



Data Monitoring with NightStar LX™

*Jeff Hollensen and Todd Allen
Software Engineers
Concurrent Computer Corporation*

NightStar LX provides a suite of integrated development and debug tools specifically designed for use with complex and time-critical Linux® applications. NightStar LX's unique debugging methods allow programmers to debug and analyze complex applications with minimal intrusion, preserving the real execution behavior of the applications. NightStar LX includes NightView™, a multi-process and multi-thread debugger, NightTrace™, a graphical event analyzer, NightProbe™, a data monitor and recorder, and NightTune™, a process and system analysis tool. The information within this paper presents capabilities of NightView, NightTrace and NightProbe.

Probing

The NightStar LX tool suite provides a number of capabilities collectively called “probing” that allow tools and programs to obtain the value of variables in executing user applications. The probed data can be recorded for subsequent playback or can be viewed or consumed in real time in a variety of ways. Probing differs fundamentally from standard process-level debugging. Standard process-level debugging involves context switching between the user application and the debugging program; values of variables typically are obtained while the user application is stopped. Probing obtains data values as the application is running with minimal intrusion and with minimal overhead.

Synchronous vs. Asynchronous

A variety of methods are available for probing data but they fall into two broad categories: synchronous and asynchronous probing.

Synchronous probing implies coordination with the user process. A probe may be defined to occur at a specific location in the user's application specified by source file and line number or by instruction address. Alternatively, a probe may be defined to occur at times specified by the user.

The synchronous method allows the probed data to be consistent with respect to specific application execution. For example, consider the following code fragment:

```
1   int calc (int n) {
2       x += n;
3       y += n/3;
4       return x+y;
5   }
```

If the probe data set includes x and y , then defining a probe at line 4 ensures that the values returned for x and y reflect data from the same call to *calc*.

The asynchronous method does not provide that same level of consistency. Probes happen at arbitrary times which are not necessarily synchronized with the user application. In the example above, if x and y were sampled just before line 3 is executed, then the values of x and y would be from different invocations of the function *calc*. Asynchronous probing occurs in any of the following ways:

- on-demand, when the user clicks a button to request a data sample
- at user-defined frequencies based on the system clock, which typically is not synchronized with a user application
- at times determined by another user-defined application, which may not be synchronized with the application being probed

What Can Be Probed?

Both the synchronous and asynchronous methods are capable of monitoring static data values. Any program object with a static address and static size and shape may be monitored using this technique. The only requirement on the user application is that it be built with the “debug” compilation option, typically **-g** for most compilers. Candidate variables for probing include variables with the following types:

- integer
- floating point

- enumeration
- fixed point
- complex
- pointer

In addition to variables, the following objects are candidates for probing, assuming their types permit it:

- structure components
- array components
- class members

These variables and objects are called *static data candidates*.

In addition to static variables and components, the following can be probed with the synchronous probe method:

- stack variables
- dynamically-allocated objects
- components or members thereof
- arbitrary expressions, possibly involving dynamically shaped and sized objects, pointer indirection, and even function calls

These variables, objects, and expressions are called *unrestricted data candidates*.

How Probing Works

There are two techniques used to probe data values.

The *direct-read* technique can be used for *static data candidates*, and utilizes the Linux */proc* filesystem and *mmap(2)* system service which allow applications to map independent processes' data pages into their own address space. Once the probing program has used *mmap* to share the target application's physical pages, the probing program executes only a memory read to obtain the value of the variable of the probed program. This technique also provides for modification of user variables as discussed below. The level of intrusion incurred with this technique is almost nonexistent. It is simply the overhead involved with an independent process issuing a memory fetch and any possible memory bus contention due to the shared nature of the underlying physical memory location. Combined with Remote Prob-

ing, this method provides an efficient and non-intrusive technique for gathering data.

