/\* This is an implementation of the threads API of POSIX 1003.1-2001.

\*

\* --------------------------------------------------------------------------

\*

\* Pthreads-win32 - POSIX Threads Library for Win32

\* Copyright(C) 1998 John E. Bossom

\* Copyright(C) 1999,2005 Pthreads-win32 contributors

\*

\* Contact Email: rpj@callisto.canberra.edu.au

\*

\* The current list of contributors is contained

\* in the file CONTRIBUTORS included with the source

\* code distribution. The list can also be seen at the

\* following World Wide Web location:

\* http://sources.redhat.com/pthreads-win32/contributors.html

\*

\* This library is free software; you can redistribute it and/or

\* modify it under the terms of the GNU Lesser General Public

\* License as published by the Free Software Foundation; either

\* version 2 of the License, or (at your option) any later version.

\*

\* This library is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

\* Lesser General Public License for more details.

\*

\* You should have received a copy of the GNU Lesser General Public

\* License along with this library in the file COPYING.LIB;

\* if not, write to the Free Software Foundation, Inc.,

\* 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

\*/

#if !defined( PTHREAD\_H )

#define PTHREAD\_H

/\*

\* See the README file for an explanation of the pthreads-win32 version

\* numbering scheme and how the DLL is named etc.

\*/

#define PTW32\_VERSION 2,9,1,0

#define PTW32\_VERSION\_STRING "2, 9, 1, 0\0"

/\* There are three implementations of cancel cleanup.

\* Note that pthread.h is included in both application

\* compilation units and also internally for the library.

\* The code here and within the library aims to work

\* for all reasonable combinations of environments.

\*

\* The three implementations are:

\*

\* WIN32 SEH

\* C

\* C++

\*

\* Please note that exiting a push/pop block via

\* "return", "exit", "break", or "continue" will

\* lead to different behaviour amongst applications

\* depending upon whether the library was built

\* using SEH, C++, or C. For example, a library built

\* with SEH will call the cleanup routine, while both

\* C++ and C built versions will not.

\*/

/\*

\* Define defaults for cleanup code.

\* Note: Unless the build explicitly defines one of the following, then

\* we default to standard C style cleanup. This style uses setjmp/longjmp

\* in the cancelation and thread exit implementations and therefore won't

\* do stack unwinding if linked to applications that have it (e.g.

\* C++ apps). This is currently consistent with most/all commercial Unix

\* POSIX threads implementations.

\*/

#if !defined( \_\_CLEANUP\_SEH ) && !defined( \_\_CLEANUP\_CXX ) && !defined( \_\_CLEANUP\_C )

# define \_\_CLEANUP\_C

#endif

#if defined( \_\_CLEANUP\_SEH ) && ( !defined( \_MSC\_VER ) && !defined(PTW32\_RC\_MSC))

#error ERROR [\_\_FILE\_\_, line \_\_LINE\_\_]: SEH is not supported for this compiler.

#endif

/\*

\* Stop here if we are being included by the resource compiler.

\*/

#if !defined(RC\_INVOKED)

#undef PTW32\_LEVEL

#if defined(\_POSIX\_SOURCE)

#define PTW32\_LEVEL 0

/\* Early POSIX \*/

#endif

#if defined(\_POSIX\_C\_SOURCE) && \_POSIX\_C\_SOURCE >= 199309

#undef PTW32\_LEVEL

#define PTW32\_LEVEL 1

/\* Include 1b, 1c and 1d \*/

#endif

#if defined(INCLUDE\_NP)

#undef PTW32\_LEVEL

#define PTW32\_LEVEL 2

/\* Include Non-Portable extensions \*/

#endif

#define PTW32\_LEVEL\_MAX 3

#if ( defined(\_POSIX\_C\_SOURCE) && \_POSIX\_C\_SOURCE >= 200112 ) || !defined(PTW32\_LEVEL)

#define PTW32\_LEVEL PTW32\_LEVEL\_MAX

/\* Include everything \*/

#endif

#if defined(\_UWIN)

# define HAVE\_STRUCT\_TIMESPEC 1

# define HAVE\_SIGNAL\_H 1

# undef HAVE\_PTW32\_CONFIG\_H

# pragma comment(lib, "pthread")

#endif

/\*

\* -------------------------------------------------------------

\*

\*

\* Module: pthread.h

\*

\* Purpose:

\* Provides an implementation of PThreads based upon the

\* standard:

\*

\* POSIX 1003.1-2001

\* and

\* The Single Unix Specification version 3

\*

\* (these two are equivalent)

\*

\* in order to enhance code portability between Windows,

\* various commercial Unix implementations, and Linux.

\*

\* See the ANNOUNCE file for a full list of conforming

\* routines and defined constants, and a list of missing

\* routines and constants not defined in this implementation.

\*

\* Authors:

\* There have been many contributors to this library.

\* The initial implementation was contributed by

\* John Bossom, and several others have provided major

\* sections or revisions of parts of the implementation.

\* Often significant effort has been contributed to

\* find and fix important bugs and other problems to

\* improve the reliability of the library, which sometimes

\* is not reflected in the amount of code which changed as

\* result.

\* As much as possible, the contributors are acknowledged

\* in the ChangeLog file in the source code distribution

\* where their changes are noted in detail.

\*

\* Contributors are listed in the CONTRIBUTORS file.

\*

\* As usual, all bouquets go to the contributors, and all

\* brickbats go to the project maintainer.

