# CMake build for libtiff

# Run "cmake" to generate the build files for your platform

#

# Copyright © 2015 Open Microscopy Environment / University of Dundee

# Written by Roger Leigh <rleigh@codelibre.net>

#

# Permission to use, copy, modify, distribute, and sell this software and

# its documentation for any purpose is hereby granted without fee, provided

# that (i) the above copyright notices and this permission notice appear in

# all copies of the software and related documentation, and (ii) the names of

# Sam Leffler and Silicon Graphics may not be used in any advertising or

# publicity relating to the software without the specific, prior written

# permission of Sam Leffler and Silicon Graphics.

#

# THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,

# EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY

# WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

#

# IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR

# ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND,

# OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,

# WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF

# LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE

# OF THIS SOFTWARE.

cmake\_minimum\_required(VERSION 2.8.9)

# Default policy is from 2.8.9

cmake\_policy(VERSION 2.8.9)

# 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)

# Read version information from configure.ac.

FILE(READ "${CMAKE\_CURRENT\_SOURCE\_DIR}/configure.ac" configure)

STRING(REGEX REPLACE ";" "\\\\;" configure "${configure}")

STRING(REGEX REPLACE "\n" ";" configure "${configure}")

foreach(line ${configure})

foreach(var LIBTIFF\_MAJOR\_VERSION LIBTIFF\_MINOR\_VERSION LIBTIFF\_MICRO\_VERSION LIBTIFF\_ALPHA\_VERSION

LIBTIFF\_CURRENT LIBTIFF\_REVISION LIBTIFF\_AGE)

if(NOT ${var})

string(REGEX MATCH "^${var}=(.\*)" ${var}\_MATCH "${line}")

if(${var}\_MATCH)

string(REGEX REPLACE "^${var}=(.\*)" "\\1" ${var} "${line}")

endif()

endif()

endforeach()

endforeach()

math(EXPR SO\_MAJOR "${LIBTIFF\_CURRENT} - ${LIBTIFF\_AGE}")

set(SO\_MINOR "${LIBTIFF\_AGE}")

set(SO\_REVISION "${LIBTIFF\_REVISION}")

message(STATUS "Building tiff version ${LIBTIFF\_MAJOR\_VERSION}.${LIBTIFF\_MINOR\_VERSION}.${LIBTIFF\_MICRO\_VERSION}${LIBTIFF\_ALPHA\_VERSION}")

message(STATUS "libtiff library version ${SO\_MAJOR}.${SO\_MINOR}.${SO\_REVISION}")

set(BUILD\_SHARED\_LIBS ON CACHE BOOL "Build shared libraries")

# Project version

project(tiff C)

set(VERSION "${LIBTIFF\_MAJOR\_VERSION}.${LIBTIFF\_MINOR\_VERSION}.${LIBTIFF\_MICRO\_VERSION}")

set(tiff\_VERSION "${VERSION}")

set(tiff\_VERSION\_MAJOR "${LIBTIFF\_MAJOR\_VERSION}")

set(tiff\_VERSION\_MINOR "${LIBTIFF\_MINOR\_VERSION}")

set(tiff\_VERSION\_PATCH "${LIBTIFF\_MICRO\_VERSION}")

# the other tiff\_VERSION\_\* variables are set automatically

set(tiff\_VERSION\_ALPHA "${LIBTIFF\_ALPHA\_VERSION}")

# Library version (unlike libtool's baroque scheme, WYSIWYG here)

set(SO\_COMPATVERSION "${SO\_MAJOR}")

set(SO\_VERSION "${SO\_MAJOR}.${SO\_MINOR}.${SO\_REVISION}")

# For autotools header compatibility

set(PACKAGE\_NAME "LibTIFF Software")

set(PACKAGE\_TARNAME "${PROJECT\_NAME}")

set(PACKAGE\_VERSION "${PROJECT\_VERSION}${tiff\_VERSION\_ALPHA}")

set(PACKAGE\_STRING "${PACKAGE\_NAME} ${PACKAGE\_VERSION}")

set(PACKAGE\_BUGREPORT "tiff@lists.maptools.org")

include(GNUInstallDirs)

include(CheckCCompilerFlag)

include(CheckCSourceCompiles)

include(CheckIncludeFile)

include(CheckTypeSize)

include(CheckFunctionExists)

enable\_testing()

macro(current\_date var)

if(UNIX)

execute\_process(COMMAND "date" +"%Y%m%d" OUTPUT\_VARIABLE ${var})

endif()

endmacro()

current\_date(RELEASE\_DATE)

macro(extra\_dist)

foreach(file ${ARGV})

file(RELATIVE\_PATH relfile "${PROJECT\_SOURCE\_DIR}"

