MacOS Instrumentation System User's Guide

Table of Contents

Introduction

The instrumentation system is designed to help you record information about running software. This allows you to expose its behavior and drive performance investigations.

It provides two services to client software. The first allows a running program to define a named event. Whenever the program determines that the event has occurred, it informs the instrumentation system. The instrumentation system creates an event record, including the event identifier and its timestamp, and writes it out to permanent storage – usually to disk. These event records are called “traces.”

The second service allows a client to create a unique, named container into which it can drop numbers of interest. There are a variety of different types of containers, each of which is suitable for recording a different type of number – such as a count, a transient value, a value within a range, and so on. The instrumentation system will periodically sample the containers and write their contents out to permanent storage. This data is called “statistics.”

When executing PowerPC code, these services are accessed from a shared library that your software links against. When executing 68K code, your software calls a statically-linked 68K library which calls the PowerPC implementation through the Mixed Mode Manager.

Using a program that can display instrumentation data, such as the Instrumentation Viewer, you can examine the trace and the statistics data – either after the program has completed its execution, or as it is running – to learn where it is spending its time and whether it is behaving as expected.

Typically, traces are used to track a program’s flow of execution. By recording traces from within a function, you can learn how often it is being called. By placing traces at its entry and exit points, and at those of the functions it calls, you can find out how long it takes to execute and where it is spending its time.

Statistics are good for recording user-defined counts or other measurements. They can be used to track buffer sizes, count cache hits and misses, and profile input parameters to functions. You can also use them to record your own timings by using TickCount() or Microseconds() to compute elapsed time and writing the result into a statistic.

This version of the instrumentation system operates under MacOS 7.x, and requires a PowerPC Macintosh with at least 1MB of free memory. It consists of two interface files, two shared libraries, one 68K library, and seven applications.

The System Components

The Instrumentation System must perform several operations: it must accept instrumentation data from client code, ensure that it is written out to disk, and provide an interface that allows display clients to access it.

These tasks are carried out by the following components:

The Instrumentation Library

The Instrumentation Library is a shared library that contains a public and a private interface. The public interface, specified by the “Instrumentation.h” file, allows client code to define instrumentation points and supply the data to be recorded. The Instrumentation Library holds this information in temporary buffers.

The private interface is provided to the spoolers and the OneShot Collector; it allows them to read the information out of these buffers and write it to disk.

The Spoolers

There are three spoolers: the ClassSpooler, the TraceSpooler, and the StatSpooler. Each retrieves a particular type of data from the Instrumentation Library and writes it to disk. The ClassSpooler records the various instrumentation points as they are defined - their names, their types, and their position in the class tree. The TraceSpooler records trace event data, and the StatSpooler records statistics data.

Each spooler creates its own file(s) to hold its data. This simplifies the file formats and reduces the chance that a system crash will leave a file in an inconsistent state. These files are always created inside the folder called “Instrumentation *f*” on the root level of your boot volume. Existing files with the same names are replaced.

The spoolers are faceless background applications. They rely on receiving null events at WaitNextEvent-time to allow them to collect their data.

The Spooler Packager

While the spoolers are running they produce separate files. When you are using the Instrumentation Viewer to view data “live,” the data comes from these files. However, they may also be combined into a single Instrumentation Data file. This is useful for saving a

“snapshot” of instrumentation data so that it may be analyzed or compared with another run at a later time.

When you run the Spooler Packager, it looks in the folder it is running from for the spoolers’ files. Any that it finds are merged into a single new file, called “Instrumentation Data.” If no one else is using the original files, they are deleted. You can run the Spooler Packager at any time; a valid file will be produced even if the original files are still in use by the spoolers.

The OneShot Collector

Like the spoolers, the OneShot Collector is used to record instrumentation data. Rather than reading it from the Instrumentation Library periodically, however, it collects all the available information in a single pass and then quits immediately. It produces the Instrumentation Data file directly: the file contains information about all the classes that were enabled when the collector was run, all the trace event records that were waiting in the Instrumentation Library’s buffer, and a single sample of the enabled statistics classes.

Note that the buffers maintained by the Instrumentation Library are released once all programs that link against it have quit. If you run and then quit an application that produces instrumentation data, running the OneShot Collector will create an empty data file unless another client of the Instrumentation Library – such as the Instrumentation Viewer – is still running to keep the library active.

The InstrumentationBackend

The InstrumentationBackend is a shared library that provides a display client with an interface to instrumentation data. The InstrumentationBackend that is provided with this version of the Instrumentation System can be used to read data “live” from the spooler files as it appears there, or from an instrumentation data file.

(Using a separate shared library to read the instrumentation data allows display clients to remain independent of the instrumentation system’s data file format.)

The Instrumentation Viewer

The Instrumentation Viewer is the primary mechanism for displaying instrumentation data. It uses the InstrumentationBackend library to gain access to the data files, and to “live” data as it is produced by the Spoolers. It displays this data inside viewer documents configured with plug-in viewers, and can also export the data as text.