\*

\* Maintainer:

\* The code base for this project is coordinated and

\* eventually pre-tested, packaged, and made available by

\*

\* Ross Johnson <rpj@callisto.canberra.edu.au>

\*

\* QA Testers:

\* Ultimately, the library is tested in the real world by

\* a host of competent and demanding scientists and

\* engineers who report bugs and/or provide solutions

\* which are then fixed or incorporated into subsequent

\* versions of the library. Each time a bug is fixed, a

\* test case is written to prove the fix and ensure

\* that later changes to the code don't reintroduce the

\* same error. The number of test cases is slowly growing

\* and therefore so is the code reliability.

\*

\* Compliance:

\* See the file ANNOUNCE for the list of implemented

\* and not-implemented routines and defined options.

\* Of course, these are all defined is this file as well.

\*

\* Web site:

\* The source code and other information about this library

\* are available from

\*

\* http://sources.redhat.com/pthreads-win32/

\*

\* -------------------------------------------------------------

\*/

/\* Try to avoid including windows.h \*/

#if (defined(\_\_MINGW64\_\_) || defined(\_\_MINGW32\_\_)) && defined(\_\_cplusplus)

#define PTW32\_INCLUDE\_WINDOWS\_H

#endif

#if defined(PTW32\_INCLUDE\_WINDOWS\_H)

#include <windows.h>

#endif

#if defined(\_MSC\_VER) && \_MSC\_VER < 1300 || defined(\_\_DMC\_\_)

/\*

\* VC++6.0 or early compiler's header has no DWORD\_PTR type.

\*/

typedef unsigned long DWORD\_PTR;

typedef unsigned long ULONG\_PTR;

#endif

/\*

\* -----------------

\* autoconf switches

\* -----------------

\*/

#if defined(HAVE\_PTW32\_CONFIG\_H)

#include "config.h"

#endif /\* HAVE\_PTW32\_CONFIG\_H \*/

#if !defined(NEED\_FTIME)

#include <time.h>

#else /\* NEED\_FTIME \*/

/\* use native WIN32 time API \*/

#endif /\* NEED\_FTIME \*/

#if defined(HAVE\_SIGNAL\_H)

#include <signal.h>

#endif /\* HAVE\_SIGNAL\_H \*/

#include <limits.h>

/\*

\* Boolean values to make us independent of system includes.

\*/

enum {

PTW32\_FALSE = 0,

PTW32\_TRUE = (! PTW32\_FALSE)

};

/\*

\* This is a duplicate of what is in the autoconf config.h,

\* which is only used when building the pthread-win32 libraries.

\*/

#if !defined(PTW32\_CONFIG\_H)

# if defined(WINCE)

# define NEED\_ERRNO

# define NEED\_SEM

# endif

# if defined(\_\_MINGW64\_\_)

# define HAVE\_STRUCT\_TIMESPEC

# define HAVE\_MODE\_T

# elif defined(\_UWIN) || defined(\_\_MINGW32\_\_)

# define HAVE\_MODE\_T

# endif

#endif

/\*

\*

\*/

#if PTW32\_LEVEL >= PTW32\_LEVEL\_MAX

#if defined(NEED\_ERRNO)

#include "need\_errno.h"

#else

#include <errno.h>

#endif

#endif /\* PTW32\_LEVEL >= PTW32\_LEVEL\_MAX \*/

/\*

\* Several systems don't define some error numbers.

\*/

#if !defined(ENOTSUP)

# define ENOTSUP 48 /\* This is the value in Solaris. \*/

#endif

#if !defined(ETIMEDOUT)

# define ETIMEDOUT 10060 /\* Same as WSAETIMEDOUT \*/

#endif

#if !defined(ENOSYS)

# define ENOSYS 140 /\* Semi-arbitrary value \*/

#endif

#if !defined(EDEADLK)

# if defined(EDEADLOCK)

# define EDEADLK EDEADLOCK

# else

# define EDEADLK 36 /\* This is the value in MSVC. \*/

# endif

#endif

/\* POSIX 2008 - related to robust mutexes \*/

#if !defined(EOWNERDEAD)

# define EOWNERDEAD 43

#endif

#if !defined(ENOTRECOVERABLE)

# define ENOTRECOVERABLE 44

#endif

#include <sched.h>

/\*

\* To avoid including windows.h we define only those things that we

\* actually need from it.

\*/

#if !defined(PTW32\_INCLUDE\_WINDOWS\_H)

#if !defined(HANDLE)

# define PTW32\_\_HANDLE\_DEF

# define HANDLE void \*

#endif

#if !defined(DWORD)

# define PTW32\_\_DWORD\_DEF

# define DWORD unsigned long

#endif

#endif

#if !defined(HAVE\_STRUCT\_TIMESPEC)

#define HAVE\_STRUCT\_TIMESPEC

#if !defined(\_TIMESPEC\_DEFINED)

#define \_TIMESPEC\_DEFINED

struct timespec {

time\_t tv\_sec;

long tv\_nsec;

};

#endif /\* \_TIMESPEC\_DEFINED \*/

#endif /\* HAVE\_STRUCT\_TIMESPEC \*/

#if !defined(SIG\_BLOCK)

#define SIG\_BLOCK 0

#endif /\* SIG\_BLOCK \*/

#if !defined(SIG\_UNBLOCK)

#define SIG\_UNBLOCK 1

#endif /\* SIG\_UNBLOCK \*/

#if !defined(SIG\_SETMASK)

#define SIG\_SETMASK 2

#endif /\* SIG\_SETMASK \*/

#if defined(\_\_cplusplus)

extern "C"

{

#endif /\* \_\_cplusplus \*/

/\*

\* -------------------------------------------------------------

\*

\* POSIX 1003.1-2001 Options

\* =========================

\*

\* Options are normally set in <unistd.h>, which is not provided

\* with pthreads-win32.

