



## **PPCInfoSampler**

*Jim Gochee and Kevin Looney*

*March 26, 1996*

*PPCInfoSampler is a tool that samples the nanokernel to acquire registered counts of various OS events. These include MixedMode switches, amounts of time in native and emulated code, misalignment exceptions, and interrupts. This information can be used to gain insights into the performance and behavior of the OS or an application.*

### **1. Introduction**

The PPCInfoSampler tool is a control panel that when activated will record information about the PowerPC nanokernel and emulator at any millisecond interval. The information that is recorded includes MixedMode switches, interrupts, misalignment exceptions, and page faults.

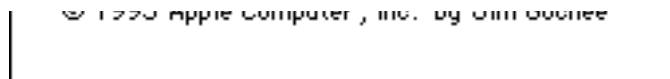
This document gives a description of the analysis and sample output of PPCInfoSampler.

### **2. Using PPCInfoSampler**

To use the PPCInfoSampler, you must first drop it into your System Folder (control panels folder) and reboot. The tool installs a code engine that sits in the system heap waiting for an action to occur.



There are three ways to activate the sampling mechanism. The first is to use a keyboard hot key. This key is currently set to “<option><command>z”. This key sequence will start the



sampler instantly at any time. Because the hot key mechanism is plugged directly into the keyboard interrupt handler, there is no delay from when the key is pressed and sampling begins. When you start the sampler, an 8 pixel line will flash in the upper left hand corner of your main screen. It will continue to flash for each sample that is recorded. To stop the sampler, use the hot key again.

The second way to start the sampler is to use the control panel interface. There are buttons for starting, stoping, and saving a sample.

**Figure 1.** PPCInfoSampler control panel interface

The third way to control the sampler is by calling the engine directly. The address of the engine's command handler function can be obtained through the Gestalt mechanism.

The exact call is:

```
Gestalt('pinf', &commandHandlerProc);
```

The command handler function is defined as:

```
typedef pascal long (*CommandFunctionType)
                        (short command, void *p1,
                        void *p2, void *p3, void *p4);
```

Valid commands to the engine are:

```
Status:                command = 1, p1 = (unsigned long *) returnStatus
Start Recording:       command = 5, p1 = (unsigned long) milliSampleInterval
Stop Recording:        command = 6
```

For Example, if you wanted to start the engine with a 10 millisecond sample interval, your code might look like:

```
CommandFunctionType    *theFunction;

Gestalt('pinf', &theFunction);

(theFunction)(5, 10, NULL, NULL, NULL);
```

### 3. How PPCInfoSampler works

The sampler uses an extended time manager task to sample at a given interval. The extended time manager provides guaranteed (or close to it) drift free timing service. When the interval expires and the sampler task is invoked, the sampler reads the current state of various nanokernel

context blocks. It writes these to a disk cache, which is composed of a set of two 64k buffers, each spooling asynchronously to disk when filled. The contexts blocks are then zeroed and the time manager is invoked to run the sampler task at the next interval.

#### 4. Text Output

The output saved from PPCInfoSampler is in tab-delimited format, and is best viewed from a spreadsheet such as Excel.

The following table gives a listing of all the table headings that PPCInfoSampler produces, and what those headings describe.

"Time Delta (millis)"	: Elapsed milliseconds since last sample
"Microseconds time"	: microseconds (calculated from timebase) since enable
"Timebase Ticks"	: A reading of the 64 bit timebase register
"MixedMode switches"	: Switches into PPC code (MixedMode switches)
"Data Page Faults"	: Page faults
"ExternalIntCount"	: External processor interrupts
"MisalignmentCount"	: Misaligned access that caused an exception
"FPUReloadCount"	: Reload of the fpu register state
"DecrementerIntCount"	: Interrupts caused by the PPC decrementer
"QuietWriteCount"	: Unknown
"HashTableCreateCount"	: Unknown
"HashTableDeleteCount"	: Unknown
"HashTableOverflowCount"	: Unknown
"EmulatedUnimpInstCount"	: Instructions that are emulated in nanokernel
"NCBPtrCacheMissCount"	: Unknown
"ExceptionPropagateCount"	: Unknown
"ExceptionForcedCount"	: Unknown
"Timebase Ticks 68k"	: Number of timebase ticks spent in 68k code.
This value seems to be updated only once per second	
"Timebase Ticks PPC"	: Number of timebase ticks spent in PPC code.
This value seems to be updated only once per second	
"Level n Int Ticks"	: Number of timebase ticks that expired per interrupt level
"Level n interrupts"	: Number of interrupts for each interrupt level

For each of the categories in the listing (with the exception of Microsecond Time), each measurement is per sample, and is not cumulative for the next quantum.

The following table is an excerpt from a report and is used here as an example to illustrate some calculations.

