sbit

COMMAND pngvalid OPTIONS --gamma-sbit)

png\_add\_test(NAME pngvalid-gamma-threshold

COMMAND pngvalid OPTIONS --gamma-threshold)

png\_add\_test(NAME pngvalid-gamma-transform

COMMAND pngvalid OPTIONS --gamma-transform)

png\_add\_test(NAME pngvalid-progressive-interlace-standard

COMMAND pngvalid OPTIONS --standard --progressive-read --interlace)

png\_add\_test(NAME pngvalid-progressive-size

COMMAND pngvalid OPTIONS --size --progressive-read)

png\_add\_test(NAME pngvalid-progressive-standard

COMMAND pngvalid OPTIONS --standard --progressive-read)

png\_add\_test(NAME pngvalid-standard

COMMAND pngvalid OPTIONS --standard)

png\_add\_test(NAME pngvalid-transform

COMMAND pngvalid OPTIONS --transform)

add\_executable(pngstest ${pngstest\_sources})

target\_link\_libraries(pngstest png)

foreach(gamma\_type 1.8 linear none sRGB)

foreach(alpha\_type none alpha)

set(PNGSTEST\_FILES)

foreach(test\_png ${TEST\_PNGS})

string(REGEX MATCH ".\*-linear[-.].\*" TEST\_PNG\_LINEAR "${test\_png}")

string(REGEX MATCH ".\*-sRGB[-.].\*" TEST\_PNG\_SRGB "${test\_png}")

string(REGEX MATCH ".\*-1.8[-.].\*" TEST\_PNG\_G18 "${test\_png}")

string(REGEX MATCH ".\*-alpha-.\*" TEST\_PNG\_ALPHA "${test\_png}")

set(TEST\_PNG\_VALID TRUE)

if(TEST\_PNG\_ALPHA)

if (NOT "${alpha\_type}" STREQUAL "alpha")

set(TEST\_PNG\_VALID FALSE)

endif()

else()

if ("${alpha\_type}" STREQUAL "alpha")

set(TEST\_PNG\_VALID FALSE)

endif()

endif()

if(TEST\_PNG\_LINEAR)

if(NOT "${gamma\_type}" STREQUAL "linear")

set(TEST\_PNG\_VALID FALSE)

endif()

elseif(TEST\_PNG\_SRGB)

if(NOT "${gamma\_type}" STREQUAL "sRGB")

set(TEST\_PNG\_VALID FALSE)

endif()

elseif(TEST\_PNG\_G18)

if(NOT "${gamma\_type}" STREQUAL "1.8")

set(TEST\_PNG\_VALID FALSE)

endif()

else()

if(NOT "${gamma\_type}" STREQUAL "none")

set(TEST\_PNG\_VALID FALSE)

endif()

endif()

if(TEST\_PNG\_VALID)

list(APPEND PNGSTEST\_FILES "${test\_png}")

endif()

endforeach()

# Should already be sorted, but sort anyway to be certain.

list(SORT PNGSTEST\_FILES)

png\_add\_test(NAME pngstest-${gamma\_type}-${alpha\_type}

COMMAND pngstest

OPTIONS --tmpfile "${gamma\_type}-${alpha\_type}-" --log

FILES ${PNGSTEST\_FILES})

endforeach()

endforeach()

add\_executable(pngunknown ${pngunknown\_sources})

target\_link\_libraries(pngunknown png)

png\_add\_test(NAME pngunknown-discard COMMAND pngunknown OPTIONS --strict default=discard FILES "${PNGTEST\_PNG}")

png\_add\_test(NAME pngunknown-IDAT COMMAND pngunknown OPTIONS --strict default=discard IDAT=save FILES "${PNGTEST\_PNG}")

png\_add\_test(NAME pngunknown-if-safe COMMAND pngunknown OPTIONS --strict default=if-safe FILES "${PNGTEST\_PNG}")

png\_add\_test(NAME pngunknown-sAPI COMMAND pngunknown OPTIONS --strict bKGD=save cHRM=save gAMA=save all=discard iCCP=save sBIT=save sRGB=save FILES "${PNGTEST\_PNG}")