\*

\* For conformance with the Single Unix Specification (version 3), all of the

\* options below are defined, and have a value of either -1 (not supported)

\* or 200112L (supported).

\*

\* These options can neither be left undefined nor have a value of 0, because

\* either indicates that sysconf(), which is not implemented, may be used at

\* runtime to check the status of the option.

\*

\* \_POSIX\_THREADS (== 200112L)

\* If == 200112L, you can use threads

\*

\* \_POSIX\_THREAD\_ATTR\_STACKSIZE (== 200112L)

\* If == 200112L, you can control the size of a thread's

\* stack

\* pthread\_attr\_getstacksize

\* pthread\_attr\_setstacksize

\*

\* \_POSIX\_THREAD\_ATTR\_STACKADDR (== -1)

\* If == 200112L, you can allocate and control a thread's

\* stack. If not supported, the following functions

\* will return ENOSYS, indicating they are not

\* supported:

\* pthread\_attr\_getstackaddr

\* pthread\_attr\_setstackaddr

\*

\* \_POSIX\_THREAD\_PRIORITY\_SCHEDULING (== -1)

\* If == 200112L, you can use realtime scheduling.

\* This option indicates that the behaviour of some

\* implemented functions conforms to the additional TPS

\* requirements in the standard. E.g. rwlocks favour

\* writers over readers when threads have equal priority.

\*

\* \_POSIX\_THREAD\_PRIO\_INHERIT (== -1)

\* If == 200112L, you can create priority inheritance

\* mutexes.

\* pthread\_mutexattr\_getprotocol +

\* pthread\_mutexattr\_setprotocol +

\*

\* \_POSIX\_THREAD\_PRIO\_PROTECT (== -1)

\* If == 200112L, you can create priority ceiling mutexes

\* Indicates the availability of:

\* pthread\_mutex\_getprioceiling

\* pthread\_mutex\_setprioceiling

\* pthread\_mutexattr\_getprioceiling

\* pthread\_mutexattr\_getprotocol +

\* pthread\_mutexattr\_setprioceiling

\* pthread\_mutexattr\_setprotocol +

\*

\* \_POSIX\_THREAD\_PROCESS\_SHARED (== -1)

\* If set, you can create mutexes and condition

\* variables that can be shared with another

\* process.If set, indicates the availability

\* of:

\* pthread\_mutexattr\_getpshared

\* pthread\_mutexattr\_setpshared

\* pthread\_condattr\_getpshared

\* pthread\_condattr\_setpshared

\*

\* \_POSIX\_THREAD\_SAFE\_FUNCTIONS (== 200112L)

\* If == 200112L you can use the special \*\_r library

\* functions that provide thread-safe behaviour

\*

\* \_POSIX\_READER\_WRITER\_LOCKS (== 200112L)

\* If == 200112L, you can use read/write locks

\*

\* \_POSIX\_SPIN\_LOCKS (== 200112L)

\* If == 200112L, you can use spin locks

\*

\* \_POSIX\_BARRIERS (== 200112L)

\* If == 200112L, you can use barriers

\*

\* + These functions provide both 'inherit' and/or

\* 'protect' protocol, based upon these macro

\* settings.

\*

\* -------------------------------------------------------------

\*/

/\*

\* POSIX Options

\*/

#undef \_POSIX\_THREADS

#define \_POSIX\_THREADS 200809L

#undef \_POSIX\_READER\_WRITER\_LOCKS

#define \_POSIX\_READER\_WRITER\_LOCKS 200809L

#undef \_POSIX\_SPIN\_LOCKS

#define \_POSIX\_SPIN\_LOCKS 200809L

#undef \_POSIX\_BARRIERS

#define \_POSIX\_BARRIERS 200809L

#undef \_POSIX\_THREAD\_SAFE\_FUNCTIONS

#define \_POSIX\_THREAD\_SAFE\_FUNCTIONS 200809L

#undef \_POSIX\_THREAD\_ATTR\_STACKSIZE

#define \_POSIX\_THREAD\_ATTR\_STACKSIZE 200809L

/\*

\* The following options are not supported

\*/

#undef \_POSIX\_THREAD\_ATTR\_STACKADDR

#define \_POSIX\_THREAD\_ATTR\_STACKADDR -1

#undef \_POSIX\_THREAD\_PRIO\_INHERIT

#define \_POSIX\_THREAD\_PRIO\_INHERIT -1

#undef \_POSIX\_THREAD\_PRIO\_PROTECT

#define \_POSIX\_THREAD\_PRIO\_PROTECT -1

/\* TPS is not fully supported. \*/

#undef \_POSIX\_THREAD\_PRIORITY\_SCHEDULING

#define \_POSIX\_THREAD\_PRIORITY\_SCHEDULING -1

#undef \_POSIX\_THREAD\_PROCESS\_SHARED

#define \_POSIX\_THREAD\_PROCESS\_SHARED -1

/\*

\* POSIX 1003.1-2001 Limits

\* ===========================

\*

\* These limits are normally set in <limits.h>, which is not provided with

\* pthreads-win32.

\*

\* PTHREAD\_DESTRUCTOR\_ITERATIONS

\* Maximum number of attempts to destroy

\* a thread's thread-specific data on

\* termination (must be at least 4)

\*

\* PTHREAD\_KEYS\_MAX

\* Maximum number of thread-specific data keys

\* available per process (must be at least 128)

\*

\* PTHREAD\_STACK\_MIN

\* Minimum supported stack size for a thread

\*

\* PTHREAD\_THREADS\_MAX

\* Maximum number of threads supported per

\* process (must be at least 64).