"${CMAKE\_CURRENT\_SOURCE\_DIR}/${file}")

list(APPEND EXTRA\_DIST "${relfile}")

endforeach()

set(EXTRA\_DIST "${EXTRA\_DIST}" PARENT\_SCOPE)

endmacro()

set(EXTRA\_DIST

HOWTO-RELEASE

Makefile.vc

SConstruct

autogen.sh

configure.com

nmake.opt

libtiff-4.pc.in)

# These are annoyingly verbose, produce false positives or don't work

# nicely with all supported compiler versions, so are disabled unless

# explicitly enabled.

option(extra-warnings "Enable extra compiler warnings" OFF)

# This will cause the compiler to fail when an error occurs.

option(fatal-warnings "Compiler warnings are errors" OFF)

# Check if the compiler supports each of the following additional

# flags, and enable them if supported. This greatly improves the

# quality of the build by checking for a number of common problems,

# some of which are quite serious.

if(CMAKE\_C\_COMPILER\_ID STREQUAL "GNU" OR

CMAKE\_C\_COMPILER\_ID MATCHES "Clang")

set(test\_flags

-Wall

-Winline

-W

-Wformat-security

-Wpointer-arith

-Wdisabled-optimization

-Wno-unknown-pragmas

-Wdeclaration-after-statement

-fstrict-aliasing)

if(extra-warnings)

list(APPEND test\_flags

-Wfloat-equal

-Wmissing-prototypes

-Wunreachable-code)

endif()

if(fatal-warnings)

list(APPEND test\_flags

-Werror)

endif()

elseif(CMAKE\_C\_COMPILER\_ID STREQUAL "MSVC")

set(test\_flags)

if(extra-warnings)

list(APPEND test\_flags

/W4)

else()

list(APPEND test\_flags

/W3)

endif()

if (fatal-warnings)

list(APPEND test\_flags

/WX)

endif()

endif()

foreach(flag ${test\_flags})

string(REGEX REPLACE "[^A-Za-z0-9]" "\_" flag\_var "${flag}")

set(test\_c\_flag "C\_FLAG${flag\_var}")

CHECK\_C\_COMPILER\_FLAG(${flag} "${test\_c\_flag}")

if (${test\_c\_flag})

set(CMAKE\_C\_FLAGS "${CMAKE\_C\_FLAGS} ${flag}")

endif (${test\_c\_flag})

endforeach(flag ${test\_flags})

if(MSVC)

set(CMAKE\_DEBUG\_POSTFIX "d")

endif()

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

# Check if LD supports linker scripts.

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

global: sym;

};

VERS\_2 {

global: sym;

} 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("int main(void){return 0;}" HAVE\_LD\_VERSION\_SCRIPT)

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

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

if (ld-version-script AND HAVE\_LD\_VERSION\_SCRIPT)

set(HAVE\_LD\_VERSION\_SCRIPT TRUE)

else()

set(HAVE\_LD\_VERSION\_SCRIPT FALSE)

endif()

# Find libm, if available

find\_library(M\_LIBRARY m)

check\_include\_file(assert.h HAVE\_ASSERT\_H)

check\_include\_file(dlfcn.h HAVE\_DLFCN\_H)

check\_include\_file(fcntl.h HAVE\_FCNTL\_H)

check\_include\_file(inttypes.h HAVE\_INTTYPES\_H)

check\_include\_file(io.h HAVE\_IO\_H)

check\_include\_file(limits.h HAVE\_LIMITS\_H)

check\_include\_file(malloc.h HAVE\_MALLOC\_H)

check\_include\_file(memory.h HAVE\_MEMORY\_H)

check\_include\_file(search.h HAVE\_SEARCH\_H)

check\_include\_file(stdint.h HAVE\_STDINT\_H)

check\_include\_file(string.h HAVE\_STRING\_H)

check\_include\_file(strings.h HAVE\_STRINGS\_H)

check\_include\_file(sys/time.h HAVE\_SYS\_TIME\_H)

check\_include\_file(sys/types.h HAVE\_SYS\_TYPES\_H)

check\_include\_file(unistd.h HAVE\_UNISTD\_H)

