NightProbe User's Guide

0890465-050
November 2003

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

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Computer Corporation, 2881 Gateway Drive, Pompano Beach, FL 33069-4324. Mark the envelope **"Attention: Publications Department**." This publication may not be reproduced for any other reason in any form without written permission of the publisher.

Printed in U. S. A.

Revision History:	Level:	Effective With:
Original Release -- November 1994	000	NightProbe 1.0
Current Release -- November 2003	050	NightProbe 2.6

Preface

Scope of Manual

This guide is designed to assist you in getting started with use of NightProbe, a real-time NightStar™ tool that provides a graphical user interface to the data recording services.

Structure of Manual

This manual consists of ten chapters and two appendixes. A brief description of the chapters and appendixes is presented as follows.

- Chapter 1 introduces you to the concepts and components of NightProbe, a real-time tool that is part of the NightStar development environment.
- Chapter 2 explains the procedures for beginning and ending a NightProbe session and explains how to get help.
- Chapter 3 introduces the components of NightProbe's Data Recording window, including the selection of timing sources and output viewers.
- Chapter 4 introduces the components of NightProbe's Target System Selection window.
- Chapter 5 introduces the components of NightProbe's Program Selection window.
- Chapter 6 introduces the components of NightProbe's Variable Browser window.
- Chapter 7 introduces the components of NightProbe's Variable Attributes window.
- Chapter 8 introduces the components of NightProbe's two output windows, the List Viewer window and the Spreadsheet Viewer window.
- Chapter 9 documents the NightProbe Application Programming Interface and provides sample programs to demonstrate usage of the data structures and functions in the API.
- Appendix A consists of tutorials for program creation and selection, for variable browsing, and for using the spreadsheet.
- Appendix B contains information about customizing the NightProbe graphical user interface.
- Appendix C provides an overview of the primary factors that need to be taken into account prior to installing and running NightProbe on Power-MAX OS™.

Syntax Notation

The following notation is used throughout this manual:

italic	Titles of books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italic*s.
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and system manual page references also appear in **`list bold`** type.
`list`	Operating system and program output such as prompts and messages and listings of files and programs appear in `list` type.
window	Keyboard sequences and window features such as button, field, and menu labels, and window titles appear in window type.
`[]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify such options or arguments.

Referenced Publications

The following publications are referenced in this document:

0890285	*Real-Time Documentation Set*
0890300	*X Window System™ User's Guide*
0890380	*OSF/Motif™ Documentation Set*
0890398	*NightTrace™ Manual*
0890429	*System Administration Volume 1*
0890430	*System Administration Volume 2*
0890423	*PowerMAX OS Programmer's Guide*
0890458	*NightSim™ User's Guide*
0891055	*Élan™ License Manager Release Notes*
0898004	*RedHawk Linux User's Guide*

Contents

Chapter 8 Using the Output Windows

Chapter 9 Using the NightProbe API

Appendix A Tutorials

Appendix B GUI Customization

Appendix C PowerMAX OS Requirements

Index

Illustrations

Tables

1
Overview

1
Overview

NightProbe is a graphical tool for real-time recording, viewing, and modifying program data within one or more executing programs without significantly affecting the execution of those programs. The source code of the target program does not need to be changed and recompiled in order to be monitored. Executing programs can be monitored and recorded without being stopped and restarted.

NightProbe can be used in a development environment as a tool for debugging, analysis, and prototyping, or in a production environment to create a “control panel” for program input and output.

Figure 1-1 shows a high-level view of the steps involved in using NightProbe.

Figure 1-1. Using NightProbe

Recording and Monitoring

Data recording refers to sampling memory locations in running programs and recording that data in real time. Memory locations may be identified by logical address or by specifying the name of a particular variable in the running program. NightProbe allows you to record data to file or shared memory in several possible formats, including a format that can be used as input to the NightTrace analysis tool.

Data monitoring refers to displaying the sampled data for visual inspection and, perhaps, modification. NightProbe provides several ways to organize the displayed information, including a flexible spreadsheet window. Figure 1-2 shows a data monitoring window using the spreadsheet viewer.

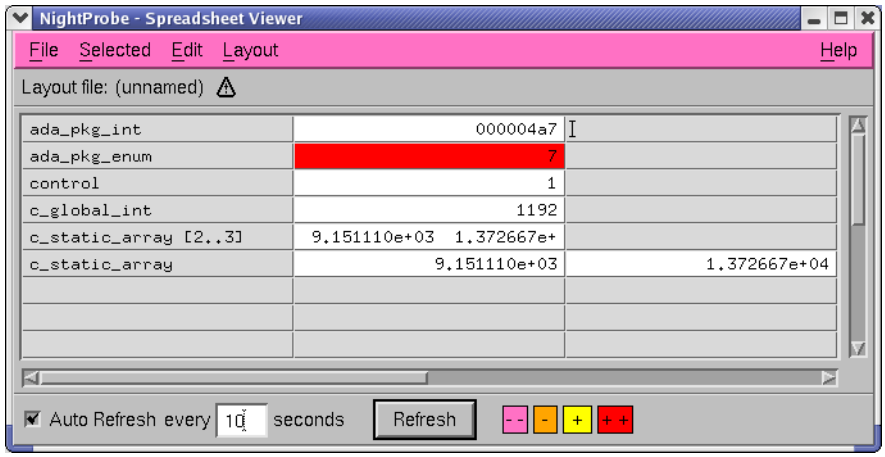

Figure 1-2. Data Monitoring Using the Spreadsheet Viewer

NightProbe can also save snapshots of the display at any time.

What You Can Watch

Any C or Fortran process on any processor can be sampled by NightProbe. On PowerMAX OS, NightProbe can also sample Ada processes on any processor.

You can monitor and record any static memory location in the program's address space, named or not.

NightProbe can be run on a different processor or system from the target program, which minimizes NightProbe's impact on the target program's performance. It also allows NightProbe to be used on a wide range of programs and applications.

Furthermore, NightProbe can probe executable programs which have been stripped of debug and symbol information, such as those in production level scenarios. In such cases, either a configuration file (created earlier by the user) must be supplied or a copy of the program containing the necessary debug and symbol information must be available for reading (not necessarily on the target system).

Selecting Data Locations

NightProbe allows data locations to be identified using logical addresses or by specifying the variable names that appear in the program source code. The data addresses and data types are then located by searching the symbol table in the executable program file. NightProbe can present lists of the static variables in programs and the user may select the variables of interest using a Motif interface.

Configuration files can be created and saved to retain variable selections and display layout, allowing for fast start-up.

How You Can Control the Sampling

Several timing sources are provided for controlling the sampling rate, including the frequency-based scheduler, the system clock, and interactive control.

The user may explicitly start, suspend, and stop sampling using the graphical user interface.

Overview of Data Sampling Procedures

NightProbe provides a graphical user interface to the several tasks you must do to collect or view sampled data from a set of running target programs. Basically, you must

- Decide what and when to sample and where to put the data sampling results, and communicate that configuration to NightProbe.
- Run your target programs.
- Connect the NightProbe sampler to the target programs.
- Begin and control the sampling session.
- Record or monitor the results.

Overview of Data Sampling Configuration

A *data sampling configuration* is a temporary or saved list of target program and memory addresses with information about the data type at that location, the name of the data item at that location, and how it should be displayed. NightProbe provides a graphical user interface to the symbol table and debugging information in program files to aid in the construction of the configuration. The main features of NightProbe's configuration construction facilities are:

- A variable browser which allows variables to be selected from a list of named memory locations in target programs.
- Selection mechanisms also exist for locating program files and processes.
- The configuration can be saved and restored for future use.
- The configuration can be created on a development system and used on a different target system.

The main data sampling configuration windows in the NightProbe graphical user interface and their primary functions are:

- The Data Recording window displays status information about the current data recording configuration and provides access to other windows. See Chapter 3, "Using the Data Recording Window" for more information.

- The Target System Selection window allows you to specify the name of the system on which the target programs are running as well as the login name of the user connecting to those programs. See Chapter 4, "Using the Target System Selection Window" for more information.
- The Program Selection window assists you in selecting programs that you want to monitor, record, or modify. See Chapter 5, "Using the Program Selection Window" for more information.
- The Variable Browser window allows you to select variables or memory locations by browsing the symbol tables. See Chapter 6, "Using the Variable Browser Window" for more information.
- The Variable Attributes window is used to select variables and assign attributes to them (for example, the default output format). See Chapter 7, "Using the Variable Attributes Window" for more information.

Overview of Data Sampling Control

Data sampling is the process of collecting the values of target memory locations in real time, and making the sampled data available to output or viewer processes that can record the data or display the data for interactive monitoring. The main features of NightProbe's data sampling capabilities are:

- A variety of timing sources, including the system clock and the frequency-based scheduler can be used to control the sampler.
- The graphical user interface provides controls for connecting a sampler to a configuration, starting, suspending, and resuming the sampler, and disconnecting the sampler.

First you must connect NightProbe to the target programs using the Probe control. Then you may start, suspend, and resume sampling using the Sampling controls.

The main Data Recording window contains the interface components for controlling the sampling session.

Overview of Data Recording

Data recording refers to the asynchronous, real-time logging of sampled data values, where "real time" implies a guaranteed short response time. The main features of NightProbe's data recording capabilities are:

- Program data can be recorded to output files or to shared memory in real time.
- Recorded data can be written to a file suitable for use with the NightTrace analysis tool.
- Recorded data can be written to a file and then translated into a form suitable for printing on a line printer.

The main Data Recording window contains the interface components for invoking the data recording capabilities.

Overview of Data Monitoring

Data monitoring consists of viewing and modifying the sampled data values as the target programs run. NightProbe has the ability to write to the program address space, so the values of the variables can be modified during execution. Developers can create system prototypes by using NightProbe to supply values for components that have not yet been implemented. Developers can also create control panels to monitor and control programs in production systems. The main features of NightProbe's data monitoring capabilities are:

- Data can be displayed in a spreadsheet window that can be configured by the user.
- Program data that is being monitored can be modified by the user simply by entering a replacement value at any time in the spreadsheet.
- Data can also be listed in a simple text window.

The main Data Recording window contains the interface components for invoking the data monitoring capabilities.

Two windows are used for monitoring sampled data:

- The List Viewer window is used to monitor sampled data as a text report, described in "Using the List Viewer Window" on page 8-1.
- The Spreadsheet Viewer window is used to monitor and modify sampled data using a flexible display layout, described in "Using the Spreadsheet Viewer Window" on page 8-5.

2
Getting Started

2
Getting Started

This chapter explains the following procedures:

- Using variable names
- Meeting target program requirements
- Handling performance issues
- Setting X Window System resources
- Working with NightProbe's user interface
- Setting environment variables
- Invoking NightProbe
- Getting help
- Using keys, accelerators, and mnemonics
- Exiting NightProbe
- Recording and monitoring procedures

Using Variable Names

Variable names may be used to identify memory addresses in C and Fortran (and on PowerMAX OS, Ada) programs. NightProbe accepts and displays variables with the following syntax.

Syntax

```
[/file/][scope.] ... name[(array_slice)]
0xaddress[:n]
```

Parameters

file	The source file name enclosed in slashes //.
scope	The name of the scope. Includes the names of enclosing functions, packages, or aggregates. Each one is separated from the next by a dot (.). (See "Aggregates" on page 2-2 for information about aggregates.)
name	The name of the variable. The variable may be either a scalar or an array.

array_slice	An index representing a single array element or an index range representing an array slice. *Array_slices* must be enclosed in either parentheses `()` or square brackets `[]`. (See "Array Slices" on page 2-2 for information about array slices.)
address	A memory address beginning with a number. If it begins with `0`, it is treated as an octal address. If it begins with `0x`, it is treated as a hexadecimal address.
n	An integer representing the size in bytes. It has a colon prefix.

Note that *name*, *name*(), and *name*[] all refer to the entire array.

For some examples using variables in NightProbe, see "Variable Browsing" on page A-2 and "Variable Browsing" on page A-14.

Aggregates

To NightProbe, C structures and unions and Ada records are *aggregates*; arrays are not. NightProbe treats aggregates as scopes and as symbols. The members of an aggregate are the components of the scope. Aggregate nesting is supported.

Array Slices

Array slices identify a single element or a range of elements in an array. You select one array element in a manner just like you would use in your program:

```
var (5)
```

Some programming languages use brackets instead of parentheses, as in

```
var [5]
```

NightProbe accepts either convention.

In some cases, it is appropriate to select a range of elements. These elements must be contiguous, and all must lie within the stated bounds of the original array declaration. You specify a range by providing the first and last items that you wish to select. The following syntaxes are all equivalent and may be used with programs of any language.

array_name `(` *first_item* `:` *last_item* `)`
array_name `[` *first_item* `:` *last_item* `]`
array_name `(` *first_item* `..` *last_item* `)`
array_name `[` *first_item* `..` *last_item* `]`

where:

array_name	The name of an array.
first_item	A valid array index, greater than or equal to the lower bound of the array and less than or equal to *last_item*.

last_item A valid array index, less than or equal to the upper bound of the array and greater than or equal to *first_item*.

For example, in Fortran the array declaration

```
integer*4    var (10)
```

declares an array of ten integers with indices 1 through 10. To specify an array slice containing the first five elements, you would use

```
var (1:5)
```

C programs use 0 as the lower bound of all arrays. The declaration

```
int  var [10];
```

also declares an array of ten integers, but with indices 0 through 9.

The equivalent array slice would be

```
var (0:4)
```

Of course, if you are a C programmer you would probably use brackets:

```
var [0:4]
```

and you might prefer the Ada range notation:

```
var [0..4]
```

These last three examples are all equivalent.

In C and Ada, the rightmost subscript of a multi-dimensional array changes most quickly. In Fortran, the leftmost subscript of a multi-dimensional array changes most quickly. Array slices must identify elements that are contiguous in memory. For example, for an 8 by 8 array:

C

Specify `var[1][2]` to refer to the memory location right after `var[1][1]`. The following array slice is valid: `var[3][1:5]`.

Fortran

Specify `var(2,1)` to refer to the memory location right after `var(1,1)`.The following array slice is valid: `var(1:5,3)`.

Ada

Specify `var(1,2)` to refer to the memory location right after `var(1,1)`. The following array slice is valid: `var(3,1..5)`.

Meeting Target Program Requirements

Any process on any processor can be a target program for data recording.

As stated before, variable names may be used to identify memory addresses in C and Fortran (and on PowerMAX OS, Ada) programs. If you wish to identify memory locations by variable name, the target program file must contain symbol table and debug information. Use the **`-g`** compiler and linker option to retain debug information, and do not use the **`-s`** linker option that strips symbol table information from the executable program file.

Any fixed (static) address in a program can be monitored and recorded. The following text lists eligible variables by language.

C

- Variables typed `static`
- Global variables declared outside all functions

Fortran

- Variables typed `static` or `save`
- Variables initialized in a `data` statement
- Variables placed in a `common` block

Ada

The following criteria are used to determine if an Ada data object is eligible for data monitoring/recording:

- The compilation unit containing the object must be a library-level package specification or body. Objects declared in nested packages inside a library-level package are also eligible.
- The object must not be declared in a generic or in the instantiation of a generic.
- The object must have a size and representation which is statically determined at compile time.
- The object may be declared in a library-level package marked with pragma `SHARED_PACKAGE`.

The following Ada data types are eligible for data monitoring/recording:

- Any integer, fixed-point or floating-point type or subtype.
- Any character, Boolean or enumeration type or subtype.
- Access types.

- Array and record types (for records with variant parts, only components that have a statically determined component offset are eligible).

NOTE

Task types and variables declared in Ada procedures or tasks, or objects in an access type's collection, are allocated dynamically, and are, therefore, ineligible for data monitoring/recording.

Currently NightProbe does not handle indirection, literal enumerations values, Fortran character types, or Fortran datapools. It does handle enumeration position numbers.

Handling Performance Issues

To prevent the execution of NightProbe from interfering with the execution of the real-time application, it is recommended that you assign NightProbe to the boot processor or any other processor on which real-time tasks are not running. You can assign NightProbe to a particular processor by invoking the **`run(1)`** command from the shell and specifying the **`-b`** *bias* option. Prior to using the **`run`** command for this purpose, you may wish to determine which processor on your system is the boot processor. You can do so by invoking the **`run`** command without specifying any arguments.

Furthermore, NightProbe provides the capability to probe programs running on a remote target system, offsetting much of the heavy GUI and symbol table processing to the host system.

Setting X Window System Resources

The user interface for the NightProbe application is based on OSF/Motif, and it runs in the environment of the X Window System. All X applications may be customized using X resources. Resources specify application attributes such as fonts, colors, screen layout, and button and label names.

NightProbe provides default values for its X resources. Each user may override any X resources with personal preferences, or a site may provide for different defaults.

Details for specifying X resource values are in **`X(1)`**, **`xrdb(1)`**, and Appendix B.

Working with the NightProbe User Interface

It is assumed that your X server has a three-button mouse. By default, mouse button 1 is the leftmost button, button 2 the middle button, and button 3 the rightmost button. You can reassign the functions associated with mouse buttons by using **xmodmap(1)**. If you do not have a three-button mouse, see your system administrator.

Before you start NightProbe, set your DISPLAY environment variable. See "Environment Variables" on page B-1 for details.

Some button labels and menu options may be disabled (dimmed) during a NightProbe session. This occurs when those buttons or menu options are not applicable to the current settings or displays.

Invoking NightProbe

The NightProbe tool is available on your system as **/usr/bin/nprobe**. The format for executing **nprobe** is as follows:

To get information about NightProbe, use

nprobe [-help] [-version]

or, to use NightProbe to record or monitor variable locations, use

nprobe [-autoupdate] [-record *output_file***]**
[-trace *output_file***][-sheet** *config_file***]**
[-list] [-*Xoption* **...] [***config_file***]**

or, to translate data files to text, use

nprobe -i *data_file*

Options are described as follows:

-help — This option allows you to display the usage information for **nprobe** and then exit. The X Window interface will not be started if you use this option.

-version — This option allows you to display the version and copyright information for **nprobe** and then exit. The X Window interface will not be started if you use this option.

-autoupdate — When using the variable browser, usually you must explicitly request that the Scope and Variable lists be updated. With this option, updates will happen automatically each time the scope changes. The same functionality is available with the Auto Update check box (see "Using the Program Scope Area" on page 6-3) and the autoUpdate resource.

-record *output_file* — Activate recording to file *output_file* when sampling begins.

-trace *output_file* — Activate recording in NightTrace format to trace-event file *output_file* when sampling begins. An event map will be written to *output_file***.evtmap**.

-sheet *config_file* — Activate a spreadsheet viewer using the layout configuration in file *config_file*.

-list — Activate a list viewer.

-i *data_file* — Translate a previously recorded data file from its internal format to printable ASCII text. The **-i** option must be followed by the name of the data file. The text translation will be written to standard output. No other options should be used with **-i**. The X Window interface will not be started when using the **-i** option.

-*Xoption* — You may also specify any standard X Toolkit command-line option. Such options include **-bg** *color* to set the color for the window background; **-fg** *color* to set the color to use for text or graphics; and **-xrm** *resourcestring* to set selected resources. For a complete list of these options, refer to the **X(1)** system manual page.

config_file — This argument allows you to specify the name of a file that contains data sampling configuration data. You may specify a full or relative path name. The file may be one that you have created by using **nprobe** or a text editor of your choice.

If you use **nprobe** to create this file, you open a Data Recording window, configure a data recording session, and then select the File Ì Save Config File As menu item. Procedures are fully explained in "Chapter 3, "Using the Data Recording Window".

You may invoke **nprobe** without specifying any options. Doing so allows you to display a blank Data Recording window. A blank Data Recording window is one that is not configured.

The steps for invoking NightProbe are as follows:

1. Log into your system.
2. Ensure that the value of your DISPLAY environment variable is set to the name of your X server. (Refer to "Environment Variables" on page B-1 for an explanation of the procedures for setting this variable.)
3. Type **nprobe** and any desired options after the system command prompt, and press the carriage return key.