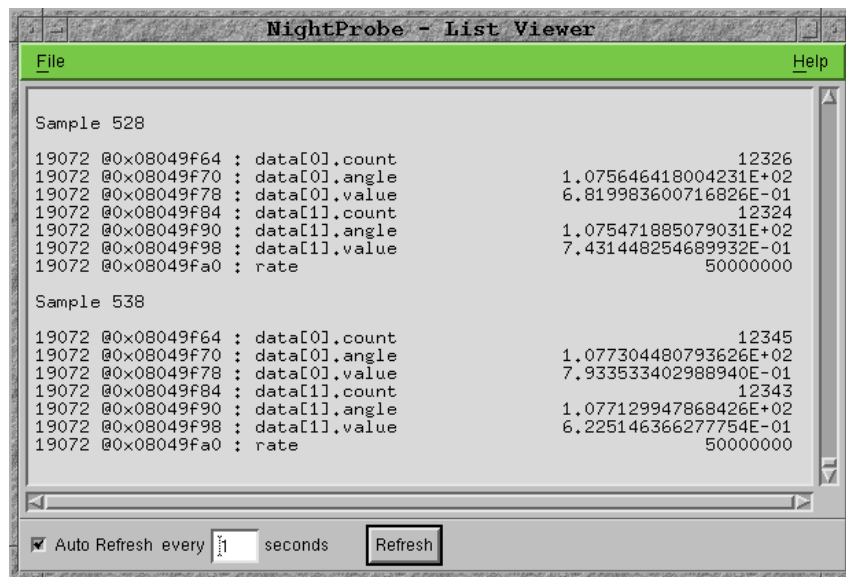
NightStar LX's *hot-patch* technique can be used for *unrestricted data candidates*. It works by inserting new code into an executing process without stopping the process. This is done through modification of the executable program's instruction pages in memory, by inserting a trap instruction. This trap efficiently passes control to a new location which evaluates the user-specified variables or expressions and then returns to execute the original instruction. The patched code either passes the data into shared memory or communicates with collection daemons. The trap transfers control without context switching to the probing tool, and the patched code typically is very small and runs at full application speed. Thus the intrusion on the process is minimal, although the *hot-patch*'s effects can be detected by precise execution monitoring. For example, a typical patch which logs one word-sized data item includes the overhead associated with a trap, a typical function call, and then a memory fetch and store required to pass the data value into shared memory. The only drawback of the *hot-patch* method is that, initially, the user's process must be stopped in order to prepare the process for subsequent *hot-patches*. Once that setup is complete, the process can continue unhindered and subsequent *hot-patches* can be inserted without stopping the user process.

Viewing The Data

Probed data can be recorded to a file for subsequent review using the NightStar LX tool suite or can be decoded using the NightProbe Datastream Application Programming Interface. Alternatively, data can be viewed immediately in a variety of ways:

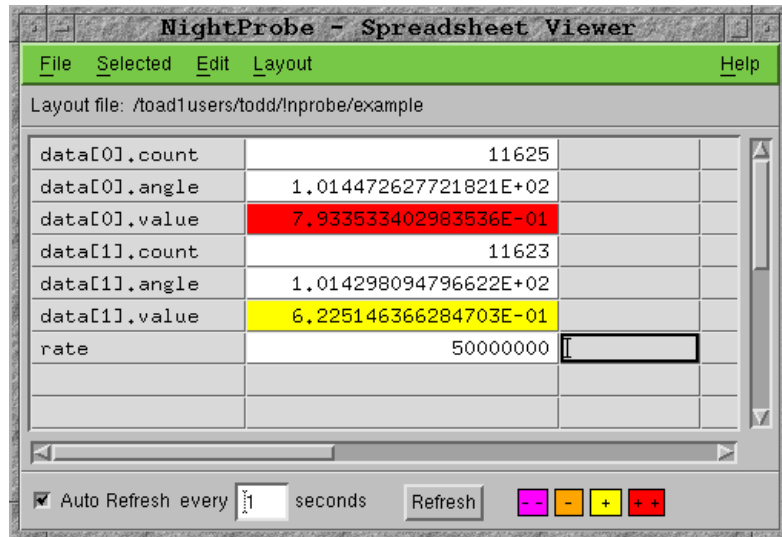
- NightProbe Scrolling Text Window
- NightProbe Spreadsheet Window
- NightView Monitor Window
- NightTrace Event Graphs
- NightTrace Data Graphs
- User-Customized Application