# Inspired from /usr/share/autoconf/autoconf/c.m4

foreach(inline\_keyword "inline" "\_\_inline\_\_" "\_\_inline")

if(NOT DEFINED C\_INLINE)

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

set(CMAKE\_REQUIRED\_DEFINITIONS ${CMAKE\_REQUIRED\_DEFINITIONS}

"-Dinline=${inline\_keyword}")

check\_c\_source\_compiles("

typedef int foo\_t;

static inline foo\_t static\_foo() {return 0;}

foo\_t foo(){return 0;}

int main(int argc, char \*argv[]) {return 0;}"

C\_HAS\_${inline\_keyword})

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

if(C\_HAS\_${inline\_keyword})

set(C\_INLINE TRUE)

set(INLINE\_KEYWORD "${inline\_keyword}")

endif()

endif()

endforeach()

if(NOT DEFINED C\_INLINE)

set(INLINE\_KEYWORD)

endif()

# off\_t and size\_t checks omitted; not clear they are used at all

# Are off\_t and size\_t checks strictly necessary?

# Check if sys/time.h and time.h allow use together

check\_c\_source\_compiles("

#include <sys/time.h>

#include <time.h>

int main(void){return 0;}"

TIME\_WITH\_SYS\_TIME)

# Check if struct tm is in sys/time.h

check\_c\_source\_compiles("

#include <sys/types.h>

#include <time.h>

int main(void){

struct tm tm;

int \*p = &tm.tm\_sec;

return !p;

}"

TM\_IN\_SYS\_TIME)

# Check type sizes

# NOTE: Could be replaced with C99 <stdint.h>

check\_type\_size("signed short" SIZEOF\_SIGNED\_SHORT)

check\_type\_size("unsigned short" SIZEOF\_UNSIGNED\_SHORT)

check\_type\_size("signed int" SIZEOF\_SIGNED\_INT)

check\_type\_size("unsigned int" SIZEOF\_UNSIGNED\_INT)

check\_type\_size("signed long" SIZEOF\_SIGNED\_LONG)

check\_type\_size("unsigned long" SIZEOF\_UNSIGNED\_LONG)

check\_type\_size("signed long long" SIZEOF\_SIGNED\_LONG\_LONG)

check\_type\_size("unsigned long long" SIZEOF\_UNSIGNED\_LONG\_LONG)

check\_type\_size("unsigned char \*" SIZEOF\_UNSIGNED\_CHAR\_P)

set(CMAKE\_EXTRA\_INCLUDE\_FILES\_SAVE ${CMAKE\_EXTRA\_INCLUDE\_FILES})

set(CMAKE\_EXTRA\_INCLUDE\_FILES ${CMAKE\_EXTRA\_INCLUDE\_FILES} "stddef.h")

check\_type\_size("size\_t" SIZEOF\_SIZE\_T)

check\_type\_size("ptrdiff\_t" SIZEOF\_PTRDIFF\_T)

set(CMAKE\_EXTRA\_INCLUDE\_FILES ${CMAKE\_EXTRA\_INCLUDE\_FILES\_SAVE})

macro(report\_values)

foreach(val ${ARGV})

message(STATUS "${val} set to ${${val}}")

endforeach()

endmacro()

set(TIFF\_INT8\_T "signed char")

set(TIFF\_UINT8\_T "unsigned char")

set(TIFF\_INT16\_T "signed short")

set(TIFF\_UINT16\_T "unsigned short")

if(SIZEOF\_SIGNED\_INT EQUAL 4)

set(TIFF\_INT32\_T "signed int")

set(TIFF\_INT32\_FORMAT "%d")

elseif(SIZEOF\_SIGNED\_LONG EQUAL 4)

set(TIFF\_INT32\_T "signed long")

set(TIFF\_INT32\_FORMAT "%ld")

endif()

if(SIZEOF\_UNSIGNED\_INT EQUAL 4)

set(TIFF\_UINT32\_T "unsigned int")

set(TIFF\_UINT32\_FORMAT "%u")

elseif(SIZEOF\_UNSIGNED\_LONG EQUAL 4)

set(TIFF\_UINT32\_T "unsigned long")

set(TIFF\_UINT32\_FORMAT "%lu")

