Listings for the Timing Class

```
/*
Timing.h

This class implements a simple interval timer to aid in measuring drawing performance.  It'll
measure either wall or CPU time spent within an interval delineated by a pair of messages to
the Timing object.  It's most useful in situations where you need to measure not only the time
spent within the process, but also the time spent in other processes, most notably the Window
Server.  CPU time includes process time, system time on behalf of the process, and Window
Server time on behalf of the process.  The results are most accurate if averaged over a number
of passes through the interval, and the Timing object will keep track of:  number of times
entered, cumulative elapsed time, and average elapsed time.

Use the +newWithTag: method to create a Timing object.

Use the -reset message to reset the Timing object before entering the timing interval for the
first time.

A timing interval is delineated by an -enter: message and a -leave message.  enter: takes a
single argument that specifies either WALLTIME or PSTIME.

Use the -summary: method to have the Timing object print out a summary to the stream that is
passed in as the argument to summary:.  Alternatively, the Timing object provides methods for
querying it for the appropriate values.
```

Here's an example of its use.

```
    - action4:sender
    {
        int i=100;
        id t4 = [Timing newWithTag:4];

        [t4 reset];
        [self lockFocus];
        while(i--){
            [t4 enter:PSTIME];
            [self drawCachedLines];
            [[self window] flushWindow];
            [t4 leave];
        }
        [self unlockFocus];
        [t4 summary:stream];
        [self addSummary];
        return self;
    }

*/

#import <objc/Object.h>
#include <sys/time.h>
#import <sys/resource.h>
#define PSTIME 0
#define WALLTIME 1

@interface Timing : Object
```

```
{
    struct timezone tzone;
    struct timeval realtime;
    struct rusage rtime;
    double synctime;
    int    stime;
    double cumWallTime;      /* cum. wall time app + server */
    double cumAppTime;       /* cum. app process + system time */
    double cumPSTime;        /* cum. Server time on behalf of the app */
    double avgWallTime;      /* (cum. wall time app + server)/
                                cumTimesEntered */
    double avgAppTime;       /* (cum. app process + system time)/
                                cumTimesEntered */
    double avgPSTime;        /* (cum. Server time on behalf of the
                                app)/cumTimesEntered */
    double tare;             /* used to account for ipc overhead */
    int    cumTimesEntered;  /* number of times timing interval entered
                                since last reset */
    int    tag;              /* identifies timer object */
    int    wallTime;         /* flag to specify whether wall or process
                                time is desired */
}

+newWithTag:(int) aTag;
    /* Creates a new timing object with tag = aTag */

-enter:(int)wt;
    /* Starts a timing interval measuring either elapsed wall time if
       wt ==WALLTIME or elapsed process time + system time + Server time
       if wt == PSTIME.  Sets the wallTime flag to be equal to wt. */
```

```
-wallEnter;
    /* Called by enter: if WALLTIME is desired.  You should call enter:
       rather than call this method directly. */

-psEnter;
    /* Called by enter: if PSTIME is desired.  You should call enter:
       rather than call this method directly. */

-wallLeave;
    /* Called by leave if WALLTIME was specified on the previous enter.
       You should call leave rather than call this method directly.
       Updates cumWallTime based on the elapsed time. */

-psLeave;
    /* Called by leave if PSTIME was specified on the previous enter.
       You should call leave rather than call this method directly.
       Updates cumPSTime and cumAppTime based on the elapsed time. */

-leave;
    /* Call leave to leave a timing interval.  Depending on whether
       WALLTIME or PSTIME was specified on the previous call to enter:,
       leave will invoke wallLeave or psLeave. */

-reset;
    /* Resets the values of cumWallTime, cumPSTime, cumAppTime,
       cumTimesEntered and other variables to 0 in preparation for
       measuring a series of timing intervals.  Should be called prior to
       running a timing test. */
```

```
-avgElapsedTime;
    /* Calculates averages.  Called automatically by summary:. */

-summary:(NXStream *)c;
    /* Writes out a summary to the stream pointed to by c.  Depending on
       the current value of wallTime will write out a summary for either
       wall time or ps time. */

-(double) cumWallTime;
    /* Returns cumWallTime if wallTime == WALLTIME, -1 otherwise. */

-(double) cumAppTime;
    /* Returns cumAppTime if wallTime == PSTIME, -1.0 otherwise.