NightProbe Scrolling Text Window



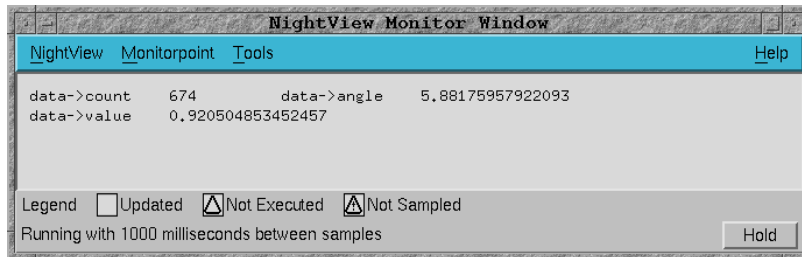
The figure above is an example of a NightProbe Scrolling Text window that samples asynchronously and displays the most recently probed data set in textual format, on-demand or at a user-defined rate. Each sample displayed includes the sample number, the process ID of the user application, and the variable's address, name, and value. The results of the displayed data can be saved to a text file. The Scrolling Text window can also be used to display previously-recorded data. The display rate is independent of the sampling rate. This is useful when probes are being executed at high speed for subsequent playback, yet provides for human interaction. The advantages of the Scrolling Text window mainly are its simplicity and quick setup. The Scrolling Text window supports *static data candidates* and utilizes *direct-read* probing.

NightProbe Spreadsheet Window



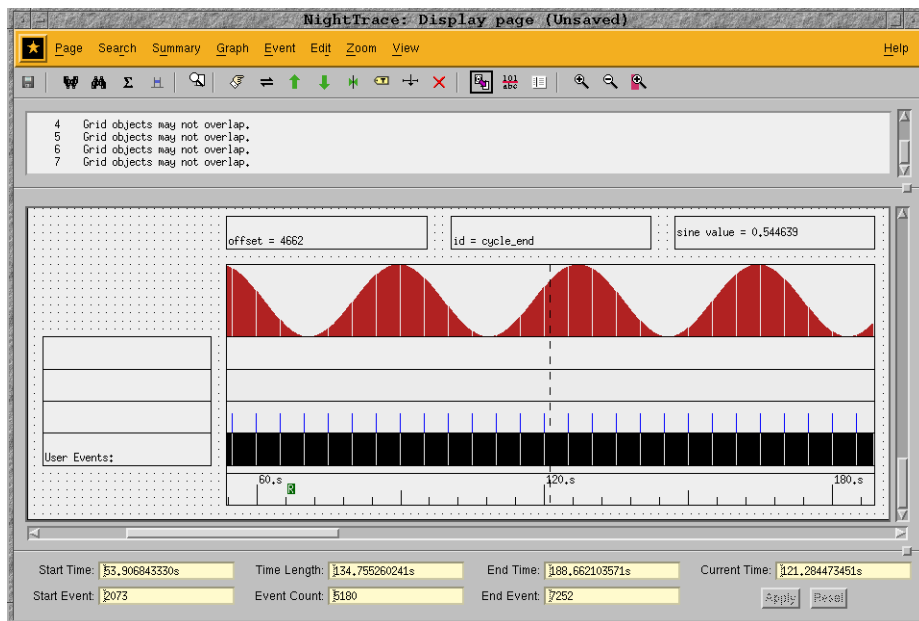
The figure above is an example of a NightProbe Spreadsheet window that allows users to configure the layout of data values and provides additional format and labelling customization, including colored warning and danger thresholds. The most recently probed data set is refreshed on-demand or at a user-defined rate. This technique supports *static data candidates* and utilizes *direct-read* probing. The Spreadsheet window also allows for program variables to be modified. By clicking on a displayed data value and typing in a new value, the corresponding variable is updated by writing the value directly into the application's memory page which is shared with the probing program.