Getting Help

Besides the *NightProbe User's Guide*, there are several sources of information on the operation of NightProbe. These include:

- the *NightProbe Release Notes*
- the **nprobe(1)** and **nprobe(9)** system manual pages
- the **-help** command line option (described in "Invoking NightProbe" on page 2-6)
- the Help menu on the menu bars of several of the NightProbe windows (see "Help" on page 3-21)
- the Help button on several of the NightProbe dialogs

  When you click on the Help button in one of these windows, the help information for that window is displayed.

Using Keys, Accelerators, and Mnemonics

NightProbe uses certain key combinations as shortcuts for displaying menus and selecting menu items. These key combinations are called *accelerators* and *mnemonics*. Each window has its own set of accelerators and mnemonics that are active only while the keyboard focus is in that window. However, the keyboard focus does not have to be in any particular field of the window to use accelerators and mnemonics. This manual shows the supplied mnemonics and accelerators associated with a menu or menu item. However, users can alter this behavior with resources. See "NightStar Resources" on page B-3 for details.

- Menus can be displayed with mnemonics.

  Menus can be displayed from the keyboard by typing <Alt>+*mnemonic*. Each of the main windows has a menu bar near the top of the window. The different menus are labeled. For example, the Data Recording window has a Timer menu. If you look at the Timer menu, you can see that the T is underlined. T is the mnemonic for the Timer menu. That means that, in addition to displaying the Timer menu by clicking on it with mouse button 1, you can also display it with <Alt>+t (hold down <Alt> and press t).

  If you decide you don't want to select any of the menu items, you can make the menu go away by typing <Esc> or by clicking somewhere else.

- Menu items can be selected with mnemonics.

  Once a menu is displayed, you can select a menu item by typing only the mnemonic for that item. The mnemonics for the menu items are underlined, just as the mnemonics for the menus are underlined. To select a menu item by using its mnemonic, just press the key.

- Menu functions can be invoked with accelerators.

  Some commonly used menu items have accelerator keys. The functions associated with these menu items can be invoked directly, without displaying the menu, by pressing the accelerator keys. The accelerator keys for a particular menu item are listed next to the item in the menu.

  The accelerator keys are often a combination of a control key plus a letter, such as Ctrl+O. To type Ctrl+O, hold down the control key and press o.

In addition to mnemonics and accelerators, there are also special keys used for navigation within and among windows and fields. These keys include Tab, Shift Tab, Home, End, Page Up, Page Down and the arrow keys. The documentation of these keys is beyond the scope of this chapter. For more information about keys, see the *OSF/Motif User's Guide*.

There are many special keys used to edit text input areas.

Table 2-1 contains a list of some of NightProbe's accelerators and the resulting actions; where applicable, it indicates the menu items for which the accelerators provide shortcuts. Note that you can define additional accelerators through the use of X resources (refer to the **X(1)** system manual page).

Table 2-1. Some NightProbe Accelerators

Accelerator	Menu Item	Action
<Control> <N>	File Ì New	Clears all information from the current window; resets the various areas to blank or default values
<Control> <O>	File Ì Open Config File	Opens a configuration file that you have previously saved
<Control> <S>	File Ì Save Config File	Saves the configuration data in the file that is associated with the current window
<Control> <Q>	File Ì Exit	Exits **nprobe**
<Control> <W>	File Ì Close	Closes the window
<F1>		Displays help for the component that currently has the focus
<Shift> <F1>		Performs same function as Help Ì On Context (see "Help" on page 3-21 for more information)

Exiting NightProbe

Prior to exiting **nprobe** you should stop and disconnect the sampler process if you are doing data recording or monitoring. Use the Stop and Disconnect buttons on the Data Recording window.

You exit **nprobe** from the Data Recording window or from a monitoring window by selecting the File Ì Exit menu item.

If you have not saved the changes that you have made in any window, you will be asked to confirm that you wish to exit.

Recording and Monitoring Procedures

This section summarizes the tasks you will need to undertake in order to begin a data monitoring or data recording session. These tasks include:

1. Create a target configuration.
2. Specify the timing source for the sampler.
3. Specify and configure all output viewers that you wish to employ.
4. Verify that target programs are running.
5. Connect the sampler.
6. Start the sampler.

You can perform tasks 1-3 in any order.

Creating a Target Configuration

If you have already created and saved a target configuration in a configuration file, use the File Ì Open Config File menu of the Data Recording window to open your configuration file. You may make changes to the configuration, as described in the rest of this section, or you may proceed to the next task.

If you must create a new configuration or modify a restored configuration, you will use the Program Selection window and one or both of the Variable Attributes and Variable Browser windows.

1. Use the Program Selection window to identify all target programs that you will be sampling. You may enter program names, or use the File Selection or Process Selection windows. See Chapter 5, "Using the Program Selection Window", for details about the Program Selection window.
2. Add addresses or variable names in the Variable Attributes window, or browse the programs' symbol tables and select variables using the Variable Browser window. See Chapter 7, "Using the Variable Attributes Window" for details about the Variable Attributes window. See Chapter 6, "Using the Variable Browser Window", "Variable Browsing" on page A-2, and "Variable Browsing" on page A-14 for details about the Variable Browser window.

   Selected variables are displayed in the list area of the Data Recording

window. You may delete variables using the controls in that window, or modify some of the attributes for displaying the variable by using the Variable Attributes window. See Chapter 3, "Using the Data Recording Window", for details about the Data Recording window.

3. Use the Save Config File or Save Config File As menu items on the File menu of the Data Recording window menu bar to save your created configuration if you wish to reuse it (see "File" on page 3-3). If you do not wish to do any data recording or monitoring at this time, you may exit now.

Specifying the Timing Source

A timing source is used to determine the sampling interval. You may choose from a menu of three sources, using the Timer menu of the Data Recording window menu bar.

- The default timing source is "On Demand", which means that samples are taken only when the Sample button is pressed on the Data Recording window.
- The system clock timing source allows you to specify a time interval that will be measured using the system clock. This clock operates in "wall time" (real world time). You will be asked to specify a time interval between samples.
- The Frequency-Based Scheduler may be used to trigger samples. You will be asked to specify a configured scheduler, plus the first cycle per frame and the cycle period within each frame for the samples to be triggered. You must start the scheduler prior to starting the sampler (starting a scheduler can be accomplished using NightSim or **`rtcp(1)`**).

NOTE

NightSim may be started by selecting the NightSim Scheduler/Performance Monitor menu item from the Tools menu on the Data Recording window. See the *NightSim User's Guide* (0890480) for more information on this tool.

Your currently selected timing source is displayed in the Timing Source area of the Data Recording window.

Specifying and Configuring Output Viewers

You may activate any number of output viewers. Viewers are selected using the Output menu of the Data Recording window menu bar. The choices for viewers are:

- Logging to a file

  You must specify a file name in which the data recording information will be logged during sampling. This file will contain the recorded information in an internal format that can be translated to readable form using the List Viewer window or the **-i** option to **nprobe**. See "Using the List Viewer Window" on page 8-1 for details about the List Viewer window.

- Logging to NightTrace

  You must specify a file name where NightProbe can save the sampled data in a format that can be read by NightTrace. You may also specify the name of a file for NightProbe to write the event name associations map for NightTrace.

  The user is responsible for starting the NightTrace collection daemon. Events sampled before the daemon is running are discarded.

  The output files created by NightProbe and the NightTrace daemon may be used as input for NightTrace.

NOTE

The NightTrace daemon may be started either through the NightTrace GUI or by invoking the user daemon **ntraceud(1)** from the command line. The NightTrace GUI may be started by issuing **ntrace(1)** on the command line or by selecting the NightTrace System Tracing and Analysis menu item from the Tools menu on the Data Recording window (see "Tools" on page 3-19). See the *NightTrace Manual* (0890398) for more information on this tool.

- The List Viewer

  This interactive window reports the recorded data as it is sampled in text form. You may scroll back and forth through the window to see earlier or later samples. See "Using the List Viewer Window" on page 8-1 for details about the List Viewer window.

- The Spreadsheet Viewer

  This interactive window allows for monitoring and modification of the target data locations as the target programs run. You will need to configure the layout of the window to show the data locations in a format that is appropriate for your use. Configuration of the spreadsheet window is described in "Using the Spreadsheet Viewer Window" on page 8-5, "Using the Spreadsheet" on page A-5, and "Using the Spreadsheet" on page A-18.

The currently selected viewers are displayed in the Outputs area of the Data Recording window. You must select at least one viewer to proceed.

Verifying Target Programs

After creating your target configuration and before proceeding to connect the sampler, be sure that all target programs are running. The Program Selection window may indicate process ID numbers, in which case you should verify that these numbers correctly identify the processes running on your system. If no process IDs are indicated in the Program Selection window display area, then NightProbe will use the first process with a name matching the specified program name.

Connecting the Sampler

When you press the Connect button, the target programs will be located, and the addresses for the variables you specified will be ascertained. If any target locations cannot be accessed, the connection will be terminated.

Once NightProbe has resolved all target locations, the sampler process will be ready to begin.

Starting the Sampler

A connected sampler does not take any samples until you start it using the Start button on the Data Recording window (or the Sample button, if you are using the "On Demand" timing source). Once you activate the Start button, samples will be taken and recorded or monitored as you have requested.

You may suspend the sampling at any time using the Stop button (if you are using the "On Demand" timing source, there is nothing to suspend--no samples will be taken except when you click on the Sample button).

Once you have finished with your recording or monitoring session, you should Stop and Disconnect the sampler using the controls on the Data Recording window.

3
Using the Data Recording Window

3
Using the Data Recording Window

The Data Recording window is the primary control window for NightProbe. From this window, you will configure and control the data sampling process.

The Data Recording window allows you to:

- Display the timing source, the destinations for the output of the sampler, the identity of the locations selected for monitoring/recording, and the name of the configuration file selected.
- Access other windows to select a timing source, a destination for the output, variables and locations for monitoring, and attributes for the variables.
- Control the operation of a sampling session, and see at a glance its operational state.
- Save and restore configuration files.
- Get help information on NightProbe.
- Exit NightProbe.

The Data Recording window consists of the following components:

- The Menu Bar
- The Configuration File Status Area
- The Target System Area
- The Timing Source and Outputs Areas
- The Probe Connection Buttons and Status Icon
- The Sampling Control Buttons and Status Icon
- The Variable List
- The Control Area

Figure 3-1 identifies the location of each of these components in the Data Recording window.

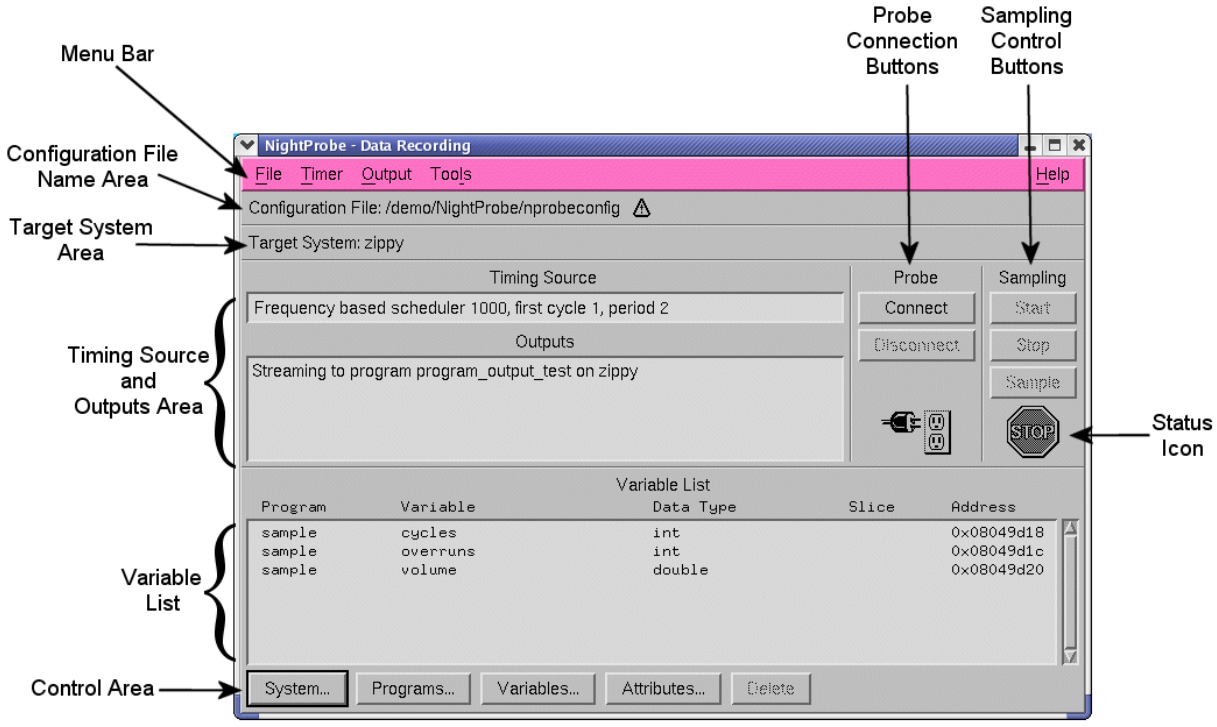

Figure 3-1. Components of the Data Recording Window

Using the Menu Bar

The menu bar provides access to the following menus:

- File
- Timer
- Output
- Tools
- Help

Each menu is described in the sections that follow.

File

Mnemonic: F

The File menu allows you to load a configuration, save the current configuration to a file, or create a new configuration. The File menu also contains the means to exit NightProbe.

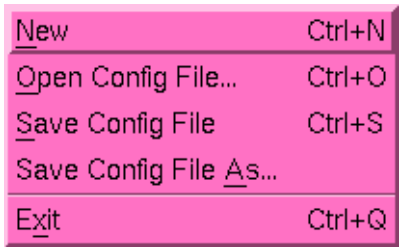

Figure 3-2. File menu

The following paragraphs describe the options on the File menu in more detail.

New
Mnemonic: N
Accelerator: <Control><N>

This option allows you to clear all information from the Data Recording window and reset the various areas to blank or default values. If the window contains unsaved changes, NightProbe displays a warning dialog. You may save the changes, clear the window without saving the changes, cancel the operation, or display help related to the dialog.

Open Config File...
Mnemonic: O
Accelerator: <Control><O>

This option allows you to open a sampler configuration file that you have previously saved.

When you select this option, NightProbe displays a file selection dialog.

To select the file to be opened, use the directory mask text area, scrolled list of directories, scrolled list of files, and file selection text area as appropriate. After making a selection, you may open the selected file, search for another file, cancel the operation, or display help related to the dialog.

If you select a configuration file while a sampler is running, the new configuration will not take effect until the sampler is removed and set up again.

Save Config File
Mnemonic: S
Accelerator: <Control><S>

This option allows you to save the configuration data entered in the Data Recording window in the file that is associated with the window. If the window is not

associated with a configuration file name, this option is the same as Save Config File As.

Save Config File As...
Mnemonic: A

This option allows you to specify the name of the file in which you wish the configuration data entered in the current Data Recording window to be saved.

When you select this option, NightProbe displays a file selection dialog. After making a selection, you may save the current configuration data in the selected file, search for another file, cancel the operation, or display help related to the dialog.

Exit
Mnemonic: X
Accelerator: <Control><Q>

This option exits NightProbe.

Timer

Mnemonic: T

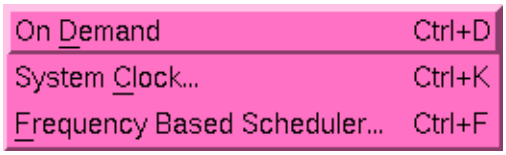

Figure 3-3. Timer menu

The following are the options on the Timer menu:

- On Demand
- System Clock
- Frequency-Based Scheduler

On Demand

Mnemonic: D
Accelerator: <Control><D>

Selecting the On Demand option from the Timer menu means the sampler will sample the variables only when the Sample button in the Sampling area is clicked on. If this option is selected, the Timing Source field will show "on demand".

System Clock

Mnemonic: C
Accelerator: <Control><K>

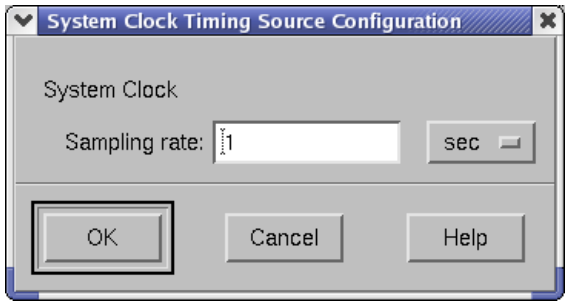

Figure 3-4. System Clock Timing Source Configuration dialog

Selecting the System Clock option from the Timer menu means the sampler will use the system clock as the timing source using the frequency that you select. Figure 3-4 shows the System Clock Timing Source Configuration dialog that appears when you select the System Clock option.

Use this dialog to configure the sampling interval. Choose a unit of time measurement using the Option menu on this window and then enter the amount of time that should pass between samples. The interval you select is displayed in the Timing Source field in the Data Recording window.

Frequency-Based Scheduler

Mnemonic: F
Accelerator: <Control><F>

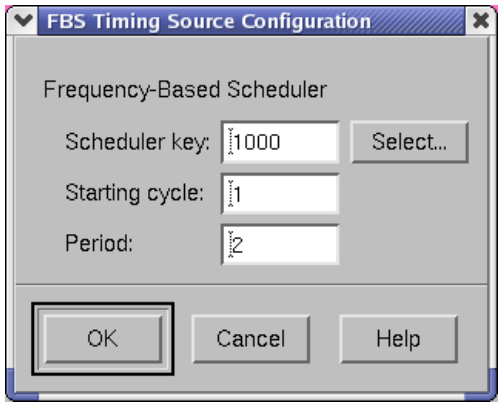

Figure 3-5. Frequency-Based Scheduler Configuration dialog

Selecting the Frequency-Based Scheduler option from the Timer menu means that the NightProbe will take samples as directed by a frequency-based scheduler. This allows for synchronization so that data recording activity does not interfere with the execution of the application.

Figure 3-5 shows the Timing Source Configuration window that appears when you select the Frequency-Based Scheduler option.

Use this dialog to configure the sampling interval. You must specify the Scheduler Key for a configured scheduler, and specify the Starting cycle within each frame and Period (cycle interval) where you want samples taken. The frequency-based scheduler that you specify must be running by the time you connect NightProbe.

To select a scheduler that has already been set up and configured, press the Select… button to display the Frequency-Based Scheduler Selection dialog (see "Frequency-Based Scheduler Selection" on page 3-6).

The timing source may not be changed while the sampler is connected. If you wish to change a timing source, or change the parameters associated with a timing source, you must disconnect the sampler, make the change, and connect again.

Frequency-Based Scheduler Selection

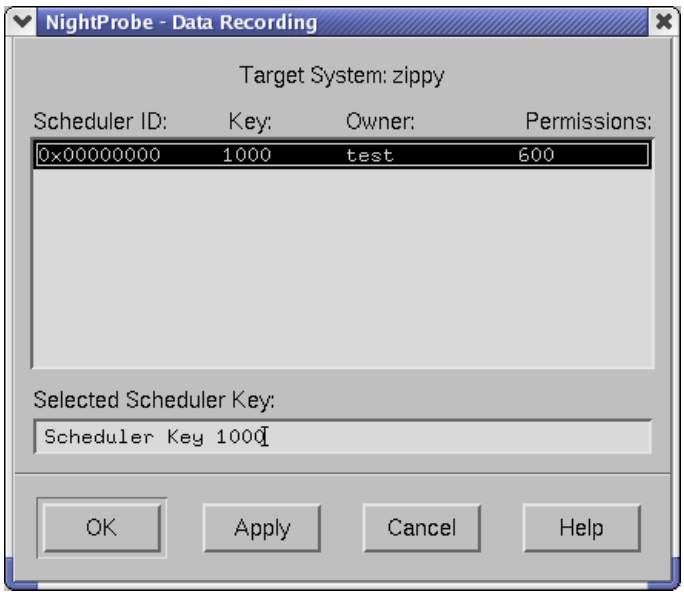

Figure 3-6. Frequency-Based Scheduler Selection dialog

The Frequency-Based Scheduler Selection dialog is opened when the user presses the Select… button for the Scheduler Key field on either the Frequency-Based Scheduler Configuration dialog (see "Frequency-Based Scheduler" on page 3-5) or the Program Output Specification dialog (see "To Program" on page 3-14). This dialog allows the user to select a frequency-based scheduler on the target system.

