PerfStats.mesa

PerfStats is a package used for gathering performance information, often in the context of a test program. It implements counters and stopwatch-like timers, and will print summary statistics to a stream.

Last edited by:

MBrown on August 26, 1982 9:12 pm

```
DIRECTORY
  IO USING [STREAM],
  Rope USING [ROPE];

PerfStats: CEDAR DEFINITIONS =
  BEGIN
  ROPE: TYPE = Rope.ROPE;
  STREAM: TYPE = IO.STREAM;

  Counter: TYPE = REF CounterObject;
  CounterObject: PRIVATE TYPE = RECORD [
    pName: ROPE, counter: LONG CARDINAL _ NULL, next: Counter];
      Defined here so we can use an inline for Inc.

  CreateCounter: PROC [name: ROPE] RETURNS [Counter];
  DuplicateName: ERROR;
    Raised on CreateCounter or CreateTimer if name matches an existing event's name.
  InitializeCounter: PROC [event: Counter];
  DestroyCounter: PROC [event: Counter];
  Inc: PROC [event: Counter] = INLINE {event.counter _ event.counter+1};

  Timer: TYPE = REF TimerObject;
  TimerObject: TYPE;

  CreateTimer: PROC [name: ROPE] RETURNS [Timer];
  InitializeTimer: PROC [event: Timer];
  DestroyTimer: PROC [event: Timer];
  Start: PROC [event: Timer];
  Stop: PROC [event: Timer];


  Initialize: PROC [];
    Initializes all events.
  Print: PROC [heading: ROPE _ NIL, oStream: STREAM, verbose: BOOL _ FALSE];
    Prints current state of stats module to putChar, then calls cleanup if not NIL; resets glitch
          count. If verbose then prints all events, otherwise only events with nonzero counts.
  END.
```

How to use

PerfStats events

PerfStats defines two types of "event": counter and timer. Each event has a ROPE name, supplied when the event is created (CreateCounter, CreateTimer) and used to identify the event in printed output. An *attempt to create an event with the same name as an existing event will* ERROR *DuplicateName.*

Counter events are used for logging the freqency with which a specific action or set of actions is performed. If e is a PerfStats.Counter, then e.Inc[] causes e's counter to be incremented by 1. e.Inc[] is very inexpensive to perform.

Timer events are used for logging the time required for a specific action or set of actions. If e is a PerfStats.Timer, then e.Start[] records the current time in a variable associated with e. Then when e.Stop[] is performed at some later time, it notes the difference between the current time and the time recorded by the previous e.Start[]. At present the average, maximum, and minimum times are maintained in the timer event. e.Start[] and e.Stop[] involve one procedure call each, and must read the processor clock and manipulate e's statistics; hence they are more expensive than e.Inc[]. On the Dorado, the timer resolution is 32 microseconds, and the cost of a Start - Stop pair is about 33 microseconds. On the Dolphin these times are 64 microseconds and 417 microseconds, respectively.

A call to PerfStats.Print prints the current state of all active events to an output IO.STREAM *called oStream. (Actually, unless its parameter verbose is* TRUE*, Print prints nothing for counters with no Inc calls and timers with no Start - Stop calls.) Print takes a* ROPE *called heading as a parameter, and prints it as part of the output. Print calls oStream.Flush when it is through sending characters to oStream. Print is the only way to get information out of PerfStats; there is no way to interrogate an event directly.*

When two calls to e.Start[] occur in succession with no intervening e.Stop[], the first e.Start[] is ignored; when two calls to e.Stop[] occur consecutively, the last is ignored. Each of these situations counts as a "glitch", and the number of glitches is reported for debugging purposes during Print. Glitches may occur even in programs whose calls to Start and Stop appear to match properly, due to a failure to call e.Stop[] during the unwind of a procedure activation that called e.Start[]. We consider it too cumbersome to provide a catch for UNWIND *simply to avoid glitches, so we recover from them with only a printed warning.*

A separate procedure, PerfStats.Initialize, is provided in order to reset all events (clears all counters, forgets all times, except that the starting time of a running timer is not forgotten). Individual events may also be initialized (e.InitializeCounter[], e.InitializeTimer[]).

An event is eliminated by calling a destroy procedure (e.DestroyCounter[], e.DestroyTimer[].) This removes the event from consideration by Print, and makes its storage reclaimable.

An example

To compile a program that uses the PerfStats package, import the PerfStats definitions module. To bind the program, use the implementation PerfStatsImpl and import IO*, Rope and System for its use.*

The following is the skeleton of a test program that uses the PerfStats package.

```
DIRECTORY
  PerfStats,
  ...
SampleTest: PROGRAM IMPORTS PerfStats, ... = BEGIN
  oStream: IO.STREAM = ... ;
  insertCounter: PerfStats.Counter = PerfStats.CreateCounter["calls to Insert"];
  lookupTimer: PerfStats.Timer = PerfStats.CreateTimer["calls to Lookup"];
  ...
  Insert: PROC [x: Widget] = {
    insertCounter.Inc[];
    ... };--Insert

  Lookup: PROC [k: WidgetKey] RETURNS [x: Widget] = {
    lookupTimer.Start[];
    ...
```

```
   lookupTimer .Stop[] };--Lookup
 ...
 Test1[]; --calls Insert and Lookup
 PerfStats.Print["Test 1", oStream]; PerfStats.Initialize[];
 Test2[]; --calls Insert and Lookup
 PerfStats.Print["Test 2", oStream];
 insertCounter .DestroyCounter[];
 lookupTimer .DestroyTimer[];
END .--SampleTest
```

Change Log

Created by MBrown on November 4, 1980 3:35 PM

By editing DBStats.

Changed by MBrown on November 7, 1980 9:31 AM

Add InitializeCounterEvent and InitializeTimerEvent.

Changed by MBrown on January 10, 1981 9:24 PM

Created Pilot/collectible storage version. Print now takes putChar and cleanup as parms. Renamed to be PerfStats; made type and proc names less verbose. Moved most comments to documentation file (PerfStats.bravo).

Changed by MBrown on January 11, 1981 4:46 PM

Added DuplicateName ERROR. Added verbose parm to Print.

Changed by MBrown on 18-Aug-81 18:28:40

CedarString -> Rope.

Changed by MBrown on 7-Dec-81 15:23:08

Use IOStream, ROPE.

Changed by MBrown on June 24, 1982 1:16 pm

IOStream -> IO, CEDAR.

Changed by MBrown on August 26, 1982 9:21 pm

Format interface with node structure, merge documentation back in.