NightView Monitor Window



The figure above is an example of a NightView Monitor window that displays the most recently probed data set in textual format. The data values are refreshed at a user-defined rate, but the probed data sets themselves are sampled synchronously at user-defined locations within the user application. This technique supports *unrestricted data candidates* and utilizes *hot-patch* probing.

NightTrace Graph Window



The figure above is an example of a NightTrace Graph window that includes data and event graphs. Each probed data item includes a code identifying the item, the value of the data item, and a precise timestamp indicating when the probe was executed. Data graphs vertically plot the values of individual data items from each probed data set on a horizontal graph. Event graphs (e.g. the bottom row labeled "User Events:") show a single vertical line for each occurrence of a probed data item on a horizontal graph. The horizontal dimension represents time. The time scale can be increased or decreased providing even sub-microsecond detail and each individual data item can be examined. Event graphs are useful when the *time* at which an event occurred is as important as the *value* associated with the event. The scaling of the data values in data graphs is configurable, as is the graph color. The graphs can scroll as data is being collected and the tool can pause and resume the collection stream at will. While the data graph does not have the full power of other graphical engines, the timestamp and streaming control provisions combined with synchronized event graphs make it powerful in its own right. This technique can support either *static data candidates* utilizing *direct-read* probing, or *unrestricted data candidates* utilizing *hot-patch* probing.

User-Customized Application

The NightProbe Datastream Application Program Interface allows user applications to decode previously recorded probe data sets for whatever customized processing is desired. NightProbe also supports a user-defined program launch that executes a user-defined program connecting the streaming probe data output to the program's *stdin*. Thus a user application can decode, manipulate, and transfer probed data sets on-the-fly, possibly passing them off to another display engine. The following code fragment illustrates a simplistic use of the API (with error checking omitted for brevity):

```
#include <nprobe.h>
#include <stdio.h>
#include <unistd.h>
main()
{
    np_header header;
    np_handle handle;
    FILE * file;
    np_item * i;
    char buffer[1024], * ptr;
    file = fopen("/tmp/probe_dump", "w+");
    np_open(STDIN_FILENO, &header, &handle);
    for (;;) {
        np_read(handle, &buffer[0]);
        for (i=header.items; i; i=i->link) {
            ptr = &buffer[0] + i->bit_offset * 8;
            fprintf(file, "name=%s bit_size=%d count=%d type=%d"
                    " value=%d\n",
                    i->name, i->bit_size, i->count, i->type, *(int*)ptr);
        }
    }
}
```

Probe Selection

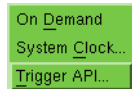
The data to be probed and the manner in which the data will be displayed can be selected in several ways.

NightProbe Utility

NightProbe allows you to locate executing processes easily and obtain lists of all data candidates that can be probed. Once the list of variables to be probed is selected, the probe and display methods are selected via menus.