Selecting the desired scheduler from the list populates the Selected Scheduler Key field appropriately and the choice will be propagated to the Scheduler Key field of the

Frequency-Based Scheduler Configuration dialog when the user presses either Apply or OK.

Output

Mnemonic: O

You must select at least one destination for the output or NightProbe will not be permitted to connect to the target program (see "Using the Probe Connection Buttons" on page 3-23).

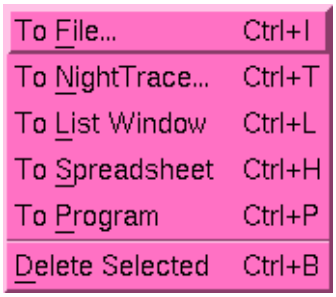

Figure 3-7. Output menu

The following paragraphs describe the options on the Output menu.

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

To File

Mnemonic: F
Accelerator: <Control><I>

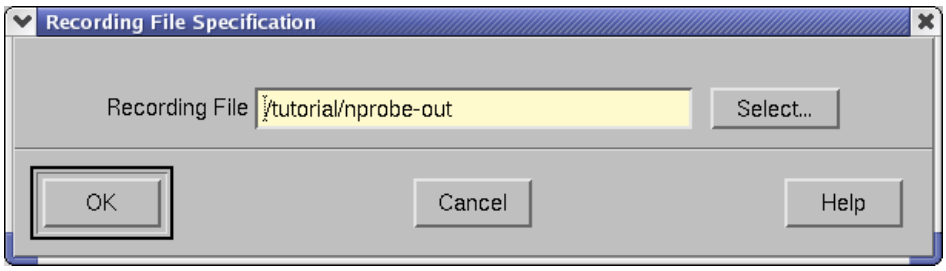

Figure 3-8. Recording File Specification Window

The To File option allows you to specify a file as a destination for the output of the sampler (the recording file).

Figure 3-8 shows the Recording File Specification window that appears when you select the To File option. You may specify the file in one of two ways: by typing in the pathname in the Recording File field or by clicking on the Select button and using the File Selection window that appears to select a new or existing file.

The data recording output in this file can subsequently be processed using the NightProbe APIs (see Chapter 9 "Using the NightProbe API") or can be viewed using the List Window dialog from within NightProbe (see "To List Window" on page 3-13).

To NightTrace

Mnemonic: N
Accelerator: <Control><T>

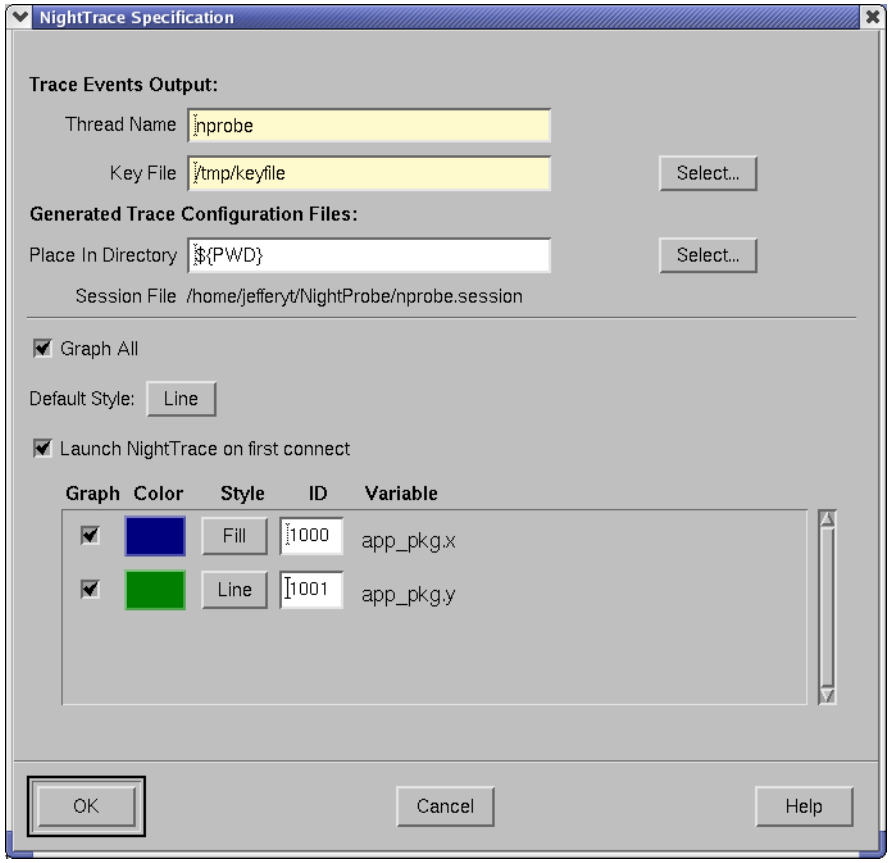

Figure 3-9. NightTrace Specification Window

Figure 3-9 shows the NightTrace Specification window that is presented when you select the To NightTrace option.

This output method allows the user to save the sampled data in the form of NightTrace records that can be streamed to a NightTrace daemon for collection.

Each sampled data value will be logged as a trace event that can be viewed using Night-Trace. In addition, the value of those data values can be graphed on a NightTrace Data Graph.

See the *NightTrace User's Guide* for more information (0890398).

Trace Events Output

Thread Name

Trace events in NightTrace are associated with a particular thread. Trace events associated with the data streamed from this NightProbe session will have the value of this field as their thread name in NightTrace. This can be useful in cases where data is streamed to NightTrace from multiple data sources; the thread name can help to determine the data source.

The default Thead Name for data being streamed to NightTrace from NightProbe is `nprobe`.

Key File

The Key File helps to synchronize the transfer of data from NightProbe to NightTrace. When the NightTrace daemon is started, it is given the value of this key file so that it can receive the trace event data from NightProbe. For instance, if the key file was **`/tmp/mykeyfile`**, the NightTrace daemon could be invoked with the key file:

```
ntraceud /tmp/mykeyfile
```

Select...

Presents a file selection dialog from which to select the key file.

Generated Trace Configuration Files

Place In Directory

NightProbe generates support files that will be used by NightTrace. This directory specifies where NightProbe should save them.

The default location is the current working directory.

Select...

Presents a file selection dialog from which to select the directory in which to store the generated NightTrace configuration files.

Session File

Session configuration files contain information (such as display page configurations and daemon definitions) specific to a particular session of NightTrace. Information related to this session of NightProbe will be stored in the file listed here.

Graph All

When this checkbox is selected, Data Graphs will be created in the generated NightTrace display page for all monitored variables. The user may make individual selec-

tions via the Graph checkbox associated with each variable in the list at the bottom of this dialog (see "Graph" on page 3-11).

Default Style

Values in the Data Graphs on the NightTrace display page associated with this session of NightProbe may appear as either lines or bars of varying height. The height of the bar or line reflects the value of the variable.

The choice of Default Style determines the default style for all monitored variables. The user may make individual selections via the Style option associated with each variable in the list at the bottom of this dialog (see "Style" on page 3-12).

Launch NightTrace on first connect

When this option is selected, NightTrace will be started the first time that NightProbe connects to the target application (see "Using the Probe Connection Buttons" on page 3-23).

You may then start the daemon either through the NightTrace GUI or by invoking the user daemon **`ntraceud(1)`** from the command line.

If NightProbe disconnects from the target application and changes are made in this dialog, NightTrace must reload its configuration files to be informed of the changes.

NOTE

If this option is not selected, NightTrace may be started by issuing **`ntrace(1)`** on the command line or by selecting the NightTrace System Tracing and Analysis menu item from the Tools menu on the Data Recording window (see "Tools" on page 3-19). See the *NightTrace Manual* (0890398) for more information on this tool.

The bottom portion of this dialog contains information specific to each of the monitored variables.

Graph

When this checkbox is selected, a Data Graph will be created in the generated NightTrace display page for the associated variable. The user may select all variables via the Graph All checkbox that appears in the upper portion of this dialog (see "Graph All" on page 3-10).

Color

Specifies the color of the bar or line representing the trace event associated with the variable.

Clicking on the color presents the user with a color selection dialog.

Style

Trace events may appear as either lines or bars of varying height in the Data Graphs on the NightTrace display page associated with this session of NightProbe. The height of the bar or line reflects the value of the variable.

The value selected here determines the style of the associated variable. The default style is determined by the Default Style setting that appears in the upper portion of this dialog (see "Default Style" on page 3-11).

ID

Unique identifier used as the trace event ID for the trace events associated with this variable.

Variable

The name of the monitored variable.

Figure 3-9 shows a NightTrace display page generated using the To NighTrace ouput method. Data Graphs for two variables appear on the grid. The Data Graph associated with the variable `app_pkg.x` contains blue bars of varying heights; the Data Graph associated with the variable `app_pkg.y` consists of green lines of varying heights.

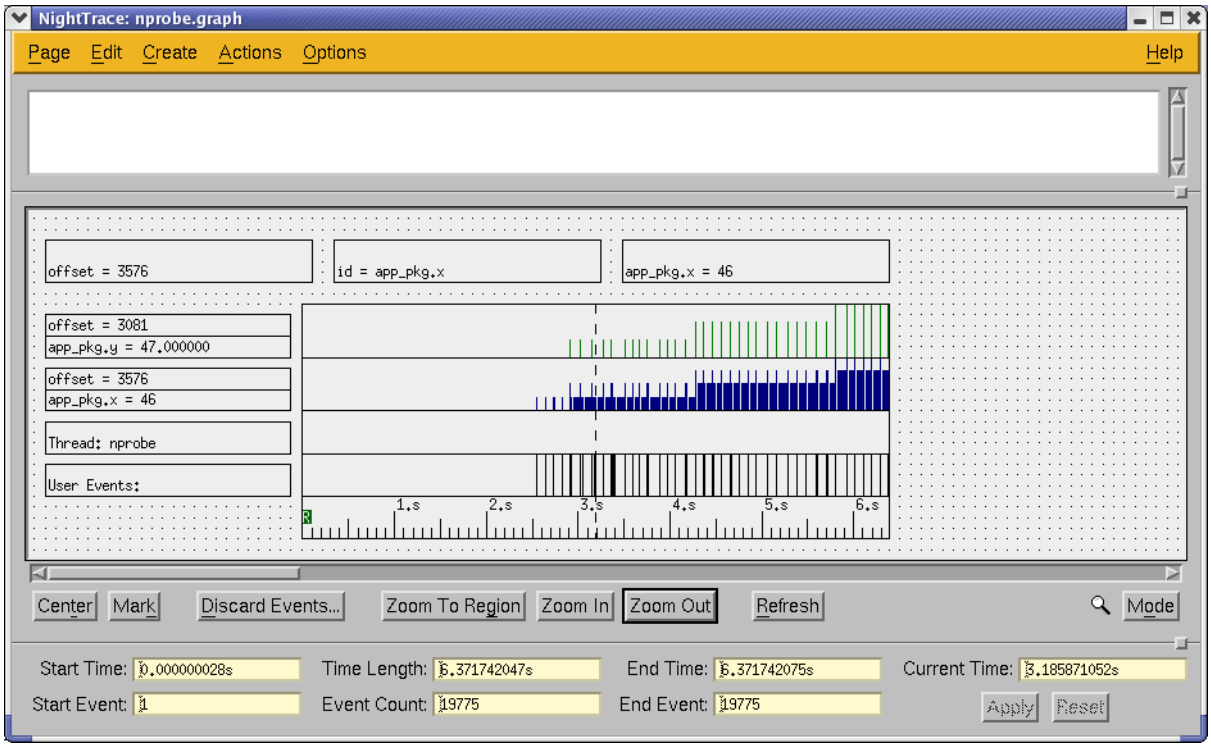

Figure 3-10. NightTrace display page

To List Window

Mnemonic: L
Accelerator: <Control><L>

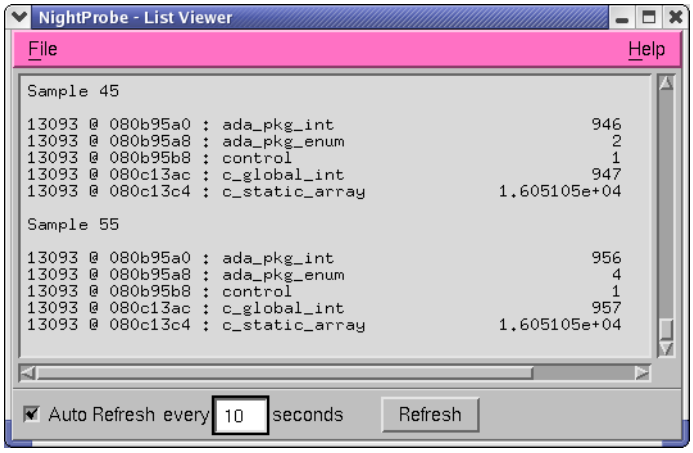

Figure 3-11. List Viewer Window

The To List Window option displays a new window, as shown in Figure 3-11, that allows you to monitor variable values while the program is running. The List Viewer window is described in "Using the List Viewer Window" on page 8-1.

To Spreadsheet

Mnemonic: S
Accelerator: <Control><H>

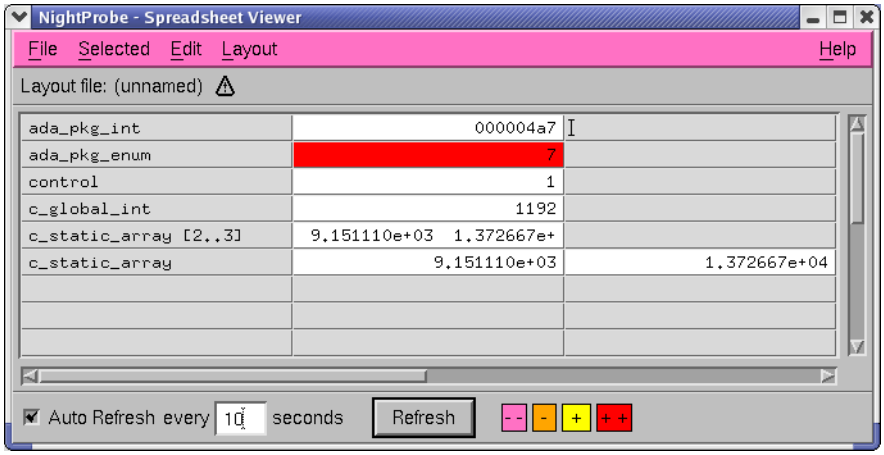

Figure 3-12. Spreadsheet Viewer Window

The To Spreadsheet option displays a new window, as shown in Figure 3-12, that allows you to monitor and modify variables while the program is running. The Spreadsheet Viewer window is described in "Using the Spreadsheet Viewer Window" on page 8-5.

To Program

Mnemonic: P
Accelerator: <Control><P>

The To Program option on the Output menu presents the user with the Program Output Specification window as shown in Figure 3-13.

This dialog allows the user to specify a program that will be used to process the data recording output using the NightProbe Application Programming Interface (see Chapter 9 "Using the NightProbe API").

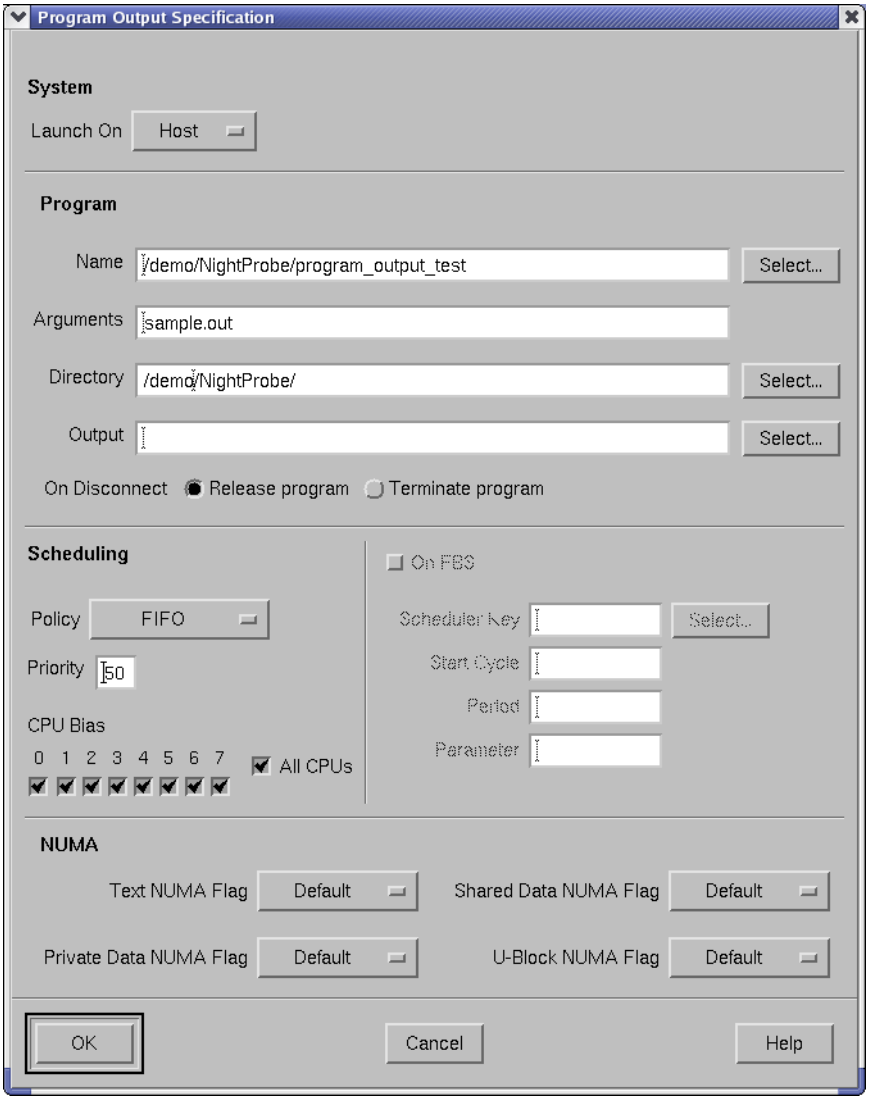

Figure 3-13. Program Output Specification Window

Computer

Launch On

Indicates whether the program should be executed on the Host system (where the GUI portion of NightProbe is running) or the Target system (where the probed application is running).

Program

Name

The name of the program used to process the data recording output streamed from NightProbe.

Select...

Presents a file selection dialog from which to select the desired program.

NOTE

Pathnames are relative to the host system.

Arguments

Any arguments to be passed to the program are entered in this field.

Directory

The current working directory for the program.

Select...

Presents a file selection dialog from which to select the desired directory.

NOTE

Pathnames are relative to the host system.

On Disconnect

This selection determines how NightProbe handles the program which processed the data recording output when NightProbe disconnects from the target program.

Release Program

Allow the program which processed the data streamed from NightProbe to continue running after NightProbe has disconnected from the target program (see "Using the Probe Connection Buttons" on page 3-23).

Terminate Program

Terminates the program which processed the data streamed from NightProbe to continue running after NightProbe has disconnected from the target program (see "Using the Probe Connection Buttons" on page 3-23).

Scheduler

Policy

This panel of radio buttons allows you to select the POSIX scheduling policy for the specified program. The options are as follows: the first–in–first–out scheduling policy (FIFO), the round–robin scheduling policy (Round Robin), and the other scheduling policy (Time-Sharing)

For a full description of the behavior of processes that are scheduled under the respective policies on PowerMAX OS systems, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide* (0890423); on RedHawk Linux systems, refer to the "Process Scheduling" chapter of the *RedHawk Linux User's Guide* (0898004).