png\_add\_test(NAME pngunknown-save COMMAND pngunknown OPTIONS --strict default=save FILES "${PNGTEST\_PNG}")

png\_add\_test(NAME pngunknown-sTER COMMAND pngunknown OPTIONS --strict sTER=if-safe FILES "${PNGTEST\_PNG}")

png\_add\_test(NAME pngunknown-vpAg COMMAND pngunknown OPTIONS --strict vpAg=if-safe FILES "${PNGTEST\_PNG}")

add\_executable(pngimage ${pngimage\_sources})

target\_link\_libraries(pngimage png)

png\_add\_test(NAME pngimage-quick COMMAND pngimage OPTIONS --list-combos --log FILES ${PNGSUITE\_PNGS})

png\_add\_test(NAME pngimage-full COMMAND pngimage OPTIONS --exhaustive --list-combos --log FILES ${PNGSUITE\_PNGS})

endif()

if(PNG\_SHARED)

add\_executable(pngfix ${pngfix\_sources})

target\_link\_libraries(pngfix png)

set(PNG\_BIN\_TARGETS pngfix)

add\_executable(png-fix-itxt ${png\_fix\_itxt\_sources})

target\_link\_libraries(png-fix-itxt ${ZLIB\_LIBRARY} ${M\_LIBRARY})

list(APPEND PNG\_BIN\_TARGETS png-fix-itxt)

endif()

# Set a variable with CMake code which:

# Creates a symlink from src to dest (if possible) or alternatively

# copies if different.

include(CMakeParseArguments)

function(CREATE\_SYMLINK DEST\_FILE)

cmake\_parse\_arguments(S "" "FILE;TARGET" "" ${ARGN})

if(NOT S\_TARGET AND NOT S\_FILE)

message(FATAL\_ERROR "Specify either a TARGET or a FILE for CREATE\_SYMLINK to link to.")

endif(NOT S\_TARGET AND NOT S\_FILE)

if(S\_TARGET AND S\_FILE)

message(FATAL\_ERROR "CREATE\_SYMLINK called with both source file ${S\_FILE} and build target ${S\_TARGET} arguments - can only handle 1 type per call.")

endif(S\_TARGET AND S\_FILE)

if(S\_FILE)

# If we don't need to symlink something that's coming from a build target,

# we can go ahead and symlink/copy at configure time.

if(CMAKE\_HOST\_WIN32 AND NOT CYGWIN AND NOT MSYS)

execute\_process(

COMMAND "${CMAKE\_COMMAND}" -E copy\_if\_different ${S\_FILE} ${DEST\_FILE}

WORKING\_DIRECTORY "${CMAKE\_CURRENT\_BINARY\_DIR}"

)

else(CMAKE\_HOST\_WIN32 AND NOT CYGWIN AND NOT MSYS)

execute\_process(

COMMAND ${CMAKE\_COMMAND} -E create\_symlink ${S\_FILE} ${DEST\_FILE}

WORKING\_DIRECTORY "${CMAKE\_CURRENT\_BINARY\_DIR}"

)

endif(CMAKE\_HOST\_WIN32 AND NOT CYGWIN AND NOT MSYS)

endif(S\_FILE)

if(S\_TARGET)

# We need to use generator expressions, which can be a bit tricky, so for

# simplicity make the symlink a POST\_BUILD step and use the TARGET

# signature of add\_custom\_command.

if(CMAKE\_HOST\_WIN32 AND NOT CYGWIN AND NOT MSYS)

add\_custom\_command(TARGET ${S\_TARGET} POST\_BUILD

COMMAND "${CMAKE\_COMMAND}" -E copy\_if\_different $<TARGET\_LINKER\_FILE\_NAME:${S\_TARGET}> $<TARGET\_LINKER\_FILE\_DIR:${S\_TARGET}>/${DEST\_FILE}

)

else(CMAKE\_HOST\_WIN32 AND NOT CYGWIN AND NOT MSYS)

add\_custom\_command(TARGET ${S\_TARGET} POST\_BUILD

COMMAND "${CMAKE\_COMMAND}" -E create\_symlink $<TARGET\_LINKER\_FILE\_NAME:${S\_TARGET}> $<TARGET\_LINKER\_FILE\_DIR:${S\_TARGET}>/${DEST\_FILE}