endif()

if(SIZEOF\_SIGNED\_LONG EQUAL 8)

set(TIFF\_INT64\_T "signed long")

set(TIFF\_INT64\_FORMAT "%ld")

elseif(SIZEOF\_SIGNED\_LONG\_LONG EQUAL 8)

set(TIFF\_INT64\_T "signed long long")

if (MINGW)

set(TIFF\_INT64\_FORMAT "%I64d")

else()

set(TIFF\_INT64\_FORMAT "%lld")

endif()

endif()

if(SIZEOF\_UNSIGNED\_LONG EQUAL 8)

set(TIFF\_UINT64\_T "unsigned long")

set(TIFF\_UINT64\_FORMAT "%lu")

elseif(SIZEOF\_UNSIGNED\_LONG\_LONG EQUAL 8)

set(TIFF\_UINT64\_T "unsigned long long")

if (MINGW)

set(TIFF\_UINT64\_FORMAT "%I64u")

else()

set(TIFF\_UINT64\_FORMAT "%llu")

endif()

endif()

if(SIZEOF\_UNSIGNED\_INT EQUAL SIZEOF\_SIZE\_T)

set(TIFF\_SIZE\_T "unsigned int")

set(TIFF\_SIZE\_FORMAT "%u")

elseif(SIZEOF\_UNSIGNED\_LONG EQUAL SIZEOF\_SIZE\_T)

set(TIFF\_SIZE\_T "unsigned long")

set(TIFF\_SIZE\_FORMAT "%lu")

elseif(SIZEOF\_UNSIGNED\_LONG\_LONG EQUAL SIZEOF\_SIZE\_T)

set(TIFF\_SIZE\_T "unsigned long")

if (MINGW)

set(TIFF\_SIZE\_FORMAT "%I64u")

else()

set(TIFF\_SIZE\_FORMAT "%llu")

endif()

endif()

if(SIZEOF\_SIGNED\_INT EQUAL SIZEOF\_UNSIGNED\_CHAR\_P)

set(TIFF\_SSIZE\_T "signed int")

set(TIFF\_SSIZE\_FORMAT "%d")

elseif(SIZEOF\_SIGNED\_LONG EQUAL SIZEOF\_UNSIGNED\_CHAR\_P)

set(TIFF\_SSIZE\_T "signed long")

set(TIFF\_SSIZE\_FORMAT "%ld")

elseif(SIZEOF\_SIGNED\_LONG\_LONG EQUAL SIZEOF\_UNSIGNED\_CHAR\_P)

set(TIFF\_SSIZE\_T "signed long long")

if (MINGW)

set(TIFF\_SSIZE\_FORMAT "%I64d")

else()

set(TIFF\_SSIZE\_FORMAT "%lld")

endif()

endif()

if(NOT SIZEOF\_PTRDIFF\_T)

set(TIFF\_PTRDIFF\_T "${TIFF\_SSIZE\_T}")

set(TIFF\_PTRDIFF\_FORMAT "${SSIZE\_FORMAT}")

else()

set(TIFF\_PTRDIFF\_T "ptrdiff\_t")

set(TIFF\_PTRDIFF\_FORMAT "%ld")

endif()

#report\_values(TIFF\_INT8\_T TIFF\_INT8\_FORMAT

# TIFF\_UINT8\_T TIFF\_UINT8\_FORMAT

# TIFF\_INT16\_T TIFF\_INT16\_FORMAT

# TIFF\_UINT16\_T TIFF\_UINT16\_FORMAT

# TIFF\_INT32\_T TIFF\_INT32\_FORMAT

# TIFF\_UINT32\_T TIFF\_UINT32\_FORMAT

# TIFF\_INT64\_T TIFF\_INT64\_FORMAT

# TIFF\_UINT64\_T TIFF\_UINT64\_FORMAT

# TIFF\_SSIZE\_T TIFF\_SSIZE\_FORMAT

# TIFF\_PTRDIFF\_T TIFF\_PTRDIFF\_FORMAT)

# Nonstandard int types

check\_type\_size(INT8 int8)

set(HAVE\_INT8 ${INT8})

check\_type\_size(INT16 int16)

set(HAVE\_INT16 ${INT16})

check\_type\_size(INT32 int32)

set(HAVE\_INT32 ${INT32})

