thread.cm

```
/*
    ========================================================================

    Copyright (c) 2012-2015 Seppo Laakko
    http://sourceforge.net/projects/cmajor/

    Distributed under the GNU General Public License, version 3 (GPLv3).
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html
        )

    ========================================================================
    */

// Copyright (c) 1994
// Hewlett-Packard Company
// Copyright (c) 1996
// Silicon Graphics Computer Systems, Inc.
// Copyright (c) 2009 Alexander Stepanov and Paul McJones

using System;

namespace System.Threading
{
    public const int EXIT_THREADS_NOT_JOINED = 250;

    public class ThreadingException: Exception
    {
        public ThreadingException(const string& operation, const string&
            reason): base(operation + ": " + reason)
        {
        }
    }

    public delegate void ThreadFun(void* arg);

    internal class ThreadData
    {
        public nothrow ThreadData(ThreadFun start_, void* arg_): start(
            start_), arg(arg_)
        {
        }
        public ThreadFun start;
        public void* arg;
    }

    public nothrow void ThreadStart(void* arg)
    {
        allocate_thread_data(this_thread());
        ThreadData* threadData = cast<ThreadData*>(arg);
        try
        {
```

```
            ThreadFun start = threadData->start;
            void* arg = threadData->arg;
            start(arg);
        }
        catch (const Exception& ex)
        {
            try
            {
                Console.Error() << ex.ToString() << endl();
            }
            catch (const Exception& ex)
            {
            }
        }
        delete threadData;
        free_thread_data(this_thread());
    }

    public class Thread
    {
        public Thread(): handle(), joinable(false)
        {
        }
        public Thread(ThreadFun start, void* arg)
        {
            ThreadData* threadData = new ThreadData(start, arg);
            thread_fun outerStart = ThreadStart;
            int result = create_thread(&handle, outerStart, threadData);
            if (result != 0)
            {
                delete threadData;
                string reason = strerror(result);
                throw ThreadingException("could not start a thread",
                    reason);
            }
            joinable = true;
        }
        public Thread(Thread&& that): handle(that.handle), joinable(that.
            joinable)
        {
            that.handle = thread_t();
            that.joinable = false;
        }
        public void operator=(Thread&& that)
        {
            if (joinable)
            {
                Console.Error() << "exiting because thread not joined" <<
                    endl();
                exit(EXIT_THREADS_NOT_JOINED);
            }
            Swap(handle, that.handle);
            Swap(joinable, that.joinable);
```

```
        }
        public ~Thread()
        {
            if (joinable)
            {
                Console.Error() << "exiting because thread not joined" <<
                    endl();
                exit(EXIT_THREADS_NOT_JOINED);
            }
        }
        public thread_t Handle() const
        {
            return handle;
        }
        public bool Joinable() const
        {
            return joinable;
        }
        public void Join()
        {
            if (joinable)
            {
                int result = join_thread(handle, null);
                if (result != 0)
                {
                    string reason = strerror(result);
                    throw ThreadingException("could not join a thread",
                        reason);
                }
                joinable = false;
            }
        }
        public void Detach()
        {
            if (joinable)
            {
                int result = detach_thread(handle);
                if (result != 0)
                {
                    string reason = strerror(result);
                    throw ThreadingException("could not detach a thread",
                        reason);
                }
                joinable = false;
            }
        }
        private thread_t handle;
        private bool joinable;
    }

    public bool operator==(const Thread& t1, const Thread& t2)
    {
        return equal_thread(t1.Handle(), t2.Handle());
```

```
    }

    public void SleepFor(Duration d)
    {
        long secs = d.Rep() / 1000000000;
        int nanosecs = cast<int>(d.Rep() % 1000000000);
        int result = cmsleep(secs, nanosecs);
        if (result != 0)
        {
            string reason = strerror(result);
            throw ThreadingException("could not sleep", reason);
        }
    }

    public void SleepUntil(TimePoint tp)
    {
        Duration d = tp - Now();
        SleepFor(d);
    }
}
```