Detailed documentation of the Instrumentation Viewer and its features may be found in the *Instrumentation Viewer User's Guide*.

Installing the Instrumentation System

The first step is to ensure that any existing instrumentation in your system is inactive. Quit any active spoolers using the "Quit Spoolers" application, shut down any software that is generating instrumentation data, and quit the Instrumentation Viewer if it is running. If you have never installed the instrumentation system before, you may skip this step.

Next, place the InstrumentationLib located in the folder "Contents -> Extensions" into the Extensions folder. This ensures that both the target application and the various spoolers will be able to find and link against it when they start up. The Instrumentation Viewer will also link against it to provide a user interface for enabling and disabling instrumentation points.

You should put the InstrumentationLib stub library , located in the "Interfaces & Libraries:Shared Libraries" folder, where it may be easily included in your code's project or makefile.

Drag the "Instrumentation *f*" folder from the SDK distribution folder to the root level of your boot volume. It contains the spoolers, the Spooler Packager, Quit Spoolers, and the OneShot Collector. It is also where the instrumentation data files that are produced by the spoolers are created.

If you wish, you can place aliases of the spoolers in the Startup Items folder of your System Folder. This will ensure that the spoolers are run whenever you reboot your Macintosh.

Finally, drag the "Viewer" folder to any convenient place on a local volume. It contains the Instrumentation Viewer application and the InstrumentationBackend.

(It is not necessary to reboot the the system after installing instrumentation. However, Quit Spoolers requires AppleScript in order to operate. You may wish to install AppleScript if you have not already done so.)

If you intend to use MrPlus to automatically instrument Code Fragments, you will have to install it as well. See the section on “Using MrPlus to instrument Code Fragments,” below.

The SDK also includes documentation and sample code in the “Documentation” folder. A demonstration application that produces instrumentation is located in the “Test Target” folder.

Using the Instrumentation System to collect data

Once the instrumentation system has been installed, you can begin collecting and examining instrumentation data.

The first step is to place calls into your code that will call the InstrumentationLib to record instrumentation data. Read the *Instrumentation Programmer’s Guide* to learn how to instrument your code manually. Alternately, you can use MrPlus to instrument an existing code fragment. See the section on “Using MrPlus to instrument Code Fragments,” below.

You must decide how you wish to collect the instrumentation data: using the spoolers or the OneShot Collector. This will likely depend upon how much data you plan to generate. If your code will log more traces than the RAM trace buffer can hold, then the spoolers may be preferable because they empty the buffer at WaitNextEvent-time. The spoolers also take regular samples of the enabled statistics, and allow the Instrumentation Viewer to display “live data.”

On the other hand, the OneShot Collector does not cause disk activity while you are making time-critical measurements, and is easier to use. You may find yourself using the spoolers for one application and the OneShot Collector for another.

If you wish to use the spoolers to collect the instrumentation data, launch them now. The easiest way to do this is to select all three in the Finder and choose “Open.” The spoolers will collect any data passed to the InstrumentationLib and write it to disk.

If another application – such as the Instrumentation Viewer – still has a file open from a previous run of a spooler, the spooler will display an alert reporting that it could not start up. Quit any applications using its file and launch the spooler again.

Once the spoolers are running, they will periodically collect data from the Instrumentation

Library and write it to disk. The Class Spooler writes out instrumentation class node descriptor data whenever a new class is created enabled, and whenever a class that was created disabled is enabled. The Statistics Spooler will write out the current values of the enabled statistics nodes every sample period; the default sample period is one second. The Trace Spooler will write out as many traces as it can, every chance it gets. All spoolers poll the Instrumentation Library at WaitNextEvent-time.

At this point, you can run your instrumented code. Any calls to the Instrumentation Library will cause instrumentation data to appear in the spooler files. If you wish, you can run the Instrumentation Viewer to display this data "live" as it it is written to the spooler files. (There is a slight time lag due to the recording and viewing overhead.)

If the Instrumentation Viewer is viewing data live, class nodes will appear inside its Instrumentation Tree window as they are created. In order to display the data inside these nodes, you have to configure a viewer document with appropriate viewers and associate the nodes with the viewers. See the *Instrumentation Viewer User's Guide* for details.

When you quit the applications that are generating instrumentation data, no more traces will be recorded. However, any active statistics nodes will continue to be sampled and written to disk. This is because instrumentation points are system-global; the Instrumentation System has no way of knowing whether they are still being updated.

As background applications, the spoolers will shut themselves down in response to a Quit Apple Event. You can use "Quit Spoolers" to send quit events to all three spoolers. (This requires that AppleScript is installed.)

Once the data you are interested in has been collected by the spoolers, you may run the Spooler Packager to produce the Instrumentation Data file. This file can be opened from within the Instrumentation Viewer.