\*

\* SEM\_NSEMS\_MAX

\* The maximum number of semaphores a process can have.

\* (must be at least 256)

\*

\* SEM\_VALUE\_MAX

\* The maximum value a semaphore can have.

\* (must be at least 32767)

\*

\*/

#undef \_POSIX\_THREAD\_DESTRUCTOR\_ITERATIONS

#define \_POSIX\_THREAD\_DESTRUCTOR\_ITERATIONS 4

#undef PTHREAD\_DESTRUCTOR\_ITERATIONS

#define PTHREAD\_DESTRUCTOR\_ITERATIONS \_POSIX\_THREAD\_DESTRUCTOR\_ITERATIONS

#undef \_POSIX\_THREAD\_KEYS\_MAX

#define \_POSIX\_THREAD\_KEYS\_MAX 128

#undef PTHREAD\_KEYS\_MAX

#define PTHREAD\_KEYS\_MAX \_POSIX\_THREAD\_KEYS\_MAX

#undef PTHREAD\_STACK\_MIN

#define PTHREAD\_STACK\_MIN 0

#undef \_POSIX\_THREAD\_THREADS\_MAX

#define \_POSIX\_THREAD\_THREADS\_MAX 64

/\* Arbitrary value \*/

#undef PTHREAD\_THREADS\_MAX

#define PTHREAD\_THREADS\_MAX 2019

#undef \_POSIX\_SEM\_NSEMS\_MAX

#define \_POSIX\_SEM\_NSEMS\_MAX 256

/\* Arbitrary value \*/

#undef SEM\_NSEMS\_MAX

#define SEM\_NSEMS\_MAX 1024

#undef \_POSIX\_SEM\_VALUE\_MAX

#define \_POSIX\_SEM\_VALUE\_MAX 32767

#undef SEM\_VALUE\_MAX

#define SEM\_VALUE\_MAX INT\_MAX

#if defined(\_\_GNUC\_\_) && !defined(\_\_declspec)

# error Please upgrade your GNU compiler to one that supports \_\_declspec.

#endif

/\*

\* When building the library, you should define PTW32\_BUILD so that

\* the variables/functions are exported correctly. When using the library,

\* do NOT define PTW32\_BUILD, and then the variables/functions will

\* be imported correctly.

\*/

#if !defined(PTW32\_STATIC\_LIB)

# if defined(PTW32\_BUILD)

# define PTW32\_DLLPORT \_\_declspec (dllexport)

# else

# define PTW32\_DLLPORT \_\_declspec (dllimport)

# endif

#else

# define PTW32\_DLLPORT

#endif

/\*

\* The Open Watcom C/C++ compiler uses a non-standard calling convention

\* that passes function args in registers unless \_\_cdecl is explicitly specified

\* in exposed function prototypes.

\*

\* We force all calls to cdecl even though this could slow Watcom code down

\* slightly. If you know that the Watcom compiler will be used to build both

\* the DLL and application, then you can probably define this as a null string.

\* Remember that pthread.h (this file) is used for both the DLL and application builds.

\*/

#define PTW32\_CDECL \_\_cdecl

#if defined(\_UWIN) && PTW32\_LEVEL >= PTW32\_LEVEL\_MAX

# include <sys/types.h>

#else

/\*

\* Generic handle type - intended to extend uniqueness beyond

\* that available with a simple pointer. It should scale for either

\* IA-32 or IA-64.

\*/

typedef struct {

void \* p; /\* Pointer to actual object \*/

unsigned int x; /\* Extra information - reuse count etc \*/

} ptw32\_handle\_t;

typedef ptw32\_handle\_t pthread\_t;

typedef struct pthread\_attr\_t\_ \* pthread\_attr\_t;

typedef struct pthread\_once\_t\_ pthread\_once\_t;

typedef struct pthread\_key\_t\_ \* pthread\_key\_t;

typedef struct pthread\_mutex\_t\_ \* pthread\_mutex\_t;

typedef struct pthread\_mutexattr\_t\_ \* pthread\_mutexattr\_t;

typedef struct pthread\_cond\_t\_ \* pthread\_cond\_t;

typedef struct pthread\_condattr\_t\_ \* pthread\_condattr\_t;

#endif

typedef struct pthread\_rwlock\_t\_ \* pthread\_rwlock\_t;

typedef struct pthread\_rwlockattr\_t\_ \* pthread\_rwlockattr\_t;

typedef struct pthread\_spinlock\_t\_ \* pthread\_spinlock\_t;

typedef struct pthread\_barrier\_t\_ \* pthread\_barrier\_t;

typedef struct pthread\_barrierattr\_t\_ \* pthread\_barrierattr\_t;