# Check functions

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

set(CMAKE\_REQUIRED\_LIBRARIES ${CMAKE\_REQUIRED\_LIBRARIES} ${M\_LIBRARY})

check\_function\_exists(floor HAVE\_FLOOR)

check\_function\_exists(pow HAVE\_POW)

check\_function\_exists(sqrt HAVE\_SQRT)

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

check\_function\_exists(isascii HAVE\_ISASCII)

check\_function\_exists(memmove HAVE\_MEMMOVE)

check\_function\_exists(memset HAVE\_MEMSET)

check\_function\_exists(mmap HAVE\_MMAP)

check\_function\_exists(setmode HAVE\_SETMODE)

check\_function\_exists(strcasecmp HAVE\_STRCASECMP)

check\_function\_exists(strchr HAVE\_STRCHR)

check\_function\_exists(strrchr HAVE\_STRRCHR)

check\_function\_exists(strstr HAVE\_STRSTR)

check\_function\_exists(strtol HAVE\_STRTOL)

check\_function\_exists(strtol HAVE\_STRTOUL)

check\_function\_exists(strtoull HAVE\_STRTOULL)

check\_function\_exists(getopt HAVE\_GETOPT)

check\_function\_exists(lfind HAVE\_LFIND)

# May be inlined, so check it compiles:

check\_c\_source\_compiles("

#include <stdio.h>

int main(void) {

char buf[10];

snprintf(buf, 10, \"Test %d\", 1);

return 0;

}"

HAVE\_SNPRINTF)

if(NOT HAVE\_SNPRINTF)

add\_definitions(-DNEED\_LIBPORT)

endif()

# CPU bit order

set(fillorder FILLORDER\_MSB2LSB)

if(CMAKE\_HOST\_SYSTEM\_PROCESSOR MATCHES "i.\*86.\*" OR

CMAKE\_HOST\_SYSTEM\_PROCESSOR MATCHES "amd64.\*" OR

CMAKE\_HOST\_SYSTEM\_PROCESSOR MATCHES "x86\_64.\*")

set(fillorder FILLORDER\_LSB2MSB)

endif()

set(HOST\_FILLORDER ${fillorder} CACHE STRING "Native CPU bit order")

mark\_as\_advanced(HOST\_FILLORDER)

# CPU endianness

include(TestBigEndian)

test\_big\_endian(bigendian)

if (bigendian)

set(bigendian ON)

else()

set(bigendian OFF)

endif()

set(HOST\_BIG\_ENDIAN ${bigendian} CACHE STRING "Native CPU bit order")

mark\_as\_advanced(HOST\_BIG\_ENDIAN)

if (HOST\_BIG\_ENDIAN)

set(HOST\_BIG\_ENDIAN 1)

else()

set(HOST\_BIG\_ENDIAN 0)

endif()

# IEEE floating point

set(HAVE\_IEEEFP 1 CACHE STRING "IEEE floating point is available")

mark\_as\_advanced(HAVE\_IEEEFP)

report\_values(CMAKE\_HOST\_SYSTEM\_PROCESSOR HOST\_FILLORDER

HOST\_BIG\_ENDIAN HAVE\_IEEEFP)

# Large file support

if (UNIX OR MINGW)

# This might not catch every possibility catered for by

# AC\_SYS\_LARGEFILE.

add\_definitions(-D\_FILE\_OFFSET\_BITS=64)

set(FILE\_OFFSET\_BITS 64)

endif()

# Documentation install directory (default to cmake project docdir)

set(LIBTIFF\_DOCDIR "${CMAKE\_INSTALL\_FULL\_DOCDIR}")

# Options to enable and disable internal codecs

option(ccitt "support for CCITT Group 3 & 4 algorithms" ON)

set(CCITT\_SUPPORT ${ccitt})

option(packbits "support for Macintosh PackBits algorithm" ON)

set(PACKBITS\_SUPPORT ${packbits})

option(lzw "support for LZW algorithm" ON)

set(LZW\_SUPPORT ${lzw})

option(thunder "support for ThunderScan 4-bit RLE algorithm" ON)

set(THUNDER\_SUPPORT ${thunder})

option(next "support for NeXT 2-bit RLE algorithm" ON)

set(NEXT\_SUPPORT ${next})

option(logluv "support for LogLuv high dynamic range algorithm" ON)

set(LOGLUV\_SUPPORT ${logluv})