Priority

The priority of the program used to process data recording output. The range of priority values that you can enter is governed by the scheduling policy specified (see "Policy" above). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities on PowerMAX OS systems, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide* (0890423); on RedHawk Linux systems, refer to the "Process Scheduling" chapter of the *RedHawk Linux User's Guide* (0898004).

CPU Bias

This panel of checkbuttons allows you to select the processor or processors on which the program processing data recording output can be scheduled.

The user can choose to run the program that processes the data recording output on a different CPU from the CPU on which their shielded application is running, thereby reducing interference.

All CPUs

Specifies that the program processing data recording output can be scheduled on all available processors.

On FBS

This checkbox should be checked if the program processing the data recording output is to be scheduled on the frequency-based scheduler.

Scheduling the program in cycles unused by the probed application allows for minimal interference with that application.

Scheduler Key

This field allows you to specify the *key* of the frequency–based scheduler that you wish to use. The key is a user–chosen numeric identifier with which the scheduler is associated.

Select...

Presents the Frequency-Based Scheduler Selection dialog (see "Frequency-Based Scheduler Selection" on page 3-6) allowing the user to select a frequency-based scheduler on the target system.

Start Cycle

This field allows you to specify the first *minor cycle* in which the specified program is to be wakened in each *major frame*. Enter a number ranging from zero to the total number of minor cycles per frame minus one.

Selecting a start cycle which is not used by the probed application will reduce the interference with that application.

Period

This field allows you to establish the frequency with which the specified program is to be wakened in each *major frame*. A period of one indicates that the specified program is to be wakened every *minor cycle*; a period of two indicates that it is to be wakened once every two minor cycles, etc. Enter the number of minor cycles representing the frequency with which you wish the program to be wakened. This number can range from zero to the number of minor cycles that compose a frame on the scheduler.

Parameter

This field allows you to pass an integer value to the program when it is scheduled on the frequency–based scheduler. The value must be a 32-bit decimal value. The default is no value.

NUMA

NOTE

The NUMA flags only apply to programs running on a PowerMAX OS system.

Text NUMA Flag

This item selects the NUMA policy to apply to text (code) pages.

See "Policies" on page 4-6 for a list of applicable NUMA policies.

Private Data NUMA Flag

This item selects the NUMA policy to apply to private data pages.

See “Policies” on page 4-6 for a list of applicable NUMA policies.

Shared Data NUMA Flag

This item selects the NUMA policy to apply to shared data pages.

See “Policies” on page 4-6 for a list of applicable NUMA policies.

U-Block NUMA Flag

This item selects the NUMA policy to apply to kernel data structures that contain the stack used during system calls, the register save area used during context switches, and miscellaneous other bits of information about the LWP.

See “Policies” on page 4-6 for a list of applicable NUMA policies.

Delete Selected

Mnemonic: D
Accelerator: <Control><B>

The Delete Selected option on the Output menu allows you to delete the currently selected output destination. Select an output by clicking mouse button 1 when positioned over the output viewer name in the outputs list.

Tools

Mnemonic: L

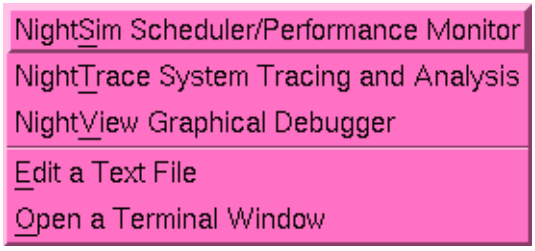

Figure 3-14. Tools menu

The following describe the options on the Tools menu:

NightBench Program Development Environment
Mnemonic: B

Opens the NightBench Program Development Environment. NightBench is a set of graphical user interface (GUI) tools for developing software with the Concurrent C/C++ and MAXAda™ compiler toolsets.

NOTE

NightBench is currently not available on RedHawk systems.

See also:

- *NightBench User's Guide* (0890480)

NightSim Scheduler/Performance Monitor
Mnemonic: S

Opens the NightSim Application Scheduler. NightSim is a tool for scheduling and monitoring real-time applications which require predictable, repetitive process execution. With NightSim, application builders can control and dynamically adjust the periodic execution of multiple coordinated processes, their priorities, and their CPU assignments.

See also:

- *NightSim User's Guide* (0890480)

NightTrace System Tracing and Analysis
Mnemonic: T

Opens the NightTrace Analyzer. The NightTrace Analyzer is a graphical tool for analyzing the dynamic behavior of multiprocess and/or multiprocessor user applications and operating system activity. NightTrace allows the user to control user and kernel trace collection daemons and can graphically display the interplay between many real-time programs and processes across multiple processors and systems.

See also:

- *NightTrace Manual* (0890398)

NightView Graphical Debugger
Mnemonic: V

Opens the NightView Source-Level Debugger. NightView is a graphical source-level debugging and monitoring tool specifically designed for real-time applications. NightView can monitor, debug, and patch multiple real-time processes running on multiple processors with minimal intrusion.

See also:

- *NightView User's Guide* (0890395)

Edit a Text File
Mnemonic: E

Opens the NEdit text editor.

Open a Terminal Window
Mnemonic: O

Opens a terminal window by invoking **`nterm(1)`**, the NightStar enhanced terminal emulator. If **`nterm`** is not installed, **`xterm(1)`** is invoked.

Help

Mnemonic: H

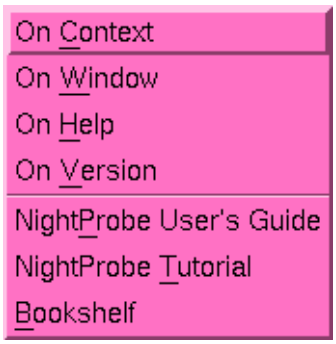

Figure 3-15. Help menu

The following describe the options on the Help menu:

On Context
Mnemonic: C

Gives context-sensitive help on the various menu options, dialogs, or other parts of the user interface.

Help for a particular item is obtained by first choosing this menu option, then clicking the mouse pointer on the object for which help is desired (the mouse pointer will become a floating question mark when the On Context menu item is selected).

In addition, context-sensitive help may be obtained for the currently highlighted option by pressing the F1 key. HyperHelp™, NightProbe's online help system, will open with the appropriate topic displayed.

On Window
Mnemonic: W

Displays help information for the current window.

On Help
Mnemonic: H

Displays help information about how to use HyperHelp, NightProbe's online help system.

NightProbe User's Guide
Mnemonic: P

Opens the online version of the *NightProbe User's Guide* in the HyperHelp viewer.

NightProbe Tutorial
Mnemonic: T

Opens HyperHelp, NightProbe's online help system, to the section containing a tutorial which shows some of the commonly used features of NightProbe.

Bookshelf
Mnemonic: B

Opens a HyperHelp window that lists all of the currently available HyperHelp publications.

Using the Configuration File Status Area

The Configuration File Status Area is located just below the menu bar and contains information on the name of the currently selected configuration file and a warning indicator icon if the current configuration is different from what was loaded from or last saved to that file. Configuration files are selected with the Open Config File option on the File menu or are provided on the command line. This area is display-only; no user modifications are accepted here.

Using the Target System Area

The Target System area indicates the target system where data sampling will occur. Currently, a single NightProbe session can only target one system. If multiple target systems are required, use multiple NightProbe sessions.

Use the System… button to obtain a window to allow you to select the target system as well as the login name of the user connecting to the target programs. See "Using the Control Area" on page 3-25 for more information.

Using the Timing Source and Outputs Areas

The Timing Source and Outputs areas are display areas that show the currently selected Timing Source and Output destination, respectively.

The possible timing sources are: on demand, system clock, and frequency-based scheduler. Use the Timer menu bar to select and configure a timing source.

The Outputs area shows you the output destinations you have selected. You must select at least one viewer or NightProbe will not allow you to sample data. The possible output destinations for the data from the sampling are: a file, a file especially formatted for use by the NightTrace application, a List Viewer window, and a Spreadsheet Viewer window. Use the Outputs menu bar to add output destinations.

Using the Probe Connection Buttons

The two Probe buttons located to the right of the Timing Source and Output areas of the Data Recording window connect and disconnect a sampler process, which samples the values of the target program(s) variables.

Connect

This button connects (initializes) the sampler. A sampler may be connected after the timing source, outputs, and variables have been selected, and when the target program(s) are all running. Once connected, changes to the configuration are not allowed.

If a target system has been selected which requires user authentication for the specified user (see "Using the Target System Selection Window" on page 4-1), the User Authentication dialog will be presented when the Connect button is pressed (see "User Authentication" on page 4-3).

Disconnect

This button stops and disconnects the current sampler.

If you wish to change the configuration, you must first disconnect the active sampler, make the changes, and use Connect again.

It is important to realize that the binding of specified locations to the target program(s) takes place when Connect is activated. It is at that time that the target program(s) must be executing.

An icon below the buttons indicates whether the sampler is connected.

Using the Sampling Control Buttons

The three Sampling buttons located on the right side of the Data Recording window control the sampling of the target program variables. A sampler may be connected after the timing source, outputs, and target locations have been selected. Samples will be taken at intervals between the time that you press the Start button and the time you press the Stop button, or at the time you press the Sample button (when the Timing Source is set to On Demand).

Start

Clicking on the Start button causes the sampler to start sampling all the selected variable values and record the results to the appropriate file or viewer.

Stop

Clicking on the Stop button causes the sampler to stop sampling all the selected variable values. The Start button may be used to resume sampling.

Sample

Clicking on the Sample button causes the sampler to sample all the selected variable values once. This option is available only when On Demand has been selected as the Timing Source (in which case the Start and Stop options are disabled). See "On Demand" on page 3-4.

An icon below the buttons indicates whether the sampler is actively sampling or is currently suspended.

Using the Variable List

The Variable List is a scrolled list near the bottom of the Data Recording window that shows you what program data locations you have selected for monitoring. The Variable List is divided into the following five different fields:

Program

The name of the program that is being monitored. Use the Programs… button to obtain a window to allow you to select programs. See "Using the Control Area" on page 3-25 for more information.

Variable

The name of the variable that is being monitored. Use the Variables… button to obtain a window to allow you to search for and select the variables you want to monitor. See "Using the Control Area" on page 3-25 for more information.

If the location was specified using an absolute address, then the Variable and Data Type fields will be blank. If the name is long, the leftmost characters are replaced with two dots (`..`) in the display.

TIP:

The fields in the Variable List in the Data Recording window have a fixed width that may be too small to display all the information for a variable. To see the complete information, bring up the Variable Attributes window and select a variable in the Variable List. The Variable Attributes window displays all information about that variable.

Data Type

The data type of the variable (for example, `char`). If the data type is long, the leftmost characters are replaced with two dots (`..`) in the display.

Slice

The array element or range of elements when a part of an array is selected. For more information, see "Array Slices" on page 2-2.

Address

The logical address of the location to be monitored. If more than address is to be monitored (for example, an array), this is the starting address.

Selecting one line in this output area, by pointing the mouse and clicking mouse button 1, will show the variable and its attributes in the Variable Attributes window, if that window is visible. Double clicking on a line will open the Variable Attributes window and display the selected variable and its attributes.

In order to delete variables from the Variable List, you must first select them and then use the Delete button in the control area.

Using the Control Area

At the bottom of the Data Recording window is the control area for the Variable List. These buttons affect the addition and deletion of information to the Variable List. See "Using the Variable List" on page 3-24.

System...

Clicking on the System button brings up the Target System Selection window, which allows you to select the system on which the target programs are running. See Chapter 4, "Using the Target System Selection Window".

Programs...

Clicking on the Programs button brings up the Program Selection window, which allows you to select a program for monitoring. See Chapter 5, "Using the Program Selection Window".

Variables...

Clicking on the Variables button brings up the Variable Browser window, which allows you to search for and select the variables you want to monitor. See Chapter 6, "Using the Variable Browser Window".

Attributes...

Clicking on the Attributes button brings up the Variable Attributes window, which allows you to specify a variable for monitoring/recording, view information about a variable (for example, its data type), and set various attributes (for example, the output format for displaying the variable's value). See Chapter 7, "Using the Variable Attributes Window".

Delete

Clicking on the Delete button removes all selected variable locations from the current configuration. Select locations using the mouse to highlight items in the Variable List.

4
Using the Target System Selection Window

4
Using the Target System Selection Window

The Target System Selection window allows the user to specify the system on which the target programs are running, the login name of the user connecting to the target system, and the run-time attributes associated with the data monitoring activities on the target.

The Target System Selection window is opened by clicking on the System… push button at the bottom of the Data Recording window (see "Using the Data Recording Window" on page 3-1).

The Target System Selection window can be divided into the following areas:

- Server Attributes Area (see "Using the Server Attributes Area" on page 4-2)
- Scheduling Area (see "Using the Scheduling Area" on page 4-3)
- CPU Bias Area (see "Using the CPU Bias Area" on page 4-5)
- NUMA Area (see "Using the NUMA Area" on page 4-5)
- Control Area (see "Using the Control Area" on page 4-6)

Figure 4-1 identifies the location of each of these components in the Target System Selection window.

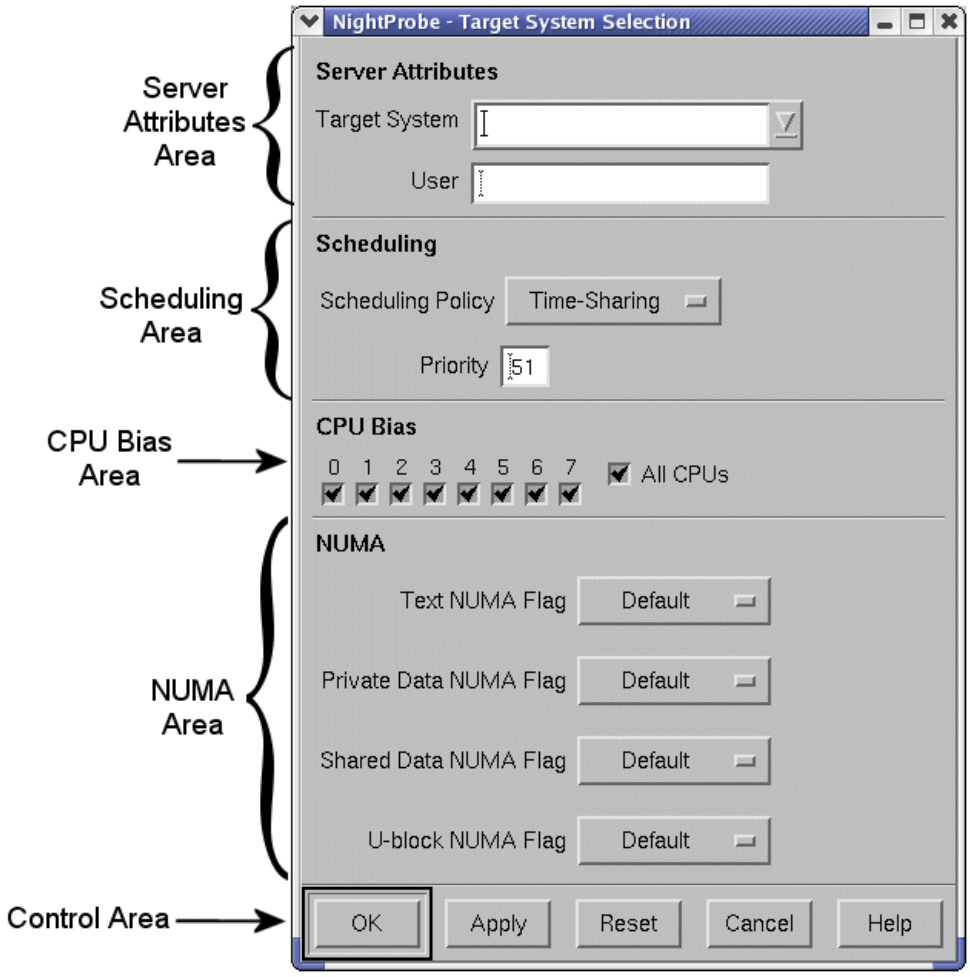

Figure 4-1. Components of the Target System Selection Window

Using the Server Attributes Area

The server attributes area consists of the following fields:

Target System — The Target System field is the name of the system on which the programs to be monitored by NightProbe are running.

User — The User field specifies the login name of the user connecting to the target programs on the target system.

User Authentication

User authentication may be necessary when NightProbe attempts to connect to the target system as the specified user.

If user authentication is required, the following dialog will be presented when the Connect button on the Data Recording window is pressed (see "Using the Probe Connection Buttons" on page 3-23).

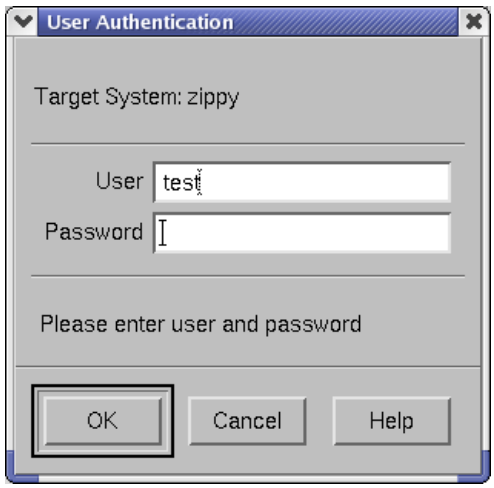

Figure 4-2. User Authentication Window

The User Authentication dialog contains the following fields:

User — The name of the user connecting to the target system.

Password — The password for the specified User on the target system.

Once authentication has succeeded, NightProbe will no longer require you to re-authenticate even if you disconnect and reconnect, unless you change the target system or user name, or exit NightProbe. Note that NightProbe does not store any password information on disk.

Using the Scheduling Area

The Scheduling Area allows the user to select a scheduling policy and assign a priority for the data monitoring activities on the target system.

Scheduling Policy

POSIX defines three types of policies that control the way a process is scheduled by the operating system. They are `SCHED_FIFO` (FIFO), `SCHED_RR` (Round Robin), and `SCHED_OTHER` (Time-Sharing). Each of these scheduling policies

is associated with one of the System V scheduler classes. See either the *PowerMAX OS Programming Guide* (0890423) or the *RedHawk Linux User's Guide* (0898004) for more detailed information regarding these policies and their associated classes.

FIFO

The FIFO (first–in–first–out) policy (SCHED_FIFO) is associated with the fixed-priority class in which critical processes and LWPs can run in predetermined sequence. Fixed priorities never change except when a user requests a change.

This policy is almost identical to the Round Robin (SCHED_RR) policy. The only difference is that a process scheduled under the FIFO policy does not have an associated time quantum. As a result, as long as a process scheduled under the FIFO policy is the highest priority process scheduled on a particular CPU, it will continue to execute until it voluntarily blocks.

Round Robin

The Round Robin policy (SCHED_RR), like the FIFO policy, is associated with the fixed-priority class in which critical processes and LWPs can run in predetermined sequence. Fixed priorities never change except when a user requests a change.

A process that is scheduled under this policy (as opposed to the FIFO policy) has an associated time quantum.

Time-Sharing

The Time-Sharing policy (SCHED_OTHER) is associated with the time-sharing class, changing priorities dynamically and assigning time slices of different lengths to processes in order to provide good response time to interactive processes and LWPs and good throughput to CPU-bound processes and LWPs.

Priority

The Priority is relative to the selected Scheduling Policy (see "Scheduling Policy" on page 4-3) and the range of allowable values is dependent on the operating system.

For example, on PowerMAX OS systems, the priority values for the FIFO class include 0..59, where 59 is the most urgent user priority available on the system.

On RedHawk systems, the priority values for the FIFO class include 1..99, where 99 is the most urgent user priority available on the system.

Using the CPU Bias Area

The CPU Bias Area allows the user to select certain CPUs on the target system which can be used for monitoring and recording data.

CPU Bias

Select particular CPUs by checking the desired checkboxes.

All CPUs

Selects all CPUs on the target system.

Using the NUMA Area

On platforms belonging to the local/global/remote subclass of non-uniform memory access (NUMA) architectures, primary memory is divided into global and local memories.

Global memory is located on a memory board where it is equally distant, in terms of access time, from all of the CPUs in the system. All CPUs share a single data path to global memory known as the system bus.