)

endif(CMAKE\_HOST\_WIN32 AND NOT CYGWIN AND NOT MSYS)

endif(S\_TARGET)

endfunction()

# Create source generation scripts.

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/genchk.cmake.in

${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/genchk.cmake @ONLY)

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/genout.cmake.in

${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/genout.cmake @ONLY)

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/scripts/gensrc.cmake.in

${CMAKE\_CURRENT\_BINARY\_DIR}/scripts/gensrc.cmake @ONLY)

# libpng is a library so default to 'lib'

if(NOT DEFINED CMAKE\_INSTALL\_LIBDIR)

set(CMAKE\_INSTALL\_LIBDIR lib)

endif(NOT DEFINED CMAKE\_INSTALL\_LIBDIR)

# CREATE PKGCONFIG FILES

# we use the same files like ./configure, so we have to set its vars

# Only do this on Windows for Cygwin - the files don't make much sense outside

# a UNIX look alike

if(NOT WIN32 OR CYGWIN OR MINGW)

set(prefix ${CMAKE\_INSTALL\_PREFIX})

set(exec\_prefix ${CMAKE\_INSTALL\_PREFIX})

set(libdir ${CMAKE\_INSTALL\_PREFIX}/${CMAKE\_INSTALL\_LIBDIR})

set(includedir ${CMAKE\_INSTALL\_PREFIX}/include)

set(LIBS "-lz -lm")

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/libpng.pc.in

${CMAKE\_CURRENT\_BINARY\_DIR}/${PNGLIB\_NAME}.pc @ONLY)

CREATE\_SYMLINK(libpng.pc FILE ${PNGLIB\_NAME}.pc)

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/libpng-config.in

${CMAKE\_CURRENT\_BINARY\_DIR}/${PNGLIB\_NAME}-config @ONLY)

CREATE\_SYMLINK(libpng-config FILE ${PNGLIB\_NAME}-config)

endif(NOT WIN32 OR CYGWIN OR MINGW)

# SET UP LINKS

if(PNG\_SHARED)

set\_target\_properties(png PROPERTIES

# VERSION 16.${PNGLIB\_RELEASE}.1.6.34

VERSION 16.${PNGLIB\_RELEASE}.0

SOVERSION 16

CLEAN\_DIRECT\_OUTPUT 1)

endif()

# If CMake > 2.4.x, we set a variable used below to export

# targets to an export file.

# TODO: Use VERSION\_GREATER after our cmake\_minimum\_required >= 2.6.2

if(CMAKE\_MAJOR\_VERSION GREATER 1 AND CMAKE\_MINOR\_VERSION GREATER 4)

set(PNG\_EXPORT\_RULE EXPORT libpng)

elseif(CMAKE\_MAJOR\_VERSION GREATER 2) # future proof

set(PNG\_EXPORT\_RULE EXPORT libpng)

endif()

# INSTALL

if(NOT SKIP\_INSTALL\_LIBRARIES AND NOT SKIP\_INSTALL\_ALL )

install(TARGETS ${PNG\_LIB\_TARGETS}

${PNG\_EXPORT\_RULE}

RUNTIME DESTINATION bin

LIBRARY DESTINATION ${CMAKE\_INSTALL\_LIBDIR}

ARCHIVE DESTINATION ${CMAKE\_INSTALL\_LIBDIR}

FRAMEWORK DESTINATION ${CMAKE\_INSTALL\_LIBDIR})

if(PNG\_SHARED)

# Create a symlink for libpng.dll.a => libpng16.dll.a on Cygwin

if(CYGWIN OR MINGW)

CREATE\_SYMLINK(libpng${CMAKE\_IMPORT\_LIBRARY\_SUFFIX} TARGET png)

install(FILES $<TARGET\_LINKER\_FILE\_DIR:png>/libpng${CMAKE\_IMPORT\_LIBRARY\_SUFFIX} DESTINATION ${CMAKE\_INSTALL\_LIBDIR})

endif(CYGWIN OR MINGW)

if(NOT WIN32)