# Option for Microsoft Document Imaging

option(mdi "support for Microsoft Document Imaging" ON)

set(MDI\_SUPPORT ${mdi})

# ZLIB

option(zlib "use zlib (required for Deflate compression)" ON)

if (zlib)

find\_package(ZLIB)

endif()

set(ZLIB\_SUPPORT 0)

if(ZLIB\_FOUND)

set(ZLIB\_SUPPORT 1)

endif()

set(ZIP\_SUPPORT ${ZLIB\_SUPPORT})

# Option for Pixar log-format algorithm

# Pixar log format

option(pixarlog "support for Pixar log-format algorithm (requires Zlib)" ON)

set(PIXARLOG\_SUPPORT FALSE)

if (ZLIB\_SUPPORT)

if(pixarlog)

set(PIXARLOG\_SUPPORT TRUE)

endif()

endif()

# JPEG

option(jpeg "use libjpeg (required for JPEG compression)" ON)

if (jpeg)

find\_package(JPEG)

endif()

set(JPEG\_SUPPORT FALSE)

if(JPEG\_FOUND)

set(JPEG\_SUPPORT TRUE)

endif()

option(old-jpeg "support for Old JPEG compression (read-only)" ON)

set(OJPEG\_SUPPORT FALSE)

if (JPEG\_SUPPORT)

if (old-jpeg)

set(OJPEG\_SUPPORT TRUE)

endif()

endif()

# JBIG-KIT

option(jbig "use ISO JBIG compression (requires JBIT-KIT library)" ON)

if (jbig)

set(JBIG\_FOUND 0)

find\_path(JBIG\_INCLUDE\_DIR jbig.h)

set(JBIG\_NAMES ${JBIG\_NAMES} jbig libjbig)

find\_library(JBIG\_LIBRARY NAMES ${JBIG\_NAMES})

if (JBIG\_INCLUDE\_DIR AND JBIG\_LIBRARY)

set(JBIG\_FOUND 1)

set(JBIG\_LIBRARIES ${JBIG\_LIBRARY})

endif()

endif()

set(JBIG\_SUPPORT 0)

if(JBIG\_FOUND)

set(JBIG\_FOUND TRUE)

set(JBIG\_SUPPORT 1)

else()

set(JBIG\_FOUND FALSE)

endif()

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

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

set(CMAKE\_REQUIRED\_INCLUDES ${CMAKE\_REQUIRED\_INCLUDES} ${JBIG\_INCLUDE\_DIR})

set(CMAKE\_REQUIRED\_LIBRARIES ${CMAKE\_REQUIRED\_LIBRARIES} ${JBIG\_LIBRARY})

check\_function\_exists(jbg\_newlen HAVE\_JBG\_NEWLEN)

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

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

# liblzma2

option(lzma "use liblzma (required for LZMA2 compression)" ON)

if (lzma)

find\_package(LibLZMA)

endif()

set(LZMA\_SUPPORT 0)

if(LIBLZMA\_FOUND)

set(LZMA\_SUPPORT 1)

endif()

# 8/12-bit jpeg mode

option(jpeg12 "enable libjpeg 8/12-bit dual mode (requires separate

12-bit libjpeg build)" ON)

set(JPEG12\_INCLUDE\_DIR JPEG12\_INCLUDE\_DIR-NOTFOUND CACHE PATH "Include directory for 12-bit libjpeg")

set(JPEG12\_LIBRARY JPEG12\_LIBRARY-NOTFOUND CACHE FILEPATH "12-bit libjpeg library")

set(JPEG12\_FOUND FALSE)

if (JPEG12\_INCLUDE\_DIR AND JPEG12\_LIBRARY)

set(JPEG12\_LIBRARIES ${JPEG12\_LIBRARY})

set(JPEG12\_FOUND TRUE)

endif()

if (JPEG12\_FOUND)

set(JPEG\_DUAL\_MODE\_8\_12 1)

set(LIBJPEG\_12\_PATH "${JPEG12\_INCLUDE\_DIR}/jpeglib.h")

endif()

# C++ support

option(cxx "Enable C++ stream API building (requires C++ compiler)" ON)

set(CXX\_SUPPORT FALSE)

if (cxx)

enable\_language(CXX)

set(CXX\_SUPPORT TRUE)

endif()