Local memory is located on a CPU board where it is closer, in terms of access time, to the co-resident CPUs. The path between a CPU and its local memory does not include the system bus. Local memory usage improves the throughput of the system in two ways: smaller access times for the co-resident CPUs and less system bus contention for the remaining CPUs.

Applications can influence the page placement decisions made by the kernel by selecting NUMA policies for different parts of their address spaces. NUMA policies specify where data should reside in the local/global/remote hierarchy.

NOTE

These settings are ignored for non-NUMA target systems architectures, such as PowerHawk, PowerStack, and iHawk series machines.

Text NUMA Flag

This item selects the NUMA policy to apply to text (code) pages.

See “Policies” in the section below for a list of applicable NUMA policies.

Private Data NUMA Flag

This item selects the NUMA policy to apply to private data pages.

See "Policies" in the section below for a list of applicable NUMA policies.

Shared Data NUMA Flag

This item selects the NUMA policy to apply to shared data pages.

See "Policies" in the section below for a list of applicable NUMA policies.

U-block NUMA Flag

This item selects the NUMA policy to apply to kernel data structures that contain the stack used during system calls, the register save area used during context switches, and miscellaneous other bits of information about the LWP.

See "Policies" in the section below for a list of applicable NUMA policies.

Policies

Each of the above flags can be set to one of the following:

Global

Specifies that the designated pages should be placed in global memory.

Soft Local

Specifies that the designated pages be placed in local memory if space is available, otherwise they should be placed in global memory.

Hard Local

Specifies that the designated pages must be placed in local memory.

Default

Specifies that the default NUMA policy on the target system should be used.

Using the Control Area

The control area is located along the bottom of the Target System Selection window.

The following paragraphs explain the effects the buttons have.

OK — The OK button sets the values specified in the Target System Selection dialog and closes the dialog.

Apply — The Apply button sets the values specified in the Target System Selection dialog but leaves the dialog open.

Reset — The Reset button clears the server attributes area.

Close — The Close button closes the Target System Selection window without making any changes.

Help — The Help button opens the HyperHelp viewer displaying the online help topic for the Target System Selection window. See "Getting Help" on page 2-8 for details.

5
Using the Program Selection Window

5
Using the Program Selection Window

The Program Selection window specifies the programs to be monitored or recorded.

The Program Selection window is selected by clicking on the Programs push button at the bottom of the Data Recording window (see "Using the Data Recording Window" on page 3-1) or the Programs push button at the top of the Variable Browser or Variable Attributes windows (see "Using the Variable Browser Window" on page 6-1 and "Using the Variable Attributes Window" on page 7-1).

The Program Selection window can be divided into the following areas:

- Program List (see "Using the Program List" on page 5-1)
- Program Information Area (see "Using the Program Information Area" on page 5-2)
- Control Area (see "Using the Control Area" on page 5-5)

Figure 5-1 identifies the location of each of these components in the Program Selection window.

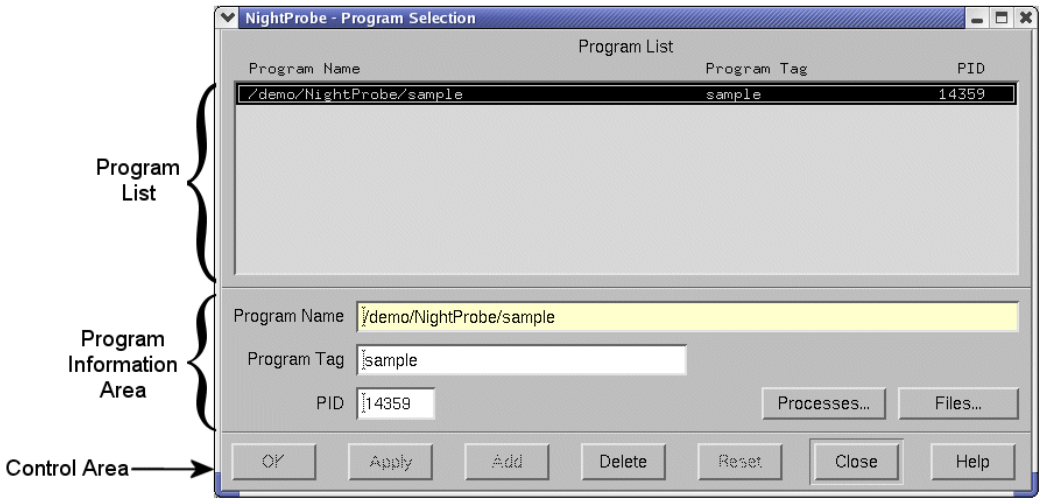

Figure 5-1. Components of the Program Selection Window

Using the Program List

The top half of the Program Selection window contains the Program List. It displays in a scrolled list the programs that you have selected for monitoring/recording.

When a program is added to the list, it can then be accessed by the Variable Attributes and Variable Browser windows (using the Program Name menu) to select symbols within the program for monitoring/recording. The Program List contains the following fields:

Program Name	The Program Name is the file name of the program to monitor. Note that this file name is relative to the current working directory if not specified as a complete pathname.
Program Tag	The Program Tag is an alternate and unique name for the program. This is useful if more than one copy of the same program is running.
PID	The PID field is the process ID of the program when it is executing. If no PID is shown, NightProbe will monitor the first process it encounters in the process table with a matching name.

Using the Program Information Area

The program information area is below the program list area of the Program Selection window. It provides three different ways to add/change a program to the Program List for monitoring/recording:

- Pressing the Processes button, which brings up the Process Selection window (see "Process Selection Window" on page 5-2).
- Pressing the Files button, which brings up the File Selection window (see "File Selection Window" on page 5-3).
- Entering information directly into the program information fields (see "Program Information Fields" on page 5-4).

Process Selection Window

The Processes button brings up the Process Selection window, which presents a list of the process IDs, owner user names, and process names running on the system and allows you to select one for monitoring. Selecting a file in the Process Selection window automatically inserts the Program Name, Program Tag, and PID in their respective fields in the Program Selection window (see "Program Information Fields" on page 5-4).

Filters are patterns constructed using standard regular expression syntax. The default PID and Program filter is:

```
.*
```

which means that everything is displayed. The default User filter is your login name.

When you use the Process Selection window to select a program, NightProbe attempts to construct a complete pathname based on the simple name in the process table and the current `PATH` environment variable. If a complete pathname cannot be determined, then the simple name is used and you may need to add the correct pathname to the program name field in the Program Selection window. Otherwise, NightProbe will not be able to locate the target process and will generate an error message.

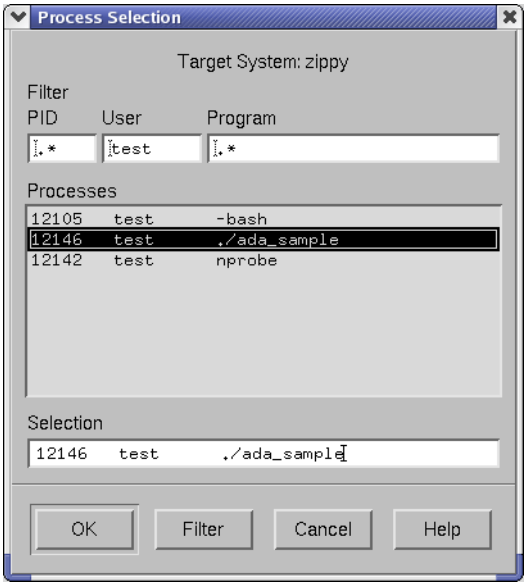

Figure 5-2. Process Selection Window

File Selection Window

The Files button brings up the File Selection window, which presents lists of files and directories on the system and allows you to select one for monitoring/recording. The default filter is `*`, which means that everything is displayed. The wildcard characters are the same as the ones used by the shell (e.g., `*`, `?`, etc.). Selecting a file in the File Selection window automatically inserts the file name in the Program Name field (see "Program Information Fields" on page 5-4).

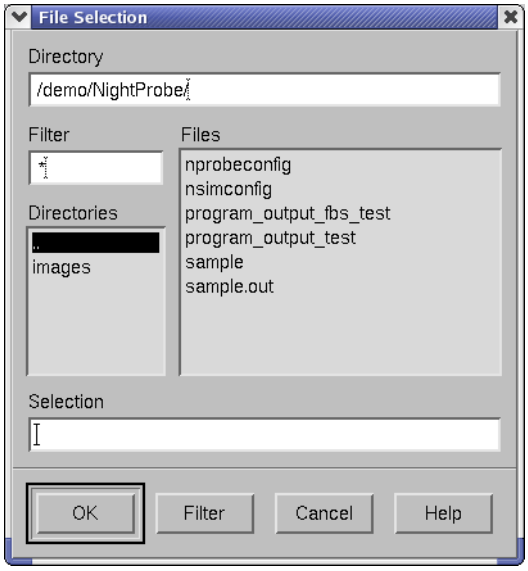

Figure 5-3. File Selection Window

Program Information Fields

The program information fields are described below:

Program Name — The Program Name field is the full pathname of the program to monitor or a relative pathname based at the current working directory. You must either give an absolute pathname or the correct relative pathname.

Program Tag — The Program Tag field is an alternate and unique name for the program. This is useful if more than one copy of the same program is running. If this field is left blank, it is filled in automatically based on the simple file name of the program. If the same program is referenced again later, the program tag appends a unique character sequence to the file name.

PID — The PID field contains the process ID of an executing program. If it is left blank, NightProbe will monitor the first process it encounters in the process table with a matching name.

Adding or modifying the Program List using the buttons in the control area (see "Using the Control Area" on page 5-5) will move the information from the program information fields to the Program List area.

Using the Control Area

The control area is located along the bottom of the Program Selection window. Some of the buttons in the control area affect the Program List, some buttons affect the program information area, and some of the buttons affect the Program Selection window itself.

The following paragraphs explain the effects the buttons have. Note that to highlight an item in the Program List, move the mouse pointer to the row the item is on and click mouse button 1.

OK — The OK button replaces the highlighted item in the Program List with the information in the program information area and closes the Program Selection window. If no program was highlighted in the Program List, the program specified in the program information area is appended to the Program List.

Apply — The Apply button replaces the highlighted item in the Program List with the information in the program information area and clears the program information area. If no program was highlighted in the Program List, the program specified in the program information area is appended to the Program List.

Add — The Add button adds the program specified in the program information area to the Program List. It also clears the program information area.

Delete — The Delete button deletes the highlighted item in the Program List.

Reset — The Reset button clears the program information area.

Close — The Close button closes the Program Selection window. Any programs on the Program List remain on the list but any programs specified in the program information area are discarded.

Help — The Help button opens the HyperHelp viewer displaying the online help topic for the Program Selection window. See “Getting Help” on page 2-8 for details.

6
Using the Variable Browser Window

6
Using the Variable Browser Window

The Variable Browser window allows you to peruse the symbol tables of the target programs to help you to select variables for monitoring and recording. The following figure identifies the location of each of the four main areas in the Variable Browser window.

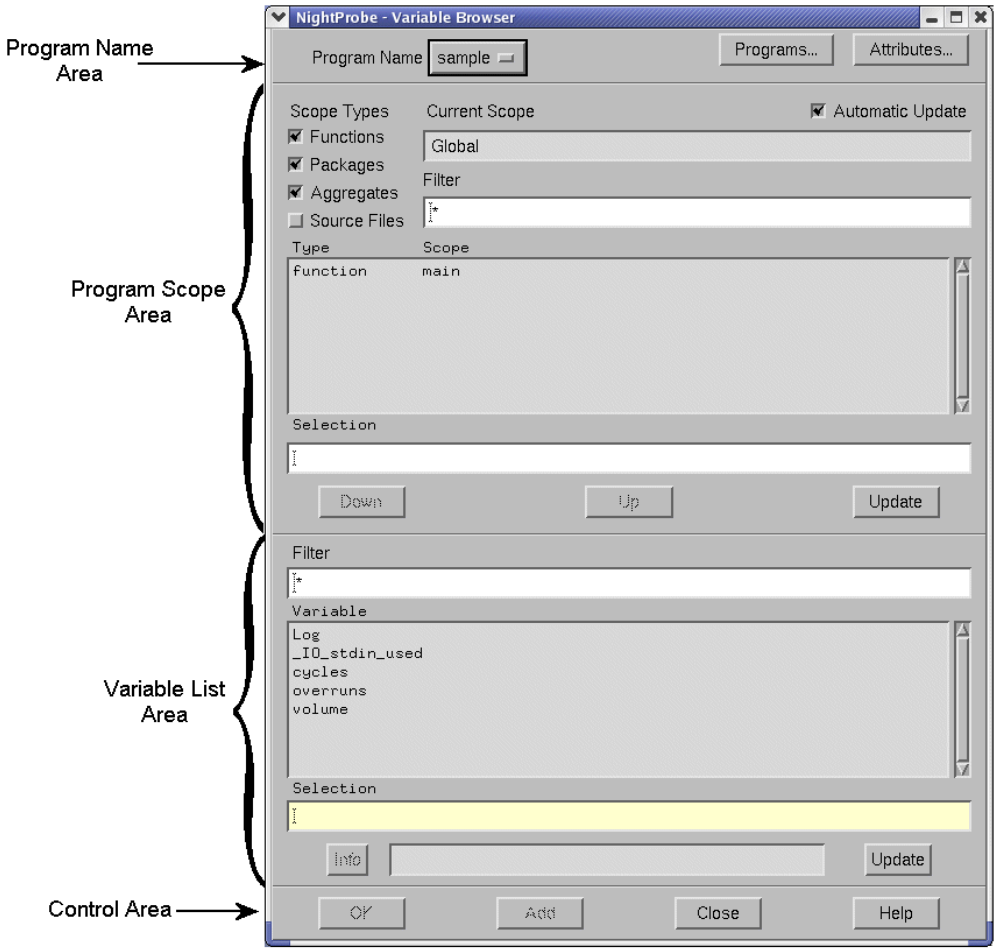

Figure 6-1. Components of the Variable Browser Window

- The Program Name Area (see “Using the Program Name Area” on page 6-2)
- The Program Scope Area (see “Using the Program Scope Area” on page 6-3)
- The Variable List Area (see “Using the Variable List Area” on page 6-6)

- The Control Area (see "Using the Control Area" on page 6-6)

The general procedure for using this window is as follows.

1. Using the Program Name Area, select a program name to monitor/record.
2. Using the Program Scope Area, select a function, package, aggregate, or source file that contains the variable you wish to monitor.
3. Using the Variable List Area, select the variable or variables in the function or source file that you wish to monitor.
4. Using the Control Area, add these variables to the Variable List (see "Using the Variable List" on page 3-24) shown in the Data Recording window (see Chapter 3, "Using the Data Recording Window").

See "Variable Browsing" on page A-2 for a tutorial on variable browsing in an Ada program and "Variable Browsing" on page A-14 for a tutorial on variable browsing in a C++ program.

Using the Program Name Area

The Program Name area is located at the top of the Variable Browser window (see Figure 6-1, "Components of the Variable Browser Window," on page 6-1). This area provides access to programs and variable attributes.

Program Name

The Program Name menu allows you to select the program you want to monitor/record. To get a program name to be listed on this menu, use the Programs button to access the Program Selection window and then select a program. The Program Selection window is discussed in Chapter 5, "Using the Program Selection Window".

Programs ...

The Programs button brings up the Program Selection window. The Program Selection window is discussed in Chapter 5, "Using the Program Selection Window".

Attributes ...

The Attributes button brings up the Variable Attributes window. The Variable Attributes window is discussed in Chapter 7, "Using the Variable Attributes Window".

Using the Program Scope Area

The program scope area is located just below the program name area (see Figure 6-1, “Components of the Variable Browser Window,” on page 6-1). This area allows you to select the scope of the search for variables to monitor/record.

Scope Type Checkboxes

By enabling any combination of the checkboxes in the upper left of the scope area, you can select the types of scopes to appear in the Scope list.

Functions

The Functions checkbox lets you elect to see (or not see) functions in the Scope list. This is generally not interesting when viewing an Ada program.

Packages

The Packages checkbox lets you elect to see (or not see) Ada packages in the Scope list.

Aggregates

The Aggregates checkbox lets you elect to see (or not see) aggregates, i.e., C structures and unions and Ada records, in the Scope list. For more information about aggregates, see “Aggregates” on page 2-2.

Source Files

The Source Files checkbox lets you elect to see (or not see) source files in the Scope list. This is generally not interesting when viewing an Ada program.

Components Affecting the Entire Window

One checkbox and one output field above the Scope list apply to both the Scope list and the Variable list.

Automatic Update

The Automatic Update checkbox, when selected, ensures that the variables in the Scope list and the Variable list (see below) are displayed whenever going up or down in the scope hierarchy. If this button is not selected, the list is updated with the Update Button. (See also “Invoking NightProbe” on page 2-6 and “NightStar Resources” on page B-3.)

The Automatic Update checkbox is automatically selected by default. You may change the default behavior by setting the `autoUpdate` resource to `False` (see "NightProbe Resources" on page B-4).

NOTE

If you have very large programs, you may wish to disable automatic updates which might take a long time with programs of a significant size.

Current Scope

The Current Scope field shows the name of the currently active scope. You begin in the global scope.

Scope List

The Scope list includes a text area for filtering the display plus a scrolled list with two columns of information and a text field that displays or allows entry of a scope selection.

Filter

The Filter field filters the program's symbol table and puts in the Scope list only those names meeting the filter's criteria. The default is `*`, which means that everything is displayed. The wildcard characters are the same as the ones used by the shell (e.g., `*`, `?`, etc.). Use the Update button to replenish the list with all scopes matching the current contents of the Filter field.

TIP:
To avoid seeing system library routines that begin with an underscore, use the `[!_]*` filter.

Type

The Type column of the Scope list shows the scope type of each entry in the Scope list. Possibilities include: `function`, `package`, `aggregate`, or `file`.

Scope

The Scope list contents depend on the status of the Functions, Packages, Aggregates, and Source Files checkboxes (see above) and the current scope. The display always represents findings from the symbol table of the selected program. If you specify:

Functions	then the symbol table for the selected program is searched for all functions that are defined in the current scope. The members of each source file scope are the static functions.
Packages	then the symbol table for the selected Ada program is searched for all package names.

Aggregates — then the symbol table for the selected program is searched for all C structures and unions and Ada records.

Source Files — then the names of the program's source files are displayed. (For example, in C this would be the **`.c`** and **`.h`** files.) The members of each source file scope are the static functions.

For example, if you wish to examine the static variables within a function, click on that function name in the list and click on the Down button below the list.

TIP:

For Fortran programs, click on the function names that do not have a trailing underscore.

Selection

This field displays the current function, package, aggregate, or source file name selected. You may enter a name in this field, causing NightProbe to interpret it within the current scope.

Aggregates are listed as scopes so that you may select individual items from a structure, union, or record. If the program contains an array of aggregates, the Scope list shows a pair of brackets after the name. Before descending into this aggregate scope, you must supply the array index for the desired aggregate array item by typing the index number within the brackets in the Selection field.

Current Scope Control Area

Controls below the Scope list allow you to change the current scope.

Down

The Down button makes the scope that has been selected from the Scope list be the current scope. When the current scope is changed in this way, the Scope list will display the scopes that are immediate children of the new current scope (if any), and the Variable list will display the static variables in this new current scope (if any).

Up

The Up button reverses the effect of the Down button.

Update

The Update button re-examines the symbol table for functions, packages, aggregates, or source files that fit the criteria specified in the filter and by the scope type checkboxes. This button is used if either the filter or the program name is changed. Selecting Automatic Update performs this task automatically.

Using the Variable List Area

The Variable list area is located just below the program scope area (see Figure 6-1, "Components of the Variable Browser Window," on page 6-1). This area allows you to select the variables to monitor/record. Note that to select an item in the Variable list, move the mouse pointer to the row the item is on and click. The highlighted item then appears in the Selection area.

Filter

This field filters the list of variables that are defined in the current scope and puts in the Variable list all the variables meeting the filter's criteria. The default filter criteria is `*`, which means that everything is displayed. The wild-card characters are the same as the ones used by the UNIX shell (e.g., `*`, `?`, etc.). Use the Update button to replenish the list with all scopes matching the current contents of the Filter field.