Microsecond Time	Timebase Ticks	Level 1 interrupts
100535	100747648	8
200517	100248192	14
300503	100238592	8
400510	100270336	8
500515	100266368	8
600503	100242688	8
700514	100277504	8
800516	100239360	10
900628	100368000	9
1000545	100171264	11
1100575	100284032	8
1200554	100241664	8
1300531	100227200	8
1400510	100242560	8
<b>TOTAL</b>		<b>124</b>

Table 1: Excerpt from PPCInfoSampler report

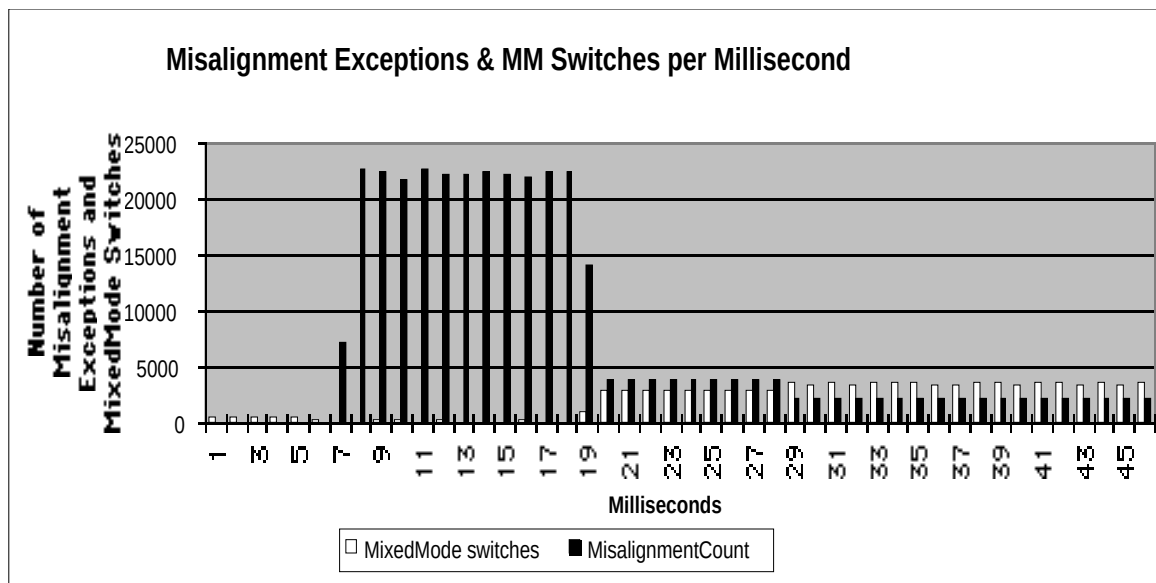
In order to determine the total number of Level 1 interrupts that occurred across the workload sample, we add up all numbers under the column “Level 1 Interrupts”. For this listing, the Total level 1 interrupts would be 124.

Total time is the accumulated value of microseconds found as the last entry in the column labeled “Microseconds Time”. For the given listing, that is 1400510 microseconds. Therefore, the average is:

$$\begin{aligned}
 \text{average level 1 interrupts per second} &= \text{Total Level 1 interrupts} / \\
 &\quad \text{Total sample time (seconds)} \\
 &= 124 \text{ (interrupts)} / 1.400510 \text{ (seconds)} \\
 &= 88.539175 \text{ level 1 interrupts per second}
 \end{aligned}$$

## 5. A Case Study

The following is a PPCInfoSampler report from the PEG Benchmark suite TB005 (ClarisWorks Recalc). Figure 1 shows a graphing of the first 50 milliseconds worth of misalignment exceptions and MixedMode switches.



Figure

e 1. Misalignment Exceptions and MM Switches per Millisecond in TB005

After 7 milliseconds, there is a dramatic increase in floating point misalignment exceptions. This seems to be prior to any rendering activity in the workload, so we are assuming that this is during the recalculation of data in the spreadsheet. We broke into Macsbug during this timeframe and found many floating point loads and stores that were relative to the stack, and at odd addresses. This suggested to us that there were many function calls to some function that took floating-point parameters, and that the applications compiler alignment settings were not correctly forcing floating point parameters into even addresses on the stack. Every access to such a parameter would force a misalignment exception to occur.

In this figure, we also see that MixedMode switching activity seems to increase at the 20 millisecond quantum. This is suspiciously the same quantum at which point the number of Misalignment exceptions decrease. When we examined these two activities in other workload samples, we could not discover any direct correlation. One possible speculation is that MixedMode activity could not occur while access exceptions were being handled - and these handlers required no MixedMode switching themselves.

## 6. Summary

From this brief analysis, we can see that PPCInfoSampler can be used as a low-overhead tool gather nanokernal statistics periodically. It is generally necessary to use some other tool (debugger, PowerTracer) to associate the behavior with what is happening from within the application to cause this behavior.

## **7. Availability**

Copies of the PPCInfoSampler tool are available on the file server:

'RD1/ATG-4SW:Wonderland:RabbitHole'.

in the folder:

'PEG Tools:PPCInfoSampler.