/\*

\* ====================

\* ====================

\* POSIX Threads

\* ====================

\* ====================

\*/

enum {

/\*

\* pthread\_attr\_{get,set}detachstate

\*/

PTHREAD\_CREATE\_JOINABLE = 0, /\* Default \*/

PTHREAD\_CREATE\_DETACHED = 1,

/\*

\* pthread\_attr\_{get,set}inheritsched

\*/

PTHREAD\_INHERIT\_SCHED = 0,

PTHREAD\_EXPLICIT\_SCHED = 1, /\* Default \*/

/\*

\* pthread\_{get,set}scope

\*/

PTHREAD\_SCOPE\_PROCESS = 0,

PTHREAD\_SCOPE\_SYSTEM = 1, /\* Default \*/

/\*

\* pthread\_setcancelstate paramters

\*/

PTHREAD\_CANCEL\_ENABLE = 0, /\* Default \*/

PTHREAD\_CANCEL\_DISABLE = 1,

/\*

\* pthread\_setcanceltype parameters

\*/

PTHREAD\_CANCEL\_ASYNCHRONOUS = 0,

PTHREAD\_CANCEL\_DEFERRED = 1, /\* Default \*/

/\*

\* pthread\_mutexattr\_{get,set}pshared

\* pthread\_condattr\_{get,set}pshared

\*/

PTHREAD\_PROCESS\_PRIVATE = 0,

PTHREAD\_PROCESS\_SHARED = 1,

/\*

\* pthread\_mutexattr\_{get,set}robust

\*/

PTHREAD\_MUTEX\_STALLED = 0, /\* Default \*/

PTHREAD\_MUTEX\_ROBUST = 1,

/\*

\* pthread\_barrier\_wait

\*/

PTHREAD\_BARRIER\_SERIAL\_THREAD = -1

};

/\*

\* ====================

\* ====================

\* Cancelation

\* ====================

\* ====================

\*/

#define PTHREAD\_CANCELED ((void \*)(size\_t) -1)

/\*

\* ====================

\* ====================

\* Once Key

\* ====================

\* ====================

\*/

#define PTHREAD\_ONCE\_INIT { PTW32\_FALSE, 0, 0, 0}

struct pthread\_once\_t\_

{

int done; /\* indicates if user function has been executed \*/

void \* lock;

int reserved1;

int reserved2;

};

/\*

\* ====================

\* ====================

\* Object initialisers

\* ====================

\* ====================

\*/

#define PTHREAD\_MUTEX\_INITIALIZER ((pthread\_mutex\_t)(size\_t) -1)

#define PTHREAD\_RECURSIVE\_MUTEX\_INITIALIZER ((pthread\_mutex\_t)(size\_t) -2)

#define PTHREAD\_ERRORCHECK\_MUTEX\_INITIALIZER ((pthread\_mutex\_t)(size\_t) -3)

/\*

\* Compatibility with LinuxThreads

\*/

#define PTHREAD\_RECURSIVE\_MUTEX\_INITIALIZER\_NP PTHREAD\_RECURSIVE\_MUTEX\_INITIALIZER

#define PTHREAD\_ERRORCHECK\_MUTEX\_INITIALIZER\_NP PTHREAD\_ERRORCHECK\_MUTEX\_INITIALIZER

#define PTHREAD\_COND\_INITIALIZER ((pthread\_cond\_t)(size\_t) -1)

#define PTHREAD\_RWLOCK\_INITIALIZER ((pthread\_rwlock\_t)(size\_t) -1)

#define PTHREAD\_SPINLOCK\_INITIALIZER ((pthread\_spinlock\_t)(size\_t) -1)

/\*

\* Mutex types.

\*/

enum

{

/\* Compatibility with LinuxThreads \*/

PTHREAD\_MUTEX\_FAST\_NP,

PTHREAD\_MUTEX\_RECURSIVE\_NP,

PTHREAD\_MUTEX\_ERRORCHECK\_NP,

PTHREAD\_MUTEX\_TIMED\_NP = PTHREAD\_MUTEX\_FAST\_NP,

PTHREAD\_MUTEX\_ADAPTIVE\_NP = PTHREAD\_MUTEX\_FAST\_NP,

/\* For compatibility with POSIX \*/

PTHREAD\_MUTEX\_NORMAL = PTHREAD\_MUTEX\_FAST\_NP,

PTHREAD\_MUTEX\_RECURSIVE = PTHREAD\_MUTEX\_RECURSIVE\_NP,

PTHREAD\_MUTEX\_ERRORCHECK = PTHREAD\_MUTEX\_ERRORCHECK\_NP,

PTHREAD\_MUTEX\_DEFAULT = PTHREAD\_MUTEX\_NORMAL

};

typedef struct ptw32\_cleanup\_t ptw32\_cleanup\_t;

#if defined(\_MSC\_VER)

/\* Disable MSVC 'anachronism used' warning \*/

#pragma warning( disable : 4229 )

#endif

typedef void (\* PTW32\_CDECL ptw32\_cleanup\_callback\_t)(void \*);

#if defined(\_MSC\_VER)

#pragma warning( default : 4229 )

#endif

struct ptw32\_cleanup\_t

{

ptw32\_cleanup\_callback\_t routine;

void \*arg;

struct ptw32\_cleanup\_t \*prev;

};

#if defined(\_\_CLEANUP\_SEH)

/\*

\* WIN32 SEH version of cancel cleanup.

\*/

#define pthread\_cleanup\_push( \_rout, \_arg ) \

{ \

ptw32\_cleanup\_t \_cleanup; \

\

\_cleanup.routine = (ptw32\_cleanup\_callback\_t)(\_rout); \

\_cleanup.arg = (\_arg); \

\_\_try \

{ \

#define pthread\_cleanup\_pop( \_execute ) \

} \

\_\_finally \

{ \

if( \_execute || AbnormalTermination()) \

{ \

(\*(\_cleanup.routine))( \_cleanup.arg ); \

} \

} \

}

#else /\* \_\_CLEANUP\_SEH \*/

#if defined(\_\_CLEANUP\_C)

/\*

\* C implementation of PThreads cancel cleanup

\*/

#define pthread\_cleanup\_push( \_rout, \_arg ) \

{ \

ptw32\_cleanup\_t \_cleanup; \

\

ptw32\_push\_cleanup( &\_cleanup, (ptw32\_cleanup\_callback\_t) (\_rout), (\_arg) ); \

#define pthread\_cleanup\_pop( \_execute ) \

(void) ptw32\_pop\_cleanup( \_execute ); \

}

#else /\* \_\_CLEANUP\_C \*/

#if defined(\_\_CLEANUP\_CXX)

/\*

\* C++ version of cancel cleanup.