Variable

The Variable list contains a list of variables that are defined in the program, function, package, aggregate, or source file selected in the Program Scope area. This list is put through the filter in the Filter field.

Selection

This field displays the current variable selected. You may type a variable name or address in this field, causing NightProbe to interpret it within the current scope. For more information about valid input for this field, see "Using Variable Names" on page 2-1.

Info

The Info field contains information about the selected variable. The information given is as follows: language type (e.g., C, Fortran, etc.), variable type (e.g., `int`), the number of bytes, and the logical address of the first monitored location (multiple addresses can be monitored). Click on the Info Button to update this field or change the input focus from the Target Location field to any other.

Update

The update button re-examines the symbol table for variables that fit the criteria specified in the filter. This button is used if the filter, the scope, or the program name is changed. Selecting Automatic Update performs this task automatically.

Using the Control Area

The control area is at the bottom of the Variable Browser window (see Figure 6-1, "Components of the Variable Browser Window," on page 6-1) and contains the buttons that control the window.

OK — The OK button adds the variable specified in the Selection area below the Variable list to the Variable List and closes the Variable Browser window. The Variable List contains the list of variables and locations that are monitored/recorded.

Add — The Add button adds the variable specified in the Selection area to the Variable List. The Variable List contains the list of variables and locations that are monitored/recorded.

Close — The Close button closes the Variable Browser window without making any further changes to the Variable List.

Help — The Help button opens the HyperHelp viewer displaying the online help topic for the Variable Browser window. See “Getting Help” on page 2-8 for details.

7
Using the Variable Attributes Window

7
Using the Variable Attributes Window

The Variable Attributes window allows you to add variables to and set attributes for the variables listed in the Variable List of the Data Recording window (see "Using the Data Recording Window" on page 3-1). Figure 7-1 identifies the location of each of the three main areas in the Variable Attributes window.

- The Program Name Area
- The Variable Location and Information Area
- The Attributes Area
- The Control Area

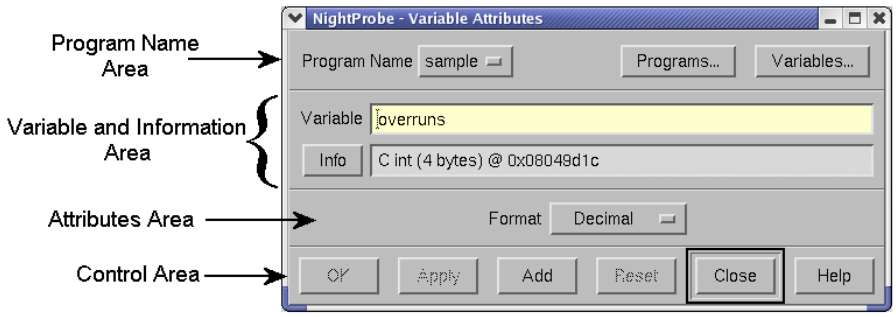

Figure 7-1. Components of the Variable Attributes Window

Using the Program Name Area

The Program Name area is located at the top of the Variable Attributes window (see Figure 7-1, "Components of the Variable Attributes Window," on page 7-1). This area provides access to programs and the symbol table browser.

Program Name

The Program Name menu allows you to select the program you want to monitor/record. To get a program name to be listed on this menu, use the Programs button to access the Program Selection window and then select a program. The Program Selection window is discussed in Chapter 5, "Using the Program Selection Window".

Programs ...

The Programs button brings up the Program Selection window. The Program Selection window is discussed in Chapter 5, "Using the Program Selection Window".

Variables ...

The Variables button brings up the Variable Browser window. The Variable Browser window is discussed in Chapter 6, "Using the Variable Browser Window".

Using the Variable Location and Information Area

The following paragraphs describe the variable specification fields below the Program Name area.

Variable

The Variable text field allows you to type in the name or address of a variable that you wish to monitor. For more information about valid input for this field, see "Using Variable Names" on page 2-1.

Info

The Info field contains information about the selected variable. The information given is as follows: language type (e.g., C, Fortran, etc.), variable type (e.g., `int`), the number of bytes, and the logical address of the first monitored location (multiple addresses can be monitored). Click on the Info Button to update this field or change the input focus from the Variable field to any other.

Using the Attributes Area

The following paragraph describes the control in the middle of the Variable Attributes window.

Format

The Format menu allows you to determine the format of the data values when they are displayed in the spreadsheet or list viewer (e.g., decimal, hexadecimal, etc.).

Using the Control Area

The control area is located along the bottom of the Variable Attributes window. The following paragraphs explain the effects the buttons have.

OK The OK button assigns the attributes to the selected variable and closes the Variable Attributes window.

Apply The Apply button assigns the attributes to the selected variable.

Add The Add button adds the variable with its attributes to the end of the Variable List.

Reset The Reset button loads the data entry fields with the information matching the variable selection in the Data Recording window or, if there is no selection, clears the data entry fields in the window.

Close The Close button closes the Variable Attributes window. The Close and Cancel button are the same, the label reads Close if there are no unsaved modifications.

Cancel The Cancel button closes the Variable Attributes window. Any values you have entered into fields are discarded. The Close and Cancel button are the same, the label reads Cancel if there are unsaved modifications.

Help The Help button opens the HyperHelp viewer displaying the online help topic for the Variable Attributes window. See “Getting Help” on page 2-8 for details.

8
Using the Output Windows

8
Using the Output Windows

NightProbe provides two windows for interactive viewing of sampled data. Both are instantiated using the Output menu of the menu bar on the Data Recording window (see “Output” on page 3-7).

The two windows, described in this chapter, are:

- The List Viewer window
- The Spreadsheet Viewer window

Using the List Viewer Window

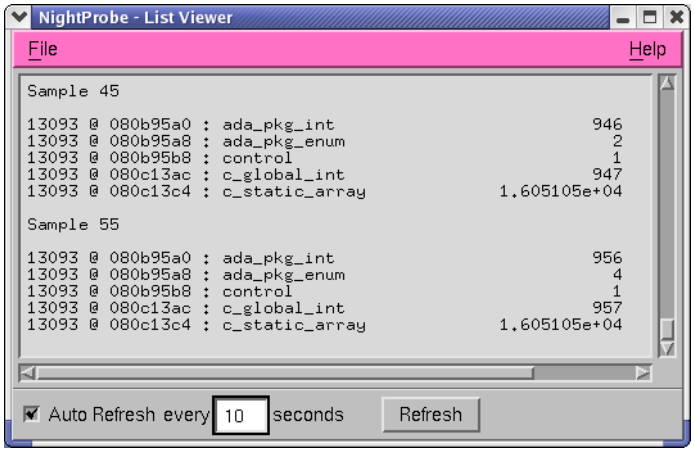

Figure 8-1. The List Viewer Window

The List Viewer window, shown in Figure 8-1, is the simpler of the two viewing windows. It allows you to:

- View a scrolled text report on the sampled data.
- View previously recorded data files as text within a scrolled window.
- Display sampled data after every sample, after a set number of samples, or upon demand.

The List Viewer window contains

- The menu bar (see “Using the Menu Bar” on page 8-2)
- The scrolled text Viewing Area

- The Control Area (see "Using the Control Area" on page 8-4)

Using the Menu Bar

The List Viewer window menu bar contains File and Help menus. These are described in the next two sections.

File

Mnemonic: F

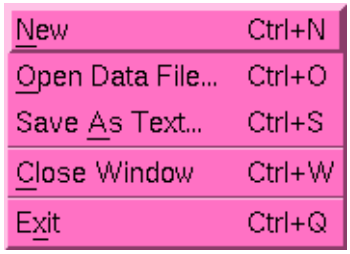

Figure 8-2. File menu

New
Mnemonic: N
Accelerator: <Control><N>

This option allows you to clear the scrolled text viewing area. If you are monitoring a running program, you will not be able to recall the erased information in this window.

Open Data File...
Mnemonic: O
Accelerator: <Control><O>

This option allows you to open a data file that was created using the To File option of the Output menu. The data file will be translated to ASCII text and displayed in the scrolled text viewing area.

Save As...
Mnemonic: A
Accelerator: <Control><S>

This option allows you to save the current contents of the text area (including what is not visible in the viewing area) to a file. You will be presented with a file selection dialog with which to choose a file name.

Close Window
Mnemonic: C
Accelerator: <Control><W>

Using this option closes this window and removes it from the Output list.

Exit
Mnemonic: X
Accelerator: <Control><Q>

This option exits NightProbe.

Help

Mnemonic: H

The Help menu operates exactly like the menu provided in the Data Recording window. It lists a number of topics on which help is available, and selecting any topic will display a help window. See "Getting Help" on page 2-8 for details.

Using the Control Area

The control area appears at the bottom of the List Viewer window. It allows you to control when new information is added to the viewing area.

Auto Refresh

The Auto Refresh checkbox and text entry field control how often the sampled values are displayed. The List Viewer is designed for displaying values at human-readable rates, not necessarily displaying all sampled values (especially if the sampling rate is extremely fast).

Refresh

The Refresh button can be used when Auto Refresh is turned off. The Refresh button gets the most recent sample taken and displays it in the List Viewer window. Note that the Refresh button does not cause the sampler to take a new sample or record a sample to a file.

Using the Spreadsheet Viewer Window

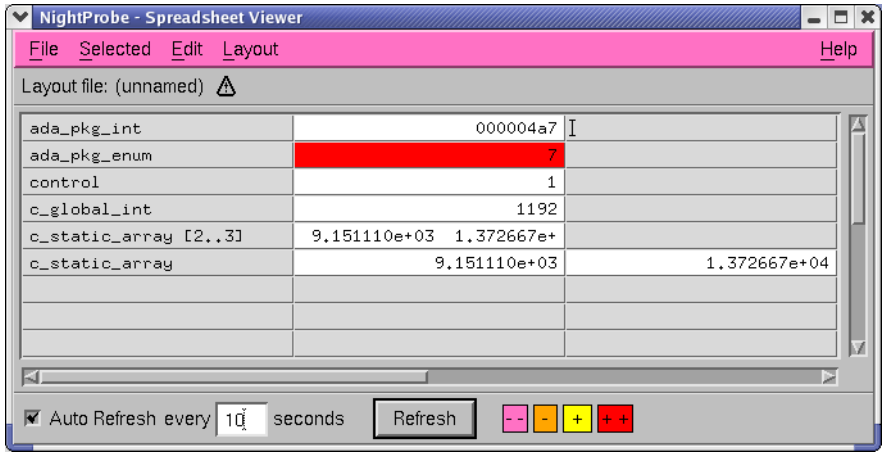

Figure 8-3. The Spreadsheet Viewer Window

The Spreadsheet Viewer window, shown in Figure 8-3, provides for both viewing and modification of sampled data. It does the following:

- Allows for flexible placement of data values and labels within a spreadsheet with user-defined number and sizes of rows and columns.
- Allows selection and modification of more than one cell at a time.
- Allows for the spreadsheet layout to be saved and restored.
- Displays sampled data after every sample, after a set number of samples, or upon demand.
- Allows modification of data simply by entering the new value into the spreadsheet cell.

For a tutorial on using the spreadsheet viewer, see “Using the Spreadsheet” on page A-5 for an example using an Ada program or “Variable Browsing” on page A-14 for an example using a C++ program.

The Spreadsheet Viewer window contains

- The Menu Bar (see “Using the Menu Bar” on page 8-6)
- The Layout Configuration Status Area (see “Using the Layout Configuration Status Area” on page 8-13)
- The Spreadsheet Viewing Area (see “Using the Spreadsheet Viewing Area” on page 8-13)
- The Control Area (see “Using the Control Area” on page 8-14)

Using the Menu Bar

The Spreadsheet Viewer window menu bar contains the following menus.

- File
- Selected
- Edit
- Layout
- Help

Each menu is described in the sections that follow.

File

Mnemonic: F

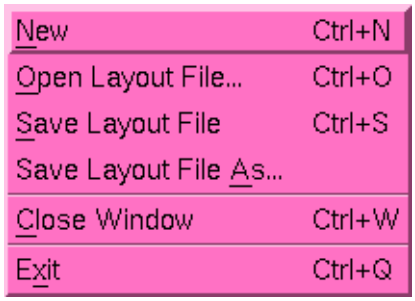

Figure 8-4. File menu

The File menu allows you to load a previously-saved layout configuration, save the current layout configuration to a file, or get a new, clean layout configuration. The File menu also contains the means to exit NightProbe. The following paragraphs describe the options on the File menu in more detail.

New
Mnemonic: N
Accelerator: <Control><N>

This option allows you to clear the cells in the spreadsheet and the layout configuration. If you are monitoring a running program, you will not be able to recall the erased information in this window.

Open Layout File...
Mnemonic: O
Accelerator: <Control><O>

This option allows you to open a layout file that was created using the Save Layout File or Save Layout File As options. The layout file saves all information

about how the cells in the spreadsheet are used to display the sampled data. You will be presented with a file selection dialog with which to choose a file name.

Save Layout File
Mnemonic: S
Accelerator: <Control><S>

This option allows you to save the spreadsheet layout configuration to the current layout file.

Save Layout File As...
Mnemonic: A

This option allows you to save the spreadsheet layout configuration to a file. You will be presented with a file selection dialog with which to choose a file name.

You may also save the image of the currently selected cells as text information to a file by selecting the Save As Text... item from the Selected menu (see "Save as Text" on page 8-10).

Close Window
Mnemonic: C
Accelerator: <Control><W>

Using this option closes this window and removes it from the Output list.

Exit
Mnemonic: X
Accelerator: <Control><Q>

This option exits NightProbe.

Selected

Mnemonic: S

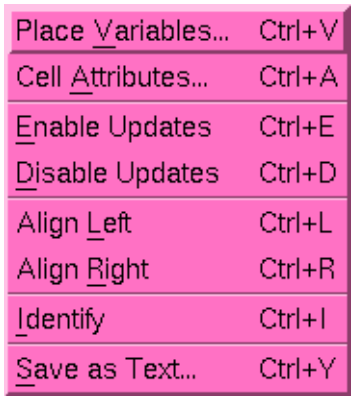

Figure 8-5. Selected menu

The Selected menu operates on a group of spreadsheet cells that have already been selected. Select cells by clicking mouse button 1 with the mouse pointer over the cell, or by dragging the mouse pointer across a rectangle of cells while mouse button 1 is depressed. Selected cells will be highlighted.

Place Variables

Mnemonic: V
Accelerator: <Control><V>

Selecting the Place Variables menu option displays the Spreadsheet Variables window. The figure below shows the Spreadsheet Variables window. This window contains controls to place variable cells onto a spreadsheet.

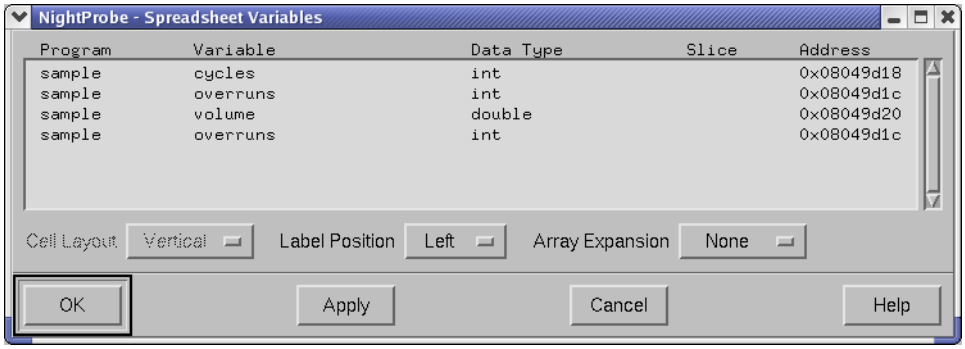

Figure 8-6. The Spreadsheet Variables Window

To use this window, first select a cell in the spreadsheet by clicking on it with the mouse. This will be the starting cell for placing variables.

Next, select the variable or variables you wish to place from the list in the Variable Placement window. You may place more than one variable at a time.

Below the Variable List are three option menus for controlling placement.

The Cell Layout menu is used whenever you place more than one variable at once. It specifies whether to place the variables going down from the starting cell (Vertical layout) or going across the spreadsheet (Horizontal layout).

The Label Position menu controls an optional label cell which will be placed along with the variable cell. The label cell will contain the name of the variable. You can choose None for no label, or a position relative to the Variable cell (Top, Bottom, Left, or Right).

The Array Expansion menu specifies what to do with variables that represent arrays. Selecting None will place all array elements in a single cell. Selecting Horizontal or Vertical will place each element in its own cell, laid out in the specified direction.

When you have selected your variables and options, click the OK button to place them on the spreadsheet and close the window, or the Apply button to place the variables and leave the window open. The Close button closes the window without placing any variables.

Cell Attributes

Mnemonic: A
Accelerator: <Control><A>

Selecting the Cell Attributes menu option displays the Cell Attributes window. The figure below shows the Cell Attributes window. This window allows you to view and change various attributes associated with a spreadsheet cell. The window is only active when a Variable cell is selected.

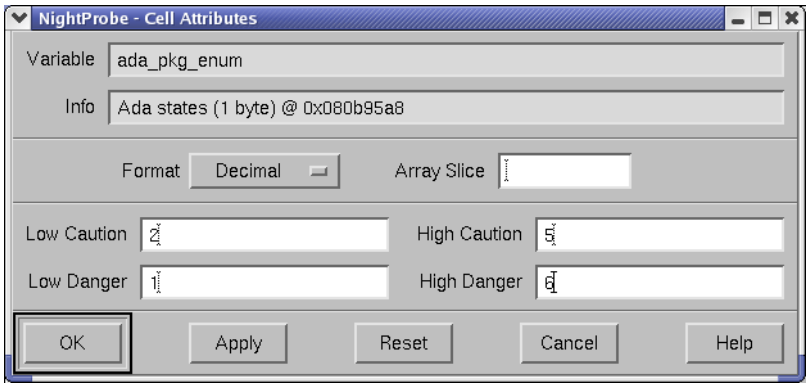

Figure 8-7. The Cell Attributes Window

Variable and Info

These fields show the variable name and information about the selected cell's variable. They are read-only text fields. To change a cell's variable or to create new variable cells, use the Place Variables option under the Selected menu.

Format

This option menu allows you to choose the output format for the cell.

Array Slice

This field allows you to specify the array indices, if the variable is an array, to display in the cell. You may specify a single index number or a range of numbers such as `3..7` or `3:7` for elements 3 through 7, inclusive. For more information about array slices, see "Array Slices" on page 2-2.

Low Caution, High Caution, Low Danger, High Danger

These fields specify limits for the specified variable. They are only appropriate for use with scalar types. When the value in the cell goes outside these boundaries, the cell's background color will change. You can define the colors of these cells with resources described in "NightStar Resources" on page B-3.

Clicking the OK button applies any changes you have made to the cell and closes the window. Clicking the Apply button applies the changes without closing the window. Clicking the Cancel button closes the window without making any changes (this button will be labeled Close if no changes were made).

Enable Updates

Mnemonic: E
Accelerator: <Control><E>

Updates the selected cells when new samples are displayed. This reverses the action of the Disable Updates selection.

Disable Updates

Mnemonic: D
Accelerator: <Control><D>

Does not update the selected cells when new samples are displayed. These cells have a darker background color than enabled cells. You would use this option to hold on to a data value in the display while allowing the sampler to continue running and updating other values.

Align Left

Mnemonic: L
Accelerator: <Control><L>

Data values in the selected cells will be aligned with the left edge of the cell.

Align Right

Mnemonic: R
Accelerator: <Control><R>

Data values in the selected cells will be aligned with the right edge of the cell.

Identify

Mnemonic: I
Accelerator: <Control><I>

Displays the variable names or addresses with which the selected cells are associated. The next update will revert to displaying the data values.

Save as Text

Mnemonic: S
Accelerator: <Control><Y>

Writes as text information to a file the image of the currently selected cells. You will be presented with a file selection dialog with which to choose a file name.

You may also save the spreadsheet layout configuration to a file by selecting the Save Layout File As… item from the File menu (see "File" on page 8-6).