NightProbe Probe Selection

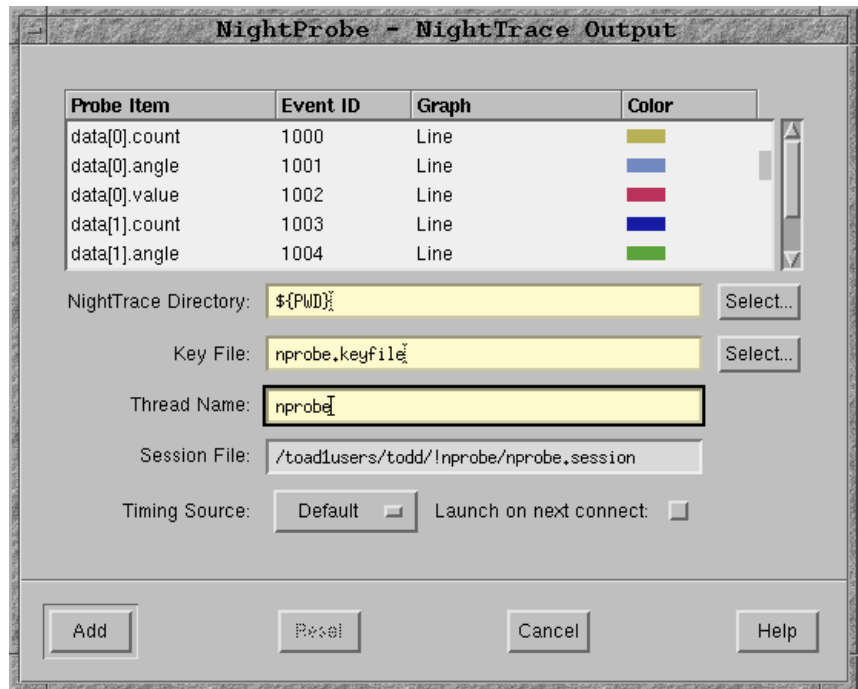
NightProbe can probe *static data candidates* using *direct-read* probing. It samples data asynchronously from the application by on-demand user action, or cyclically, based on a user-defined system clock frequency. Alternatively, NightProbe's data sampling can be synchronized with a user application using the NightProbe Trigger Application Programming Interface. Such an application may or may not be synchronized with the probed application.



NightProbe Display Selection

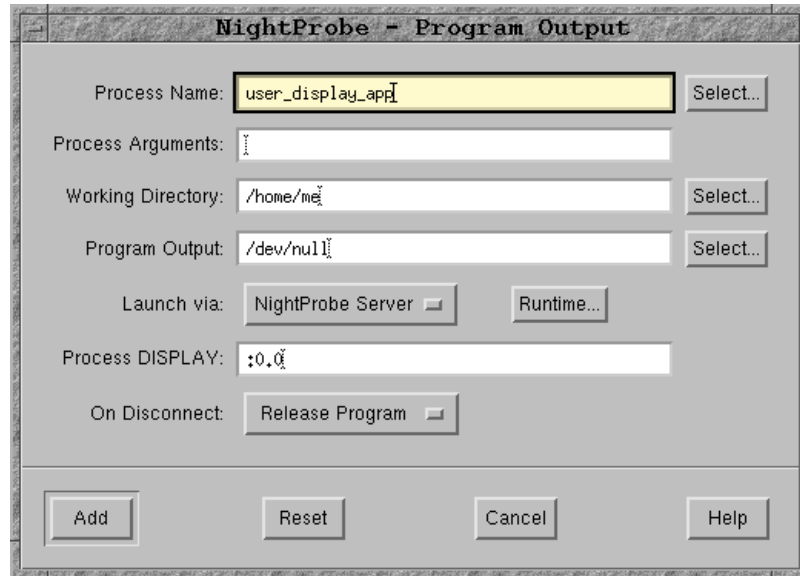
Probed data can be displayed in several ways. The figure below is a dialog that configures the Data Graph window. Individual items from the list of data candidates can be graphed and their color can be customized. Even more customization is available under the Night-Trace tool that is launched to control the data collection and analysis of the streaming data values.

- To File...
- To NightTrace...
- To List Window
- To Spreadsheet
- To Program...