\* - John E. Bossom.

\*/

class PThreadCleanup {

/\*

\* PThreadCleanup

\*

\* Purpose

\* This class is a C++ helper class that is

\* used to implement pthread\_cleanup\_push/

\* pthread\_cleanup\_pop.

\* The destructor of this class automatically

\* pops the pushed cleanup routine regardless

\* of how the code exits the scope

\* (i.e. such as by an exception)

\*/

ptw32\_cleanup\_callback\_t cleanUpRout;

void \* obj;

int executeIt;

public:

PThreadCleanup() :

cleanUpRout( 0 ),

obj( 0 ),

executeIt( 0 )

/\*

\* No cleanup performed

\*/

{

}

PThreadCleanup(

ptw32\_cleanup\_callback\_t routine,

void \* arg ) :

cleanUpRout( routine ),

obj( arg ),

executeIt( 1 )

/\*

\* Registers a cleanup routine for 'arg'

\*/

{

}

~PThreadCleanup()

{

if ( executeIt && ((void \*) cleanUpRout != (void \*) 0) )

{

(void) (\*cleanUpRout)( obj );

}

}

void execute( int exec )

{

executeIt = exec;

}

};

/\*

\* C++ implementation of PThreads cancel cleanup;

\* This implementation takes advantage of a helper

\* class who's destructor automatically calls the

\* cleanup routine if we exit our scope weirdly

\*/

#define pthread\_cleanup\_push( \_rout, \_arg ) \

{ \

PThreadCleanup cleanup((ptw32\_cleanup\_callback\_t)(\_rout), \

(void \*) (\_arg) );

#define pthread\_cleanup\_pop( \_execute ) \

cleanup.execute( \_execute ); \

}

#else

#error ERROR [\_\_FILE\_\_, line \_\_LINE\_\_]: Cleanup type undefined.

#endif /\* \_\_CLEANUP\_CXX \*/

#endif /\* \_\_CLEANUP\_C \*/

#endif /\* \_\_CLEANUP\_SEH \*/

/\*

\* ===============

\* ===============

\* Methods

\* ===============

\* ===============

\*/

/\*

\* PThread Attribute Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_init (pthread\_attr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_destroy (pthread\_attr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getdetachstate (const pthread\_attr\_t \* attr,

int \*detachstate);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getstackaddr (const pthread\_attr\_t \* attr,

void \*\*stackaddr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getstacksize (const pthread\_attr\_t \* attr,

size\_t \* stacksize);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setdetachstate (pthread\_attr\_t \* attr,

int detachstate);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setstackaddr (pthread\_attr\_t \* attr,

void \*stackaddr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setstacksize (pthread\_attr\_t \* attr,

size\_t stacksize);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getschedparam (const pthread\_attr\_t \*attr,

struct sched\_param \*param);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setschedparam (pthread\_attr\_t \*attr,

const struct sched\_param \*param);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setschedpolicy (pthread\_attr\_t \*,

int);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getschedpolicy (const pthread\_attr\_t \*,

int \*);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setinheritsched(pthread\_attr\_t \* attr,

int inheritsched);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getinheritsched(const pthread\_attr\_t \* attr,

int \* inheritsched);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_setscope (pthread\_attr\_t \*,

int);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_attr\_getscope (const pthread\_attr\_t \*,

int \*);

/\*

\* PThread Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_create (pthread\_t \* tid,

const pthread\_attr\_t \* attr,

void \*(PTW32\_CDECL \*start) (void \*),

void \*arg);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_detach (pthread\_t tid);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_equal (pthread\_t t1,

pthread\_t t2);

PTW32\_DLLPORT void PTW32\_CDECL pthread\_exit (void \*value\_ptr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_join (pthread\_t thread,

void \*\*value\_ptr);

PTW32\_DLLPORT pthread\_t PTW32\_CDECL pthread\_self (void);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cancel (pthread\_t thread);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_setcancelstate (int state,

int \*oldstate);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_setcanceltype (int type,

int \*oldtype);

PTW32\_DLLPORT void PTW32\_CDECL pthread\_testcancel (void);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_once (pthread\_once\_t \* once\_control,

void (PTW32\_CDECL \*init\_routine) (void));

#if PTW32\_LEVEL >= PTW32\_LEVEL\_MAX

PTW32\_DLLPORT ptw32\_cleanup\_t \* PTW32\_CDECL ptw32\_pop\_cleanup (int execute);

PTW32\_DLLPORT void PTW32\_CDECL ptw32\_push\_cleanup (ptw32\_cleanup\_t \* cleanup,

ptw32\_cleanup\_callback\_t routine,

void \*arg);

#endif /\* PTW32\_LEVEL >= PTW32\_LEVEL\_MAX \*/

/\*

\* Thread Specific Data Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_key\_create (pthread\_key\_t \* key,

void (PTW32\_CDECL \*destructor) (void \*));

PTW32\_DLLPORT int PTW32\_CDECL pthread\_key\_delete (pthread\_key\_t key);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_setspecific (pthread\_key\_t key,

const void \*value);

PTW32\_DLLPORT void \* PTW32\_CDECL pthread\_getspecific (pthread\_key\_t key);

/\*

\* Mutex Attribute Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_init (pthread\_mutexattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_destroy (pthread\_mutexattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_getpshared (const pthread\_mutexattr\_t