Edit

Mnemonic: E

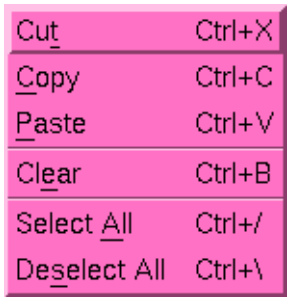

Figure 8-8. Edit menu

The Edit menu provides the means to perform some editing operations on the cells and the layout configuration.

Cut
Mnemonic: T
Accelerator: <Control><X>

Removes the layout configuration information from the selected cells and stores that information in the layout clipboard. The selected cells are cleared.

Copy
Mnemonic: C
Accelerator: <Control><C>

Copies the layout configuration information from the selected cells and stores that information in the layout clipboard. The selected cells are unaffected.

Paste
Mnemonic: P
Accelerator: <Control><V>

Inserts the contents of the layout clipboard at the current selection point. The layout clipboard retains its information and can be used again.

Clear
Mnemonic: E
Accelerator: <Control><B>

Clears the selected cells.

Select All
Mnemonic: A
Accelerator: <Control></>

Puts all cells into the “selected” state for other operations.

Deselect All
Mnemonic: S
Accelerator: <Control><\>

Puts all cells into the "unselected" state.

Layout

Mnemonic: L

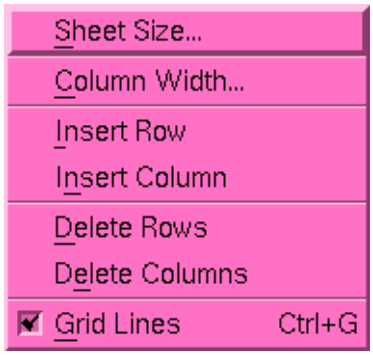

Figure 8-9. Layout menu

The Layout menu provides controls for organizing the display area into a rectangular grid of spreadsheet cells.

Sheet Size
Mnemonic: S

Displays a dialog that allows you to specify the number of rows and columns in the spreadsheet.

Column Width
Mnemonic: C

Displays a dialog that allows definition of the width (in character positions) of the currently selected columns.

Insert Row
Mnemonic: I

Inserts one row above the current selection point.

Insert Column
Mnemonic: N

Inserts one column to the left of the current selection point.

Delete Rows
Mnemonic: D

Deletes the selected rows.

Delete Columns
Mnemonic: E

Deletes the selected columns.

Grid Lines
Mnemonic: G

Enables or disables the lines delineating the spreadsheet cells by clicking on this toggle button.

Help

Mnemonic: H

The Help menu operates exactly like the menu provided in the Data Recording window. It lists a number of topics on which help is available, and selecting any topic will display a help window. See "Getting Help" on page 2-8 for details.

Using the Layout Configuration Status Area

The layout configuration status area displays the file name of the layout file, if any has been specified. It also indicates via an icon at the end of the name if there are unsaved modifications to the layout configuration.

Using the Spreadsheet Viewing Area

The spreadsheet viewing area is composed of rows and columns of spreadsheet cells. Each cell can contain either a text label or the contents of a monitored data location. Enter text labels merely by selecting the cell and typing the label. Use the Place Variables menu option (from the Selected menu) to associate a selected cell with a data location.

Data values in the spreadsheet are updated according to the specifications of the Control Area (see "Using the Control Area" on page 8-14).

You may use the scroll bars below and to the right of the viewing area to see cells that are not in the current display window.

Once NightProbe has been connected to the running programs, you can use the spreadsheet to modify data values. To modify a variable's value, click on the variable cell, type a new value, and press the <Enter> key. Values may be entered as decimal numbers, octal numbers when preceded by `0`, hexadecimal numbers when preceded by `0x`, or character strings when preceded by a double quote (").

Using the Control Area

The control area appears at the bottom of the window. It allows you to control when new information is added to the viewing area. In addition, it contains a legend indicating the two caution and two danger colors.

Auto Update

The Auto Update checkbox and text entry field allow you to display every *n*th sample, where *n* is a value you select. This is useful if you want to monitor a program while it is running but the sampler is recording values so fast they cannot be seen. (See also "Invoking NightProbe" on page 2-6 and "NightStar Resources" on page B-3.)

Update

The Update button gets the most recent sample taken and displays it in the Spreadsheet Viewer window. The Update button does not cause the sampler to take a new sample or record a sample to a file.

Cell Color Legend

These four squares are a legend indicating the colors used when the value in a cell exceeds its defined limits.

– –	represents Low Danger
–	represents Low Caution
+	represents High Caution
+ +	represents High Danger

Limits for a cell can be set using the Cell Attributes window. You can define the colors of these cells with resources described in "NightStar Resources" on page B-3.

9
Using the NightProbe API

9
Using the NightProbe API

The NightProbe Application Programming Interface provides a basic interface to the data produced by NightProbe.

This API can be used with data recording output generated by NightProbe using the To File and To Program output methods (see "To File" on page 3-8 and "To Program" on page 3-14).

The following sections describe the data structures and functions that comprise the NightProbe API.

Sample programs using these data structures and functions are also provided (see "Sample Programs" on page 9-11).

NightProbe Data Format

This section describes the general format of data generated by NightProbe sampling. This format is used when the user selects either the To File or To Program output method (see "To File" on page 3-8 and "To Program" on page 3-14).

The NightProbe Application Programming Interface (see "NightProbe Application Programming Interface" on page 9-2) allows the user to open a previously recorded file and decode the individual data items or to consume the data as it is being generated by NightProbe. In either case, the incoming data is referred to as a datastream. In the case of To File, the user opens the file and uses the resultant file descriptor with the following API calls to decode the data. In the case of To Program, a user program is launched from

NightProbe and its **stdin** file descriptor is set to the read end of a socket or pipe. The user program then specifies **stdin** to the following API calls to decode the data.

The following diagram describes the general layout of a datastream:

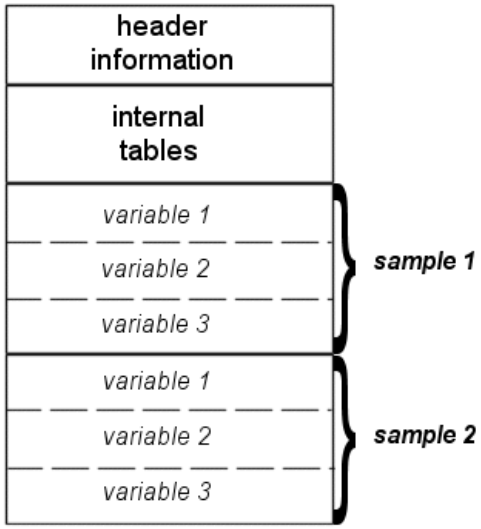

Figure 9-1. Structure of NightProbe datastream

The API calls below provide a simple interface for obtaining information about the programs from which the data was obtained, information about the variables within those programs, and individual data *sample*s.

NightProbe Application Programming Interface

The NightProbe Application Programming Interface consists of a number of data structures (see "Data Structures" on page 9-2) and functions (see "Functions" on page 9-6).

Data Structures

The following data structures are part of the NightProbe Application Programming Interface:

- `np_handle` (see "np_handle" on page 9-3)
- `np_header` (see "np_header" on page 9-3)
- `np_process` (see "np_process" on page 9-3)
- `np_item` (see "np_item" on page 9-4)
- `np_type` (see "np_type" on page 9-5)

See “Functions” on page 9-6 for information about the functions available in the NightProbe API.

np_handle

`np_handle` is a unique integer value denoting a single NightProbe datastream.

```
typedef int np_handle;
```

See “Data Structures” on page 9-2 for other data structures included in the NightProbe API.

np_header

`np_header` is a structure which is used to describe the processes and items from which data in the NightProbe datastream originates. This information is needed to interpret the sample data returned by `np_read()`.

```
typedef struct {
   int           num_items;
   int           num_processes;
   int           sample_size;
   np_process  * processes;
   np_item     * items;
} np_header ;
```

See “Data Structures” on page 9-2 for other data structures included in the NightProbe API.

SEE ALSO

- “np_read()” on page 9-8

np_process

`np_process` is a structure which contains information about a particular process from which real-time data originates.

```
typedef struct np_process np_process;
struct np_process {
   int          pid;
   char       * name;
   np_process * link;
};
```

See “Data Structures” on page 9-2 for other data structures included in the NightProbe API.

np_item

`np_item` is a structure that describes a single data item present in the NightProbe datastream.

```
typedef struct np_item np_item;
struct np_item {
   char        * name;     // name of item
   int           offset;   // byte offset within each sample
   unsigned      size;     // atomic size in bytes
   unsigned      count;    // number of atoms
   np_type       type;     // data type
   unsigned      event_id; // NightTrace event ID for item
   np_process  * process;  // process info
   np_item     * link;     // next item pointer
};
```

The item occupies `count` instances of `size` bytes begining at `offset` within the sample data returned by `np_read()`.

See "Data Structures" on page 9-2 for other data structures included in the NightProbe API.

SEE ALSO

- "np_read()" on page 9-8
- "np_type" on page 9-5

np_type

The `np_type` enumeration in the `np_item` structure may be used (along with `size`) in order to determine an appropriate format for displaying a value from the sample buffer.

```
typedef enum np_type_code {
   NP_VOID_TYPE,                 /* void                          */
   NP_CHAR_TYPE,                 /* signed byte character         */
   NP_UNSIGNED_CHAR_TYPE,        /* unsigned byte character       */
   NP_SHORT_INT_TYPE,            /* signed short int              */
   NP_UNSIGNED_SHORT_INT_TYPE,   /* unsigned short int            */
   NP_INT_TYPE,                  /* signed int                    */
   NP_UNSIGNED_INT_TYPE,         /* unsigned int                  */
   NP_LONG_INT_TYPE,             /* signed long int               */
   NP_UNSIGNED_LONG_INT_TYPE,    /* unsigned long int             */
   NP_FLOAT_TYPE,                /* single precision float        */
   NP_DOUBLE_TYPE,               /* double precision float        */
   NP_LONG_DOUBLE_TYPE,          /* long double precision float   */
   NP_SHORT_LOGICAL_TYPE,        /* short logical (boolean)       */
   NP_LOGICAL_TYPE,              /* logical (boolean)             */
   NP_COMPLEX_TYPE,              /* Fortran complex type          */
   NP_DOUBLE_COMPLEX_TYPE,       /* Fortran double complex        */
   NP_POINTER_TYPE,              /* Pointer to unspecified type   */
   NP_FIXED_POINT_TYPE,          /* fixed point                   */
   NP_EXCEPTION_TYPE,            /* exception                     */
   NP_STRUCTURE_BYTES            /* structure bytes               */
}  np_type ;
```

See “Data Structures” on page 9-2 for other data structures included in the NightProbe API.

SEE ALSO

- “np_item” on page 9-4

Functions

The following functions are part of the NightProbe Application Programming Interface:

- `np_open()` (see "np_process" on page 9-3)
- `np_avail()` (see "np_avail()" on page 9-7)
- `np_read()` (see "np_read()" on page 9-8)
- `np_close()` (see "np_close()" on page 9-9)
- `np_error()` (see "np_error()" on page 9-10)

np_open()

`np_open()` is used to open and initialize an input NightProbe datastream on an open file descriptor.

SYNTAX

```
int np_open (int fd, np_header *header, np_handle *handle);
```

PARAMETERS

fd — file descriptor associated with the file created using the To File output method (see "To File" on page 3-8) which contains the data recording output

If data recording output is streamed directly from NightProbe using the To Program output method (see "To Program" on page 3-14), *fd* should be set to the **`stdin`** file descriptor, `0`.

header — structure to contain information describing the processes from which the NightProbe data originates, as well as the number, names and types of the items appearing in the NightProbe datastream

handle — a unique value denoting the open NightProbe datastream

RETURN VALUES

0 — indicates successful completion

-1 — indicates a failure

handle contains a value which may be passed to `np_error()` to obtain a diagnotic message describing the failure

See "Functions" on page 9-6 for other functions included in the NightProbe API.

SEE ALSO

- "np_header" on page 9-3
- "np_handle" on page 9-3
- "np_error()" on page 9-10

np_avail()

`np_avail()` is used to check a NightProbe datastream for available data items.

SYNTAX

```
int np_avail (np_handle handle);
```

PARAMETERS

handle	value (obtained from `np_open()`) which identifies the NightProbe datastream of interest

RETURN VALUES

0	if data is not currently available on the NightProbe datastream and `np_read()` would block
> 0	if data is currently available for `np_read()`
-1	indicates a failure If *handle* is non-zero, `np_error()` may be called to obtain a diagnostic message describing the failure.

See "Functions" on page 9-6 for other functions included in the NightProbe API.

SEE ALSO

- "np_open()" on page 9-6
- "np_read()" on page 9-8
- "np_error()" on page 9-10

np_read()

Read a single data sample from the NightProbe datastream.

SYNTAX

```
int np_read (np_handle handle, void *sample);
```

PARAMETERS

handle — value (obtained from `np_open()`) which identifies the NightProbe datastream of interest

sample — upon successful completion, *sample* contains the NightProbe entire sample data

To get at individual data items, use the information from the `np_header` structure returned from `np_open()`. For each item, retrieve the appropriate number of bytes (as specified by `size` in the `np_item` structure associated with that item) offset from the beginning of the sample buffer (as specified by `offset` in the `np_item` structure associated with that item)

See "Sample Programs" on page 9-11 for examples.

RETURN VALUES

> 0	value represents the number of bytes in the sample obtained
0	if end-of-file (EOF) was encountered on the NightProbe datastream
-1	indicates a failure

If *handle* is non-zero, `np_error()` may be called to obtain a diagnostic message describing the failure.

NOTE

`np_read()` will block waiting for data to become available on the datastream if data is not immediately available. If time is critical and a blocking read is not desired, use `np_avail()` to first check if data is available prior to reading.

See "Functions" on page 9-6 for other functions included in the NightProbe API.

SEE ALSO

- "np_header" on page 9-3

- "np_item" on page 9-4
- "np_open()" on page 9-6
- "np_avail()" on page 9-7
- "np_error()" on page 9-10

np_close()

Close a NightProbe datastream.

SYNTAX

```
void np_close (np_handle handle);
```

PARAMETERS

handle — value (obtained from `np_open()`) which identifies the NightProbe datastream of interest

Upon completion, *handle* no longer refers to an open NightProbe datastream.

NOTE

No further diagnostic messages are available from `np_error()` after calling `np_close()`.

Furthermore, the file descriptor passed to `np_open()` remains open after the `np_close()` call. The NightProbe datastream is logically closed, but the associated file descriptor remains open. **`close(2)`** must be called to close the file descriptor as well, if desired.

See "Functions" on page 9-6 for other functions included in the NightProbe API.

SEE ALSO

- "np_open()" on page 9-6
- "np_error()" on page 9-10

np_error()

Return a diagnostic message describing the most recent failure encountered by a prior call to `np_open()`, `np_avail()`, or `np_read()`.

SYNTAX

```
char * np_error (np_handle handle);
```

PARAMETERS

handle	value (obtained from `np_open()`) which identifies the NightProbe datastream of interest

See "Functions" on page 9-6 for other functions included in the NightProbe API.

SEE ALSO

- "np_open()" on page 9-6
- "np_avail()" on page 9-7
- "np_read()" on page 9-8

Sample Programs

The following programs are given as examples of how to use the NightProbe Application Programming Interface (see “NightProbe Application Programming Interface” on page 9-2).

program_output_test.c

This program uses the NightProbe API to process a NightProbe data sample.

See “program_output_test.c” on page 9-11.

program_output_fbs_test.c

This program uses the NightProbe API to process a NightProbe data sample but uses a frequency-based scheduler in order to coordinate data recording activity so as to minimize interference with the probed application.

See “program_output_fbs_test.c” on page 9-14.

program_output_test.c

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3 #include <stdlib.h>
 4 #include <fcntl.h>
 5 #include <errno.h>
 6 #include <string.h>
 7 #include <nprobe.h>
 8
 9
10
11 int    cycles   = 0 ;
12 int    overruns = 0 ;
13 char * sample;
14
15
16 //
17 // Perform the work of consuming a single Data Recording sample from NightProbe.
18 //
19 int
20 work (FILE * ofile, np_handle h, np_header * hdr) {
21    np_item * i;
22    int status;
23    int n;
24    char * ptr;
25
26
27    // Read one sample, which may contain data for multiple processes
28    // and variables.
29    status = np_read (h, sample);
30    if (status <= 0)  {
```

```
31       return status;
32    }
33
34    cycles++ ;
35
36    fprintf (ofile, "\n");
37    for (i=hdr->items; i; i=i->link) {
38       fprintf (ofile, "item: %25s :", i->name);
39       ptr = sample + i->offset;
40       for (n=0; n<i->count; ++n) {
41
42          //
43          // Note that this simple example assumes type/format from
44          // the size of the data item.  The 'i->type' field should
45          // be taken into account for a more accurate means of
46          // determining the data format.
47          //
48          switch (i->size) {
49          case 1:
50             fprintf (ofile, " 0x%1x", ((char*)ptr)[n]);
51             break;
52          case 2:
53             fprintf (ofile, " 0x%1x", ((unsigned short*)ptr)[n]);
54             break;
55          case 4:
56             fprintf (ofile, " 0x%1x", ((unsigned*)ptr)[n]);
57             break;
58          case 8:
59             fprintf (ofile, " %lf", ((double*)ptr)[n]);
60             break;
61          default:
62             fprintf (ofile, " <size=%d>", i->size);
63          }
64       }
65       fprintf (ofile, "\n");
66    }
67    fflush (ofile) ;
68
69    return 1 ;
70 }
71
72
73
74
75 int
76 main (int argc, char *argv[])
77 {
78    np_handle h;
79    np_header hdr;
80    np_process * p;
81    np_item * i;
82    int fd;
83    int status;
84    FILE *   ofile = stdout ;
85
86
87    fd = 0 ; // stdin
88
89    status = np_open (fd, &hdr, &h);
```