# OpenGL and GLUT

find\_package(OpenGL)

find\_package(GLUT)

set(HAVE\_OPENGL FALSE)

if(OPENGL\_FOUND AND OPENGL\_GLU\_FOUND AND GLUT\_FOUND)

set(HAVE\_OPENGL TRUE)

endif()

# Purely to satisfy the generated headers:

check\_include\_file(GL/gl.h HAVE\_GL\_GL\_H)

check\_include\_file(GL/glu.h HAVE\_GL\_GLU\_H)

check\_include\_file(GL/glut.h HAVE\_GL\_GLUT\_H)

check\_include\_file(GLUT/glut.h HAVE\_GLUT\_GLUT\_H)

check\_include\_file(OpenGL/gl.h HAVE\_OPENGL\_GL\_H)

check\_include\_file(OpenGL/glu.h HAVE\_OPENGL\_GLU\_H)

# Win32 IO

set(win32\_io FALSE)

if(WIN32)

set(win32\_io TRUE)

endif()

set(USE\_WIN32\_FILEIO ${win32\_io} CACHE BOOL "Use win32 IO system (Microsoft Windows only)")

if (USE\_WIN32\_FILEIO)

set(USE\_WIN32\_FILEIO TRUE)

else()

set(USE\_WIN32\_FILEIO FALSE)

endif()

# Orthogonal features

# Strip chopping

option(strip-chopping "strip chopping (whether or not to convert single-strip uncompressed images to mutiple strips of specified size to reduce memory usage)" ON)

set(TIFF\_DEFAULT\_STRIP\_SIZE 8192 CACHE STRING "default size of the strip in bytes (when strip chopping is enabled)")

set(STRIPCHOP\_DEFAULT)

if(strip-chopping)

set(STRIPCHOP\_DEFAULT TRUE)

if(TIFF\_DEFAULT\_STRIP\_SIZE)

set(STRIP\_SIZE\_DEFAULT "${TIFF\_DEFAULT\_STRIP\_SIZE}")

endif()

endif()

# Defer loading of strip/tile offsets

option(defer-strile-load "enable deferred strip/tile offset/size loading (experimental)" OFF)

set(DEFER\_STRILE\_LOAD ${defer-strile-load})

# CHUNKY\_STRIP\_READ\_SUPPORT

option(chunky-strip-read "enable reading large strips in chunks for TIFFReadScanline() (experimental)" OFF)

set(CHUNKY\_STRIP\_READ\_SUPPORT ${chunky-strip-read})

# SUBIFD support

set(SUBIFD\_SUPPORT 1)

# Default handling of ASSOCALPHA support.

option(extrasample-as-alpha "the RGBA interface will treat a fourth sample with no EXTRASAMPLE\_ value as being ASSOCALPHA. Many packages produce RGBA files but don't mark the alpha properly" ON)

if(extrasample-as-alpha)

set(DEFAULT\_EXTRASAMPLE\_AS\_ALPHA 1)

endif()

# Default handling of YCbCr subsampling support.

# See Bug 168 in Bugzilla, and JPEGFixupTestSubsampling() for details.

option(check-ycbcr-subsampling "enable picking up YCbCr subsampling info from the JPEG data stream to support files lacking the tag" ON)

if (check-ycbcr-subsampling)

set(CHECK\_JPEG\_YCBCR\_SUBSAMPLING 1)

endif()

# Generate pkg-config file

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

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

set(libdir "${CMAKE\_INSTALL\_FULL\_LIBDIR}")

set(includedir "${CMAKE\_INSTALL\_FULL\_INCLUDEDIR}")

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/libtiff-4.pc.in

${CMAKE\_CURRENT\_BINARY\_DIR}/libtiff-4.pc)

install(FILES ${CMAKE\_CURRENT\_BINARY\_DIR}/libtiff-4.pc

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

# Includes used by libtiff (and tests)

if(ZLIB\_INCLUDE\_DIRS)

list(APPEND TIFF\_INCLUDES ${ZLIB\_INCLUDE\_DIRS})

endif()

if(JPEG\_INCLUDE\_DIR)

list(APPEND TIFF\_INCLUDES ${JPEG\_INCLUDE\_DIR})

endif()

if(JPEG12\_INCLUDE\_DIR)

list(APPEND TIFF\_INCLUDES ${JPEG12\_INCLUDE\_DIR})

endif()

if(JBIG\_INCLUDE\_DIR)

list(APPEND TIFF\_INCLUDES ${JBIG\_INCLUDE\_DIR})

endif()

if(LIBLZMA\_INCLUDE\_DIRS)

list(APPEND TIFF\_INCLUDES ${LIBLZMA\_INCLUDE\_DIRS})

endif()