User-Customized Display Selection

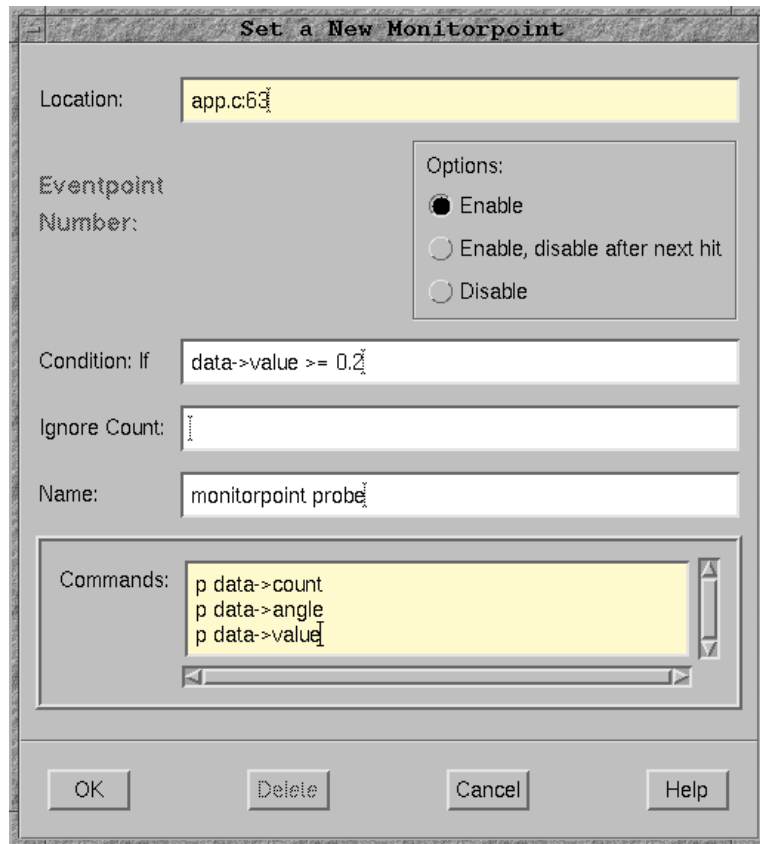
User-customized displays provide additional flexibility. The figure below is an example of the Program Output dialog configured to launch a user application.



NightView Probe Insertion

NightView allows you place synchronized probes at user-specified locations, by source file and line number or instruction address. These probes support NightView monitor windows, NightTrace event graphs, and NightView data graphs. They also allow modification of variables and other objects by specifying expressions with side-effects, such as assignments or function calls.

The figure below is the configuration dialog for a monitor window. It defines the program location and expressions to be probed. In addition, it includes *conditions* and *ignore counts* which can be configured to make probes occur selectively based on user-defined criteria. Conditions and ignore counts are evaluated by the patched code and therefore run at application speed, with no context switching to NightView. This technique supports *unrestricted data candidates* and uses *hot-patch* probes.



API Synchronous Probes

Alternatively, the user application can call the NightTrace API directly and log data values anywhere within the application. The NightTrace tool then can be launched dynamically at any time and begin to gather and display the data values. The probes include an identifying code, an optional data value, and are time-stamped automatically using a high-resolution, high-precision clock. This technique supports *unrestricted data candidates*.

Remote Probing

The NightStar tool suite supports remote probing, separating the setup, control, and display of data from the actual probing of the data. A small daemon process runs on the target system, and the remainder of each NightStar tool runs on a separate system. This reduces the intrusion on the target system significantly.

For example, a deployed application may not have symbolic debug information. However, a portable PC can be connected to the network to probe that application remotely, as long as the portable PC has a version of the application that includes symbols (before they were stripped out for production). Thus diagnostics and troubleshooting can be done after deployment with minimal intrusion on the system.

Summary

NightStar LX probing techniques provide solutions for many situations including debugging, data recording, diagnostics, performance analysis, and fault injection with minimal intrusion on performance and determinism.

Copyright

Copyright 2005 by Concurrent Computer Corporation. All rights reserved. This publication or any part thereof is intended for use with Concurrent products by Concurrent personnel, customers, and end-users. It may not be reproduced in any form without the written permission of the publisher.

The information contained in this document is believed to be correct at the time of publication. It is subject to change without notice. Concurrent makes no warranties, expressed or implied, concerning the information contained in this document.

NightStar LX, NightTrace, NightTune, NightProbe and NightView are trademarks of Concurrent Computer Corporation.

Linux is a registered trademark of Linus Torvalds.

Other products mentioned in this document are trademarks, registered trademarks, or trade names of the manufacturers or marketers of the product with which the marks or names are associated.

Rtlit-0032 0905