\* attr,

int \*pshared);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_setpshared (pthread\_mutexattr\_t \* attr,

int pshared);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_settype (pthread\_mutexattr\_t \* attr, int kind);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_gettype (const pthread\_mutexattr\_t \* attr, int \*kind);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_setrobust(

pthread\_mutexattr\_t \*attr,

int robust);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_getrobust(

const pthread\_mutexattr\_t \* attr,

int \* robust);

/\*

\* Barrier Attribute Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrierattr\_init (pthread\_barrierattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrierattr\_destroy (pthread\_barrierattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrierattr\_getpshared (const pthread\_barrierattr\_t

\* attr,

int \*pshared);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrierattr\_setpshared (pthread\_barrierattr\_t \* attr,

int pshared);

/\*

\* Mutex Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_init (pthread\_mutex\_t \* mutex,

const pthread\_mutexattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_destroy (pthread\_mutex\_t \* mutex);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_lock (pthread\_mutex\_t \* mutex);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_timedlock(pthread\_mutex\_t \* mutex,

const struct timespec \*abstime);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_trylock (pthread\_mutex\_t \* mutex);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_unlock (pthread\_mutex\_t \* mutex);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutex\_consistent (pthread\_mutex\_t \* mutex);

/\*

\* Spinlock Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_spin\_init (pthread\_spinlock\_t \* lock, int pshared);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_spin\_destroy (pthread\_spinlock\_t \* lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_spin\_lock (pthread\_spinlock\_t \* lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_spin\_trylock (pthread\_spinlock\_t \* lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_spin\_unlock (pthread\_spinlock\_t \* lock);

/\*

\* Barrier Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrier\_init (pthread\_barrier\_t \* barrier,

const pthread\_barrierattr\_t \* attr,

unsigned int count);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrier\_destroy (pthread\_barrier\_t \* barrier);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_barrier\_wait (pthread\_barrier\_t \* barrier);

/\*

\* Condition Variable Attribute Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_condattr\_init (pthread\_condattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_condattr\_destroy (pthread\_condattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_condattr\_getpshared (const pthread\_condattr\_t \* attr,

int \*pshared);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_condattr\_setpshared (pthread\_condattr\_t \* attr,

int pshared);

/\*

\* Condition Variable Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cond\_init (pthread\_cond\_t \* cond,

const pthread\_condattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cond\_destroy (pthread\_cond\_t \* cond);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cond\_wait (pthread\_cond\_t \* cond,

pthread\_mutex\_t \* mutex);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cond\_timedwait (pthread\_cond\_t \* cond,

pthread\_mutex\_t \* mutex,

const struct timespec \*abstime);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cond\_signal (pthread\_cond\_t \* cond);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_cond\_broadcast (pthread\_cond\_t \* cond);

/\*

\* Scheduling

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_setschedparam (pthread\_t thread,

int policy,

const struct sched\_param \*param);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_getschedparam (pthread\_t thread,

int \*policy,

struct sched\_param \*param);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_setconcurrency (int);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_getconcurrency (void);

/\*

\* Read-Write Lock Functions

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_init(pthread\_rwlock\_t \*lock,

const pthread\_rwlockattr\_t \*attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_destroy(pthread\_rwlock\_t \*lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_timedrdlock(pthread\_rwlock\_t \*lock,

const struct timespec \*abstime);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_timedwrlock(pthread\_rwlock\_t \*lock,

const struct timespec \*abstime);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlock\_unlock(pthread\_rwlock\_t \*lock);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlockattr\_init (pthread\_rwlockattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlockattr\_destroy (pthread\_rwlockattr\_t \* attr);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlockattr\_getpshared (const pthread\_rwlockattr\_t \* attr,

int \*pshared);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_rwlockattr\_setpshared (pthread\_rwlockattr\_t \* attr,

int pshared);

#if PTW32\_LEVEL >= PTW32\_LEVEL\_MAX - 1

/\*

\* Signal Functions. Should be defined in <signal.h> but MSVC and MinGW32

\* already have signal.h that don't define these.

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_kill(pthread\_t thread, int sig);

/\*

\* Non-portable functions

\*/

/\*

\* Compatibility with Linux.

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_setkind\_np(pthread\_mutexattr\_t \* attr,

int kind);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_mutexattr\_getkind\_np(pthread\_mutexattr\_t \* attr,

int \*kind);

/\*

\* Possibly supported by other POSIX threads implementations

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_delay\_np (struct timespec \* interval);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_num\_processors\_np(void);

PTW32\_DLLPORT unsigned \_\_int64 PTW32\_CDECL pthread\_getunique\_np(pthread\_t thread);

/\*

\* Useful if an application wants to statically link

\* the lib rather than load the DLL at run-time.

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_win32\_process\_attach\_np(void);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_win32\_process\_detach\_np(void);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_win32\_thread\_attach\_np(void);

PTW32\_DLLPORT int PTW32\_CDECL pthread\_win32\_thread\_detach\_np(void);

/\*

\* Features that are auto-detected at load/run time.

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthread\_win32\_test\_features\_np(int);

enum ptw32\_features {

PTW32\_SYSTEM\_INTERLOCKED\_COMPARE\_EXCHANGE = 0x0001, /\* System provides it. \*/

PTW32\_ALERTABLE\_ASYNC\_CANCEL = 0x0002 /\* Can cancel blocked threads. \*/

};

/\*

\* Register a system time change with the library.

\* Causes the library to perform various functions

\* in response to the change. Should be called whenever

\* the application's top level window receives a

\* WM\_TIMECHANGE message. It can be passed directly to

\* pthread\_create() as a new thread if desired.