CREATE\_SYMLINK(libpng${CMAKE\_SHARED\_LIBRARY\_SUFFIX} TARGET png)

install(FILES $<TARGET\_LINKER\_FILE\_DIR:png>/libpng${CMAKE\_SHARED\_LIBRARY\_SUFFIX} DESTINATION ${CMAKE\_INSTALL\_LIBDIR})

endif(NOT WIN32)

endif(PNG\_SHARED)

if(PNG\_STATIC)

if(NOT WIN32 OR CYGWIN OR MINGW)

CREATE\_SYMLINK( libpng${CMAKE\_STATIC\_LIBRARY\_SUFFIX} TARGET png\_static)

install(FILES $<TARGET\_LINKER\_FILE\_DIR:png\_static>/libpng${CMAKE\_STATIC\_LIBRARY\_SUFFIX} DESTINATION ${CMAKE\_INSTALL\_LIBDIR})

endif(NOT WIN32 OR CYGWIN OR MINGW)

endif()

endif()

if(NOT SKIP\_INSTALL\_HEADERS AND NOT SKIP\_INSTALL\_ALL )

install(FILES ${libpng\_public\_hdrs} DESTINATION include)

install(FILES ${libpng\_public\_hdrs} DESTINATION include/${PNGLIB\_NAME})

endif()

if(NOT SKIP\_INSTALL\_EXECUTABLES AND NOT SKIP\_INSTALL\_ALL )

if(NOT WIN32 OR CYGWIN OR MINGW)

install(PROGRAMS ${CMAKE\_CURRENT\_BINARY\_DIR}/libpng-config DESTINATION bin)

install(PROGRAMS ${CMAKE\_CURRENT\_BINARY\_DIR}/${PNGLIB\_NAME}-config

DESTINATION bin)

endif(NOT WIN32 OR CYGWIN OR MINGW)

endif()

if(NOT SKIP\_INSTALL\_PROGRAMS AND NOT SKIP\_INSTALL\_ALL )

install(TARGETS ${PNG\_BIN\_TARGETS}

RUNTIME DESTINATION bin)

endif()

if(NOT SKIP\_INSTALL\_FILES AND NOT SKIP\_INSTALL\_ALL )

# Install man pages

if(NOT PNG\_MAN\_DIR)

set(PNG\_MAN\_DIR "share/man")

endif()

install(FILES libpng.3 libpngpf.3 DESTINATION ${PNG\_MAN\_DIR}/man3)

install(FILES png.5 DESTINATION ${PNG\_MAN\_DIR}/man5)

# Install pkg-config files

if(NOT CMAKE\_HOST\_WIN32 OR CYGWIN OR MINGW)

install(FILES ${CMAKE\_CURRENT\_BINARY\_DIR}/libpng.pc

DESTINATION ${CMAKE\_INSTALL\_LIBDIR}/pkgconfig)

install(PROGRAMS ${CMAKE\_CURRENT\_BINARY\_DIR}/libpng-config

DESTINATION bin)

install(FILES ${CMAKE\_CURRENT\_BINARY\_DIR}/${PNGLIB\_NAME}.pc

DESTINATION ${CMAKE\_INSTALL\_LIBDIR}/pkgconfig)

install(PROGRAMS ${CMAKE\_CURRENT\_BINARY\_DIR}/${PNGLIB\_NAME}-config

DESTINATION bin)

endif(NOT CMAKE\_HOST\_WIN32 OR CYGWIN OR MINGW)

endif()

# On versions of CMake that support it, create an export file CMake

# users can include() to import our targets

if(PNG\_EXPORT\_RULE AND NOT SKIP\_INSTALL\_EXPORT AND NOT SKIP\_INSTALL\_ALL )

install(EXPORT libpng DESTINATION lib/libpng FILE lib${PNG\_LIB\_NAME}.cmake)

endif()

# what's with libpng-manual.txt and all the extra files?

# UNINSTALL

# do we need this?

# DIST

# do we need this?

# to create msvc import lib for mingw compiled shared lib

# pexports libpng.dll > libpng.def

# lib /def:libpng.def /machine:x86