# Libraries required by libtiff

set(TIFF\_LIBRARY\_DEPS)

if(M\_LIBRARY)

list(APPEND TIFF\_LIBRARY\_DEPS ${M\_LIBRARY})

endif()

if(ZLIB\_LIBRARIES)

list(APPEND TIFF\_LIBRARY\_DEPS ${ZLIB\_LIBRARIES})

endif()

if(JPEG\_LIBRARIES)

list(APPEND TIFF\_LIBRARY\_DEPS ${JPEG\_LIBRARIES})

endif()

if(JPEG12\_LIBRARIES)

list(APPEND TIFF\_LIBRARY\_DEPS ${JPEG12\_LIBRARIES})

endif()

if(JBIG\_LIBRARIES)

list(APPEND TIFF\_LIBRARY\_DEPS ${JBIG\_LIBRARIES})

endif()

if(LIBLZMA\_LIBRARIES)

list(APPEND TIFF\_LIBRARY\_DEPS ${LIBLZMA\_LIBRARIES})

endif()

#report\_values(TIFF\_INCLUDES TIFF\_LIBRARY\_DEPS)

# Process subdirectories

add\_subdirectory(port)

add\_subdirectory(libtiff)

add\_subdirectory(tools)

add\_subdirectory(test)

add\_subdirectory(contrib)

add\_subdirectory(build)

add\_subdirectory(man)

add\_subdirectory(html)

#message(STATUS "EXTRA\_DIST: ${EXTRA\_DIST}")

message(STATUS "")

message(STATUS "Libtiff is now configured for ${host}")

message(STATUS "")

message(STATUS " Installation directory: ${prefix}")

message(STATUS " Documentation directory: ${LIBTIFF\_DOCDIR}")

message(STATUS " C compiler: ${CMAKE\_C\_COMPILER}")

message(STATUS " C++ compiler: ${CMAKE\_CXX\_COMPILER}")

message(STATUS " Build shared libraries: ${BUILD\_SHARED\_LIBS}")

message(STATUS " Enable linker symbol versioning: ${HAVE\_LD\_VERSION\_SCRIPT}")

message(STATUS " Support Microsoft Document Imaging: ${mdi}")

message(STATUS " Use win32 IO: ${USE\_WIN32\_FILEIO}")

message(STATUS "")

message(STATUS " Support for internal codecs:")

message(STATUS " CCITT Group 3 & 4 algorithms: ${ccitt}")

message(STATUS " Macintosh PackBits algorithm: ${packbits}")

message(STATUS " LZW algorithm: ${lzw}")

message(STATUS " ThunderScan 4-bit RLE algorithm: ${thunder}")

message(STATUS " NeXT 2-bit RLE algorithm: ${next}")

message(STATUS " LogLuv high dynamic range encoding: ${logluv}")

message(STATUS "")

message(STATUS " Support for external codecs:")

message(STATUS " ZLIB support: ${zlib} (requested) ${ZLIB\_FOUND} (availability)")

message(STATUS " Pixar log-format algorithm: ${pixarlog} (requested) ${PIXARLOG\_SUPPORT} (availability)")

message(STATUS " JPEG support: ${jpeg} (requested) ${JPEG\_FOUND} (availability)")

message(STATUS " Old JPEG support: ${old-jpeg} (requested) ${JPEG\_FOUND} (availability)")

message(STATUS " JPEG 8/12 bit dual mode: ${jpeg12} (requested) ${JPEG12\_FOUND} (availability)")

message(STATUS " ISO JBIG support: ${jbig} (requested) ${JBIG\_FOUND} (availability)")

message(STATUS " LZMA2 support: ${lzma} (requested) ${LIBLZMA\_FOUND} (availability)")

message(STATUS "")

message(STATUS " C++ support: ${cxx} (requested) ${CXX\_SUPPORT} (availability)")

message(STATUS "")

# message(STATUS " X Athena Widgets support: ${HAVE\_XAW}")

message(STATUS " OpenGL support: ${HAVE\_OPENGL}")

message(STATUS "")