\*/

PTW32\_DLLPORT void \* PTW32\_CDECL pthread\_timechange\_handler\_np(void \*);

#endif /\*PTW32\_LEVEL >= PTW32\_LEVEL\_MAX - 1 \*/

#if PTW32\_LEVEL >= PTW32\_LEVEL\_MAX

/\*

\* Returns the Win32 HANDLE for the POSIX thread.

\*/

PTW32\_DLLPORT HANDLE PTW32\_CDECL pthread\_getw32threadhandle\_np(pthread\_t thread);

/\*

\* Returns the win32 thread ID for POSIX thread.

\*/

PTW32\_DLLPORT DWORD PTW32\_CDECL pthread\_getw32threadid\_np (pthread\_t thread);

/\*

\* Protected Methods

\*

\* This function blocks until the given WIN32 handle

\* is signaled or pthread\_cancel had been called.

\* This function allows the caller to hook into the

\* PThreads cancel mechanism. It is implemented using

\*

\* WaitForMultipleObjects

\*

\* on 'waitHandle' and a manually reset WIN32 Event

\* used to implement pthread\_cancel. The 'timeout'

\* argument to TimedWait is simply passed to

\* WaitForMultipleObjects.

\*/

PTW32\_DLLPORT int PTW32\_CDECL pthreadCancelableWait (HANDLE waitHandle);

PTW32\_DLLPORT int PTW32\_CDECL pthreadCancelableTimedWait (HANDLE waitHandle,

DWORD timeout);

#endif /\* PTW32\_LEVEL >= PTW32\_LEVEL\_MAX \*/

/\*

\* Thread-Safe C Runtime Library Mappings.

\*/

#if !defined(\_UWIN)

# if defined(NEED\_ERRNO)

PTW32\_DLLPORT int \* PTW32\_CDECL \_errno( void );

# else

# if !defined(errno)

# if (defined(\_MT) || defined(\_DLL))

\_\_declspec(dllimport) extern int \* \_\_cdecl \_errno(void);

# define errno (\*\_errno())

# endif

# endif

# endif

#endif

/\*

\* Some compiler environments don't define some things.

\*/

#if defined(\_\_BORLANDC\_\_)

# define \_ftime ftime

# define \_timeb timeb

#endif

#if defined(\_\_cplusplus)

/\*

\* Internal exceptions

\*/

class ptw32\_exception {};

class ptw32\_exception\_cancel : public ptw32\_exception {};

class ptw32\_exception\_exit : public ptw32\_exception {};

#endif

#if PTW32\_LEVEL >= PTW32\_LEVEL\_MAX

/\* FIXME: This is only required if the library was built using SEH \*/

/\*

\* Get internal SEH tag

\*/

PTW32\_DLLPORT DWORD PTW32\_CDECL ptw32\_get\_exception\_services\_code(void);

#endif /\* PTW32\_LEVEL >= PTW32\_LEVEL\_MAX \*/

#if !defined(PTW32\_BUILD)

#if defined(\_\_CLEANUP\_SEH)

/\*

\* Redefine the SEH \_\_except keyword to ensure that applications

\* propagate our internal exceptions up to the library's internal handlers.

\*/

#define \_\_except( E ) \

\_\_except( ( GetExceptionCode() == ptw32\_get\_exception\_services\_code() ) \

? EXCEPTION\_CONTINUE\_SEARCH : ( E ) )

#endif /\* \_\_CLEANUP\_SEH \*/

#if defined(\_\_CLEANUP\_CXX)

/\*

\* Redefine the C++ catch keyword to ensure that applications

\* propagate our internal exceptions up to the library's internal handlers.

\*/

#if defined(\_MSC\_VER)

/\*

\* WARNING: Replace any 'catch( ... )' with 'PtW32CatchAll'

\* if you want Pthread-Win32 cancelation and pthread\_exit to work.

\*/

#if !defined(PtW32NoCatchWarn)

#pragma message("Specify \"/DPtW32NoCatchWarn\" compiler flag to skip this message.")

#pragma message("------------------------------------------------------------------")

#pragma message("When compiling applications with MSVC++ and C++ exception handling:")

#pragma message(" Replace any 'catch( ... )' in routines called from POSIX threads")

#pragma message(" with 'PtW32CatchAll' or 'CATCHALL' if you want POSIX thread")

#pragma message(" cancelation and pthread\_exit to work. For example:")

#pragma message("")

#pragma message(" #if defined(PtW32CatchAll)")

#pragma message(" PtW32CatchAll")

#pragma message(" #else")

#pragma message(" catch(...)")

#pragma message(" #endif")

#pragma message(" {")

#pragma message(" /\* Catchall block processing \*/")

#pragma message(" }")

#pragma message("------------------------------------------------------------------")

#endif

#define PtW32CatchAll \

catch( ptw32\_exception & ) { throw; } \

catch( ... )

#else /\* \_MSC\_VER \*/

#define catch( E ) \

catch( ptw32\_exception & ) { throw; } \

catch( E )

#endif /\* \_MSC\_VER \*/

#endif /\* \_\_CLEANUP\_CXX \*/

#endif /\* ! PTW32\_BUILD \*/

#if defined(\_\_cplusplus)

} /\* End of extern "C" \*/

#endif /\* \_\_cplusplus \*/

#if defined(PTW32\_\_HANDLE\_DEF)

# undef HANDLE

#endif

#if defined(PTW32\_\_DWORD\_DEF)

# undef DWORD

#endif

#undef PTW32\_LEVEL

#undef PTW32\_LEVEL\_MAX

#endif /\* ! RC\_INVOKED \*/

#endif /\* PTHREAD\_H \*/