If you wish to use the OneShot Collector to record the instrumentation data, you must ensure that the spoolers are not running. Run your instrumented code so that the data is collected in the temporary buffers of the Instrumentation Library, then run the OneShot Collector to transfer the data to the Instrumentation Data file.

If too many traces are logged to the Instrumentation Library before the OneShot Collector is run, older trace event records will be lost as they are overwritten by newer ones in the circular buffer. See the section on "Setting Instrumentation System Preferences," below, to learn how to increase the circular buffer size.

You can determine if any traces were lost by in an instrumentation data file by opening it

with the Instrumentation Viewer and creating a Viewer Summary report.

Note that the Instrumentation Viewer can only view the data “live” if the spoolers are running.

Setting Instrumentation System Preferences

Whenever you run the spoolers or the OneShot Collector, preferences are read from a file called “Instrumentation System Prefs” in the Preferences folder of your System folder. If it does not exist, it is created. It may be opened and reconfigured using ResEdit.

At this time, there are three editable settings: how often the Statistics Spooler samples the enabled statistics nodes, how much memory will be allocated to the circular trace buffer, and how much memory will be allocated to the class information heap.

The sample period is expressed as a 64-bit number of nanoseconds. The default is one second.

The trace buffer size determines how many kilobytes of memory will be allocated to hold traces waiting to be recorded to disk. The default is 256K. Each trace record requires 32 bytes of memory, so the default buffer can hold about 8000 trace points before earlier trace points are overwritten. If more traces than this are likely to be logged before the buffer is emptied by the Trace Spooler or the OneShot Collector , you should raise this limit. (Users with a lot of RAM can set this limit to 8MB or higher to ensure that no traces are lost.)

The class information heap is used to hold instrumentation class definitions. If you are getting an out-of-memory error when you try to create a class, you should increase the heap size. The default is 64K.

Using the Instrumentation Startup file

When the InstrumentationLib is initialized, it looks inside the Preferences folder for a text file called Instrumentation Startup. If the file is found, it is opened and its contents are read. The Instrumentation Library looks for lines beginning with an “E” or a “D” followed by a single tab character and then a colon-delimited full path name, terminated by a carriage return. All other lines are ignored.

The Instrumentation Library will pre-set the enabled state of the specified paths: nodes

marked with an “E” are preset to enabled; those marked with a “D” are preset to disabled. If client software subsequently creates one of these nodes, the enabled state specified in the creation call is overridden by the preset state. This can be used to automatically enable or disable instrumentation without changing the instrumented executable.

Using MrPlus to instrument Code Fragments

MrPlus is a post-link code fragment processing tool. It can read a PEF container holding a PowerPC Code Fragment and produce a modified version which will log trace points whenever it makes a cross-TOC call. It can also modify a shared library so that its exported functions are instrumented with trace points.These features allows you to quickly instrument a body of code without having to recompile it.

The modified fragment can be used in place of the original; it will behave identically to the original fragment. The only difference is that trace events will be written to the Instrumentation Library when it is invoked.

MrPlus works by placing calls to the ProfileLib shared library into the modified binary. The first time the routine is called, ProfileLib creates a new trace node for it, with the path FragmentName:RoutineName. It then logs a start trace to it, calls through to the routine, and logs an end trace before returning.

This feature can be used to determine which operating system calls an application makes, and how much time is spent inside of them.

The “Test target” folder contains a small application called InstTest, which can run a few simple instrumentation tests. Its imports have been instrumented with MrPlus, so that every system call it makes generates trace events.

Installing MrPlus

MrPlus is an MPW tool; it may be installed anywhere in the MPW command path.

Software that has been instrumented with MrPlus must dynamically link to the ProfileLib shared library, so ProfileLib should be placed in the Extensions folder or alongside the

instrumented executable. ProfileLib is part of the MrPlus 1.0 release.

Using MrPlus to Instrument Imports

Suppose that you have an application called “foo” containing a code fragment whose name is “fooFrag”. In order to instrument every cross-TOC call that fooFrag makes, use the following MPW command:

MrPlus foo -instrument imports -member fooFrag

This will create a second application called “foo.prof” with trace instrumentation calls embedded in its cross-TOC glue code.

You can instrument specific routines by creating an “include file.” In order to instrument only the imported routines “bar” and “baz,” create a tab-delimited text file that contains the following:

```
fooFrag	bar
fooFrag	baz
```

If the name of this file were “foo.inc” the MrPlus syntax would be:

MrPlus foo -instrument imports -member fooFrag -include foo.inc

You can also instrument all imports *except* for baz and bar by using the exclude option:

MrPlus foo -instrument imports -member fooFrag -exclude foo.inc

Using MrPlus to Instrument Exports

In order to instrument the exported routines of a CFM shared library called “foo.lib” containing a fragment whose name is “fooFrag”, use the following MPW command:

MrPlus foo.lib -instrument exports -member fooFrag

You can instrument a subset of the routines by using include files and exclude files. The command syntax and file format are identical to -instrument imports.