       cumAppTime represents the cumulative time spent in the process and
       system calls made by the process.  It does not include time spent
       in the Server. */

-(double) cumPSTime;
    /* Returns cumPSTime if wallTime == PSTIME, -1.0 otherwise.
       cumPSTime represents the cumulative time spent in the Window
       Server on the behalf of the process. */

@end


/* Timing.m */

#import "Timing.h"
#import <stdio.h>
#import <streams/streams.h>
```

```
#import <dpsclient/wraps.h>
#import <appkit/graphics.h>

@implementation Timing

+newWithTag:(int) aTag
{
    self = [super new];
    tag = aTag;
    [self reset];
    return self;
}

-enter:(int)wt
{
    if(wallTime = (wt==WALLTIME))
        [self wallEnter];
    else
        [self psEnter];
    return self;
}

-wallEnter
{
    cumTimesEntered++;
    NXPing();
    gettimeofday(&realtime,&tzone);
    synctime = realtime.tv_sec + realtime.tv_usec/1000000.0;
    return self;
}
```

```
-tare
{
    struct timezone tzone1;
    struct timeval realtime1;
    struct timeval realtime2;
    NXPing();
    gettimeofday(&realtime1,&tzone1);
    NXPing();
    gettimeofday(&realtime2,&tzone1);
    tare = (-realtime1.tv_sec + realtime2.tv_sec)+
            (-realtime1.tv_usec+realtime2.tv_usec)/1000000.0;
    return self;
}

-psEnter
{
    cumTimesEntered++;
    PSusertime(&stime);
    getrusage(RUSAGE_SELF,&rtime);
    synctime = (rtime.ru_utime.tv_sec + rtime.ru_stime.tv_sec) +
                (rtime.ru_utime.tv_usec + rtime.ru_stime.tv_usec)/1000000.0;
    return self;
}

-wallLeave
{
    double eTime;
    NXPing();
    gettimeofday(&realtime,&tzone);
```

```
    eTime = (- synctime + realtime.tv_sec + realtime.tv_usec/1000000.0)
                -tare;
    cumWallTime += eTime;
    return self;
}

-psLeave
{
    int et;
    double appTime;
    double psTime;
    getrusage(RUSAGE_SELF,&rtime);
    PSusertime(&et);
    psTime = ((et-stime)/1000.0);
    cumPSTime += psTime;
    appTime  = ((rtime.ru_utime.tv_sec + rtime.ru_stime.tv_sec) +
                (rtime.ru_utime.tv_usec +
                rtime.ru_stime.tv_usec)/1000000.0) -synctime;
    cumAppTime += appTime;
    return self;
}

-leave
{
    if(wallTime)
        [self wallLeave];
    else
        [self psLeave];
    return self;
}
```

```
-reset
{
    cumAppTime = 0.0;
    cumPSTime = 0.0;
    cumWallTime = 0.0;
    cumTimesEntered = 0;
    return self;
}

-avgElapsedTime
{
    if(wallTime)
        avgWallTime = (cumWallTime/(double)cumTimesEntered);
    else{
        avgAppTime = (cumAppTime/(double)cumTimesEntered) ;
        avgPSTime = (cumPSTime/(double)cumTimesEntered);
    }
    return self;
}

-(double) cumWallTime
{
    if(wallTime ==WALLTIME)
        return cumWallTime;
    else
        return -1.0;
}

-(double) cumAppTime;
```

```
{
    if(wallTime ==PSTIME)
        return cumAppTime;
    else
        return -1.0;
}

-(double) cumPSTime;
{
    if(wallTime ==PSTIME)
        return cumPSTime;
    else
        return -1.0;
}

-summary:(NXStream *)c
{
    if(wallTime) {
        NXPrintf(c,"Timer %d : entered %d trials TotalWall Time  %lf\n",
                    tag, cumTimesEntered, cumWallTime);
    }
    else {
        NXPrintf(c,"Timer %d : %d trials App: %lf  Server: %lf
                    Percent Server: %lf Total: %lf\n\00",
                    tag, cumTimesEntered, cumAppTime, cumPSTime,
                    cumPSTime/(cumAppTime+cumPSTime),
                    cumAppTime+cumPSTime);
    }
    NXFlush(c);
    return self;
```

```
}

@end
```