```
 90    if (status) {
 91       fprintf (stderr, "np_open: \"%s\"\n", np_error(h));
 92       exit(1);
 93    }
 94
 95    sample = (char *) malloc(hdr.sample_size);
 96    if (sample == NULL) {
 97       fprintf (stderr, "insufficient memory to allocate sample buffer\n");
 98       exit(1);
 99    }
100
101    for (p=hdr.processes; p; p=p->link) {
102       fprintf (ofile, "process: %s (%d)\n", p->name, p->pid);
103    }
104    fprintf (ofile, "\n");
105
106    for (i=hdr.items; i; i=i->link) {
107       fprintf (ofile, "item: %s (%s) size=%d count=%d type=%d\n",
108                i->name, i->process->name, i->size, i->count, i->type);
109    }
110    fprintf (ofile, "\n");
111
112    for (;;) {
113       status = work (ofile, h, &hdr)  ;
114       if (status <= 0) break ;
115    }
116
117    fprintf (ofile, "Data Recording done: %d cycles fired, %d overruns\n",
118             cycles, overruns) ;
119
120    if (ofile != stdout) {
121       fclose (ofile) ;
122    }
123
124    if (status < 0) {
125       fprintf (stderr, "np_read: \"%s\"\n", np_error(h));
126    }
127
128    np_close (h);
129
130    // At this point, file descriptor 0 remains open, but is no
131    // longer a NightProbe Data File/Stream.
132 }
```

program_output_fbs_test.c

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3 #include <stdlib.h>
 4 #include <fcntl.h>
 5 #include <errno.h>
 6 #include <string.h>
 7 #include <nprobe.h>
 8
 9
10
11 int    cycles   = 0 ;
12 int    overruns = 0 ;
13 char * sample;
14
15
16 //
17 // Perform the work of consuming Data Recording samples from NightProbe.
18 // This function is called once every time the fbswait() system call returns
19 // successfully.
20 //
21 int
22 work (FILE * ofile, np_handle h, np_header * hdr) {
23    np_item * i;
24    int status;
25    int n;
26    char * ptr;
27
28
29    //
30    // 0, 1, or >1 trigger events may haveoccurred since we last work()ed.
31    // Check whether data is available, and process it as long as new
32    // data is already available within this work cycle.
33    //
34    // A more sophisticated program would limit the number of np_read() calls
35    // per work cycle based upon how much time is left in the current cycle.
36    //
37    while (np_avail (h)) {
38
39       // Read one sample, which may contain data for multiple processes
40       // and variables.
41       status = np_read (h, sample);
42       if (status <= 0)  {
43          return status;
44       }
45
46       cycles++;
47
48       fprintf (ofile, "\n");
49       for (i=hdr->items; i; i=i->link) {
50          fprintf (ofile, "item: %25s :", i->name);
51          ptr = sample + i->offset;
52          for (n=0; n<i->count; ++n) {
53
54             //
55             // Note that this simple example assumes type/format from
56             // the size of the data item.  The 'i->type' field should
```

```
57             // be taken into account for a more accurate means of
58             // determining the data format.
59             //
60             switch (i->size) {
61             case 1:
62                fprintf (ofile, " 0x%1x", ((char*)ptr)[n]);
63                break;
64             case 2:
65                fprintf (ofile, " 0x%1x", ((unsigned short*)ptr)[n]);
66                break;
67             case 4:
68                fprintf (ofile, " 0x%1x", ((unsigned*)ptr)[n]);
69                break;
70             case 8:
71                fprintf (ofile, " %lf", ((double*)ptr)[n]);
72                break;
73             default:
74                fprintf (ofile, " <size=%d>", i->size);
75             }
76          }
77          fprintf (ofile, "\n");
78       }
79       fflush (ofile);
80    }
81
82    return 1;
83 }
84
85
86
87
88 int
89 main (int argc, char *argv[])
90 {
91    np_handle h;
92    np_header hdr;
93    np_process * p;
94    np_item * i;
95    int fd;
96    int status;
97    FILE *   ofile = stdout ;
98
99
100 #ifdef linux
101    if (!fbsavail()) {
102       fprintf (ofile, "fbsavail() reports No FBS on this target\n") ;
103       fclose (ofile) ;
104       exit (1) ;
105    }
106 #endif
107
108
109
110    fd = 0 ; // stdin
111
112    status = np_open (fd, &hdr, &h);
113    if (status) {
114       fprintf (stderr, "np_open: \"%s\"\n", np_error(h));
115       exit(1);
```

```
116    }
117
118    sample = (char *) malloc(hdr.sample_size);
119    if (sample == NULL) {
120       fprintf (stderr, "insufficient memory to allocate sample buffer\n");
121       exit(1);
122    }
123
124    for (p=hdr.processes; p; p=p->link) {
125       fprintf (ofile, "process: %s (%d)\n", p->name, p->pid);
126    }
127    fprintf (ofile, "\n");
128
129    for (i=hdr.items; i; i=i->link) {
130       fprintf (ofile, "item: %s (%s) size=%d count=%d type=%d\n",
131               i->name, i->process->name, i->size, i->count, i->type);
132    }
133    fprintf (ofile, "\n");
134
135    for (;;) {
136       //
137       // We wait till the Concurrent FBS wakes us up at the time which is
138       // appropriate for performing data recording.  This program must be
139       // scheduled on the FBS, but doing so allows the scheduling of data
140       // recording activity at a time that won't disturb other critical
141       // application cycles.
142       //
143       int stat = fbswait() ;
144
145       // Diagnose the return value from fbswait()
146       if (stat < 0) {
147          switch (stat) {
148          case -1:
149             if (errno == ENOENT) {
150                fprintf (ofile,
151                         "%s has been removed from the scheduler\n", argv[0]) ;
152             } else {
153                fprintf (ofile, "fbs_wait(3) failed on cycle %d: "
154                              "errno is %d (%s)\n",
155                         cycles, errno, strerror (errno)) ;
156             }
157             break ;
158          default:
159             fprintf (ofile, "fbs_wait(3) returned unexpected %d on cycle %d\n",
160                      stat, cycles) ;
161             break ;
162          }
163
164          break ;
165       }
166
167       switch (stat) {
168       case 0:
169          break ;
170       case 1:
171          fprintf (ofile, "fbstrig(2) caused sim to fire: cycle %d\n", cycles) ;
172          break ;
173       case 2:
174          fprintf (ofile, "soft overrun %d detected on cycle %d\n",
```

```
175                   ++overruns, cycles) ;
176          break ;
177       }
178
179       status = work (ofile, h, &hdr);
180       if (status <= 0) {
181          break;
182       }
183    }
184
185    fprintf (ofile, "Data Recording done: %d cycles fired, %d overruns\n",
186             cycles, overruns) ;
187
188    if (ofile != stdout) {
189       fclose (ofile) ;
190    }
191
192    if (status < 0) {
193       fprintf (stderr, "np_read: \"%s\"\n", np_error(h));
194    }
195
196    np_close (h);
197
198    // At this point, file descriptor 0 remains open, but is no
199    // longer a NightProbe Data File/Stream.
200 }
```

A
Tutorials

This section contains two separate tutorials which demonstrate the commonly used features of NightProbe:

- Ada Tutorial, which probes an Ada program with calls to C functions
- “C++ Tutorial” on page A-13, which probes a program written in C++

Ada Tutorial

This tutorial demonstrates some of the commonly used features of NightProbe including:

- creating and selecting a program
- variable browsing
- spreadsheet use

The supplied tutorial programs declare and initialize static and dynamic variables. Some of the variables are scalars, some are arrays, and some are aggregates.

The tutorial files are in the **`/usr/lib/NightProbe/Tutorial`** directory. Source listings of these files are in:

- “C Sample - c_sample.c” on page A-12
- “Ada Sample - ada_sample.a” on page A-10.

Creating and Selecting a Program

1. The source code for the sample program used in this tutorial can be found in the **`/usr/lib/NightProbe/Tutorial`** directory and is included at the end of this chapter for reference. The sample program contains C and Ada code.
2. Copy the source files:
   - **`ada_sample.a`**
   - **`c_sample.c`**

from **/usr/lib/NightProbe/Tutorial** to a working directory and build the program using the following command(s):

```
cc -g -c c_sample.c
PATH=$PATH:/usr/ada/bin
a.mkenv -g
a.intro ada_sample.a
a.partition -create active ada_sample
a.build ada_sample
```

3. Invoke NightProbe with the following command:

   ```
   nprobe &
   ```

   NightProbe displays the Data Recording window.

4. Invoke the sample program with the following command:

   ```
   ./ada_sample
   ```

5. In the Data Recording window, press the Programs button.

   NightProbe displays the Program Selection window.

6. In the Program Selection window, press the Processes button and select the executing process `ada_sample` as shown in the list of processes associated with your user. Press OK.

7. In the Program Selection window, press the Add button.

   You will see your program in the Program List display area of the Program Selection window.

8. Press Close to close the Program Selection window.

Variable Browsing

The following sections provide an example of the use of the Variable Browser window. For more information about the Variable Browser window, see Chapter 6, "Using the Variable Browser Window".

Scopes and Variables

1. In the Data Recording window, press the Variables button.

   NightProbe displays the Variable Browser window. When the Variable Browser window first appears, the Current Scope is set to Global.

   The Variable Browser window is separated into two major panes which describe Scopes (the upper pane) and the Variables (the lower pane) within the selected scope. Scopes include source files, C functions, Ada packages and an anonymous scope denoted Global which contains all the other outermost scopes and all C global variables. Whenever a scope is selected in the upper pane of the Variable

Browser window, any scopes nested within it are displayed in the Scopes list within that pane. Additionally, any variables associated with the selected scope are displayed in the Variables list area of the lower pane below.

IMPORTANT

Browsing is most easily accomplished when the Automatic Update checkbox is checked. When this box is checked, selection of a scope causes automatic updating of the lists associated with the selected scope. When the Automatic Update checkbox is cleared, you must press the Update button associated with each list to refresh their contents even if you change scopes.

Browsing Scopes

1. In the Variable Browser window, enable automatic updates by ensuring the Automatic Update checkbox is checked.
2. The list of scopes nested inside the Global scope include the following, among others:
   a. `ada_pkg` which is a package of global data from the Ada source code
   b. `c_routine` which contains static data items from the C source code
3. Browsing Ada packages

   All Ada variables eligible for monitoring are contained inside Ada packages.

   a. Select package `ada_pkg` from the list of Scopes and press the Down button. Alternatively, double-clicking an item in the Scopes list is equivalent to selecting it and pressing Down.

      You see that the Current Scope is `ada_pkg` and that two scopes nested inside `ada_pkg` are displayed in the list of scopes:

      1. the aggregate `ada_pkg_record`
      2. the package `nested`

      Also note the list of variables contained in the `ada_pkg` scope which are displayed in the Variables list in the lower pane.

   b. Select the `nested` package from the list of Scopes and press the Down button.

      You see that the Current Scope is now `ada_pkg.nested` and that the aggregate `data` is now in the list of scopes nested within `ada_pkg.nested`.

   c. Press the Up button.

      You see that the Current Scope is once again `ada_pkg`.

d. Select the aggregate `ada_pkg_record` and press the Down button.

You see that records (as well as C structures and unions) are treated as scopes since the Current Scope is now `ada_pkg.ada_pkg_record`.

Note that no further nested scopes exist inside `ada_pkg.ada_pkg_record`. Also note that the list of the components of `ada_pkg.ada_pkg_record` are displayed in the Variables pane below.

e. Press the Up button twice so that the Current Scope is once again Global.

4. Browsing C routines

C routines can declare static data items whose addresses are statically determined and are therefore eligible for monitoring. When browsing such routines remember that only static items nested within these routines will be displayed. Other items either belong to the Global scope, another routine's scope, a file scope (e.g. C static variables declared outside of a function), or are stack-based entities which don't have a static address.

a. Select function `c_routine` and press the Down button.

See that the Current Scope is now `c_routine` and that the list of nested scopes within it include the aggregate `c_static_struct`. Note also the list of static variables described in the lower Variables pane associated with the scope `c_routine`.

b. Select the aggregate `c_static_struct` and press the Down button

See that the Current Scope is now `c_routine.c_static_struct` and that the components of that structure are displayed in the Variables pane below.

c. Press the Up button twice so that the Current Scope is once again Global.

Selecting Variables

Selection of variables can be done by typing in the fully expanded name of the variable or by browsing scopes and using the lists to select displayed items.

1. Selection of various variables in the Ada package `ada_pkg`

a. Starting from the Global scope, locate the `ada_pkg` from the list of nested scopes and double-click the item

b. The Current Scope is now `ada_pkg` and the Variables list in the lower pane includes `ada_pkg_int` object. Select `ada_pkg_int` from that list.

See that the Selection text field now has `ada_pkg_int` displayed and the Info description provides its type, size, and address.

c. Press the Add button

See that an entry for `ada_pkg.ada_pkg_int` has been added to the Variable List in the Data Recording window.

d. Add the variable `ada_pkg_enum` by typing its name into the Selection text field in the Variables pane of the Variable Browser and press the Add button.

The text entered in the Selection text field of the Variables pane is associated with the Current Scope. Therefore, since the Current Scope was `ada_pkg`, the variable `ada_pkg.ada_pkg_enum` was added to the Variable List of the Data Recording window.

e. Double-click on the variable `control` in the Variables list.

See that an entry for `ada_pkg.control` has been added to the Variable List in the Data Recording window.

2. Selection of C global variables

a. Press the Up button in the Scopes pane so that the Global scope is selected.

b. Add the C variable `c_global_int` variable to the Variable List in the Data Recording window by double-clicking it in the Variables list in the bottom pane.

c. Double-click on the scope `c_routine`.

d. The Current Scope is now `c_routine`. Select the array `c_static_array` from the Variables list in the lower pane.

See that the array contains 4 components as indicated in the Info description field.

If we were to press the Add button, the entire array would be added to the Variable List in the Data Recording window. Instead, we will select components 2 and 3 in the following step.

e. Click in the Selection text field and change the "[]" characters to "[2:3]" to specify a slice of the array starting at component 2 and ending at component 3. Click the Add button.

3. Press the Close button to close the Variable Browser window

Using the Spreadsheet

This section provides an example of the use of the Spreadsheet Viewer window. For more information about the Spreadsheet Viewer window, see “Using the Spreadsheet Viewer Window” on page 8-5.

From the Data Recording window’s Output menu, select To Spreadsheet to create a Spreadsheet Viewer window.

Quickly Adding Multiple Variables

1. In the Spreadsheet Viewer window, click on the uppermost left hand cell.

   The selected cell gets a black outline and an I-beam cursor.

2. From the Selected menu, select Place Variables.

   The Spreadsheet Variables window appears. The Variable List is identical to the one in the Data Recording window.

3. In the Spreadsheet Variables window, select all five entries by depressing and holding down the left mouse button on the top entry and dragging the cursor down through the last entry and then releasing the left mouse button. All five items should be highlighted.

4. Change the Cell Layout selection to Vertical.

5. Change the Label Position selection to Left.

6. Press the OK button

   The spreadsheet cells starting with the uppermost left cell now describe the five variables you selected. The left hand column is a label field which includes the simple name of the variable. This field can be edited. The right hand column initially contains the same text, but will be replaced by the value of the associated variable when actual data sampling occurs.

Handling Array Variables

In the section above, we added an entry for the array slice `c_static_array`. During that selection we did not select Array Expansion so all components associated with that entry will be displayed within a single cell (limited by the width of the cell). In this section we will add the same array slice but allow it to expand to utilize individual cells for individual components.

1. Click on an empty cell in the left-most column

2. From the Selected menu, select Place Variables.

3. As before, the Spreadsheet Variables window appears. Select the `c_routine.c_static_array` item from the list. Change the Array Expansion from None to Horizontal. Press the OK button.

   See that the left-most selected cell contains the name of the selection and the two columns to the right will contain the individual components associated with that selection, once data sampling begins.

4. Resize the columns so that the information displayed can be fully seen. Select the first three columns by using the left mouse button (selecting the cells in the uppermost row is sufficient). From the Layout menu use Column Width and change it to 25 and press OK. (You may need to resize the Spreadsheet Viewer window so that you can see all three columns.)

Start Data Sampling

1. In the Data Recording window, choose System Clock... from the Timer menu. Ensure that the Sampling Rate is set to 1 second and press OK.
2. Connect to the target process by pressing the Connect button in the Probe pane of the Data Recording window.
3. Begin sampling data by pressing the Start button in the Sampling pane of the Data Recording window.

   See the values of the variables displayed in the non-label cells of the Spreadsheet Viewer window. The `ada_sample` program changes the values of its variables once per second. Note that the frequency in which values are updated in the spreadsheet is unrelated to the frequency at which the sampling occurs. The bottom pane of the Spreadsheet Viewer window controls the frequency of spreadsheet refreshes. This is especially important in situations in which NightProbe is being used to record and log data to a file at high-frequency rates but at the same time is used interactively to peek and poke at variables.

Changing Display Cell Attributes

1. Changing the default format base for an integer object
   a. Select the cell which is displaying the value of the variable `ada_pkg_int` by clicking once inside the cell
   b. From the Selected menu choose Cell Attributes
   c. Change the Format from Decimal to Hexadecimal
   d. Press the OK button
2. Setting low and high threshold limits
   a. Select the cell which is displaying the value of the variable `ada_pkg_enum` by clicking once inside the cell
   b. From the Selected menu choose Cell Attributes
   c. Set the Low Caution value to 2, the Low Danger value to 1, the High Caution value to 5, and the High Danger value to 6.
   d. Press the OK buttonw
   e. Click on an unrelated, empty cell in the spreadsheet.

   The background color of the cell associated with the value of `ada_pkg_enum` will remain white until the value drops below or above the threshold settings just applied. The pink and orange colors correspond to Low Danger and Low Caution, respectively, while the yellow and red colors correspond to High Caution and High Danger (see “Cell Color Legend” on page 8-14).

Modifying the Value of Variables

Variables can be modified directly through the spreadsheet by typing in new values into their cells.

1. Modifying a component of an array

   a. Click the first data value cell associated with `c_static_array` in the 2nd column of the last row of the spreadsheet.

      The value moves to the left-hand side of the cell and stops updating but the remainder of the spreadsheet continues to be refreshed with data samples.

   b. Type in a new value for the cell by backspacing over the existing text and specifying 200.0. Press the Enter key.

      The value of `c_static_array(2)` has been modified and is displayed. The program multiplies the value of `c_static_array(2)` by 0.995 each second, so the value will slowly begin to reduce in magnitude.

2. Controlling execution of the program

   The main program uses the variable `ada_pkg.control` to control execution of the program in the following manner:

```
loop
   case ada_pkg.control is
   when 0 => exit program
   when 1 => update variables
   when 2 => do nothing
   end case ;
   sleep 1 second
end loop ;
```

   a. Click on the cell showing the value of the control variable whose address is displayed in the cell to its left.

   b. Backspace over the existing value and type in 2 and press the Enter key

      This causes the program to skip updating variables -- notice how the values in the spreadsheet no longer change even though sampling is still active.

   c. Click on the same cell and change the value to 1 which causes the program to begin continuously updating the spreadsheet once again.

   d. Click on the same cell and change the value to 0 which causes the program to exit.

NOTE

When the program exits, NightProbe presents an error dialog to the user stating that it is no longer able to read the items that it was monitoring.

Conclusion

This concludes the tutorial. We hope that we have given you a sufficient overview so that you can get started using NightProbe. Reference the *NightProbe User's Guide* or use the context-sensitive help for the product if you have any questions.

Ada Sample - ada_sample.a

```
package ada_pkg is
--
   type states is (none, init, freeze, start, stop, in_flight, crash, flames);

   type record1_type is
      record
         int   : integer := 0;
         flt   : float := 0.0;
         enum  : states := none;
      end record;

   ada_pkg_int    : integer := 0;
   ada_pkg_float  : float := 0.0;
   ada_pkg_enum   : states := none;
   ada_pkg_record : record1_type;

   control        : integer := 0;

   package nested is
      type array_type is array (1..4) of integer;
      type record2_type is
         record
            x : record1_type;
            y : array_type;
         end record;
      data : record2_type;
   end nested;
--
end ada_pkg;

with ada_pkg;

procedure ada_routine is
begin
--
   -- Increment variables in ada_pkg
   ada_pkg.ada_pkg_int := ada_pkg.ada_pkg_int + 1;
   ada_pkg.ada_pkg_float := ada_pkg.ada_pkg_float + 2.0;
   case ada_pkg.ada_pkg_enum is
   when ada_pkg.states'last =>
      ada_pkg.ada_pkg_enum := ada_pkg.states'first;
   when others =>
      ada_pkg.ada_pkg_enum := ada_pkg.states'succ(ada_pkg.ada_pkg_enum);
   end case;
   ada_pkg.ada_pkg_record.int := ada_pkg.ada_pkg_record.int + 1;
   ada_pkg.ada_pkg_record.flt := ada_pkg.ada_pkg_record.flt + 2.0;
   case ada_pkg.ada_pkg_enum is
   when ada_pkg.states'last =>
      ada_pkg.ada_pkg_record.enum := ada_pkg.states'first;
   when others =>
      ada_pkg.ada_pkg_record.enum :=
         ada_pkg.states'succ(ada_pkg.ada_pkg_record.enum);
   end case;

   -- Increment variables in nested_pkg in ada_pkg
   for i in ada_pkg.nested.data.y'range loop
      ada_pkg.nested.data.x.int := ada_pkg.nested.data.x.int + 1;
      ada_pkg.nested.data.y(i) :=
         ada_pkg.nested.data.y(i) + i;
   end loop;
--
```

```
end ada_routine;


with ada_routine;
with ada_pkg;

procedure ada_sample is
--
   procedure c_routine;
   pragma import (C, c_routine);
   pragma linker_options ("c_sample.o");
--
begin
--
   ada_pkg.control := 1;

   loop
      case ada_pkg.control is
      when 0 =>
         exit;
      when 1 =>
         ada_routine;
         c_routine;
      when others =>
         null;
      end case;
      delay 1.0;
   end loop;
--
end ada_sample;
```

C Sample - c_sample.c

```
long            c_global_int = 1;

static long     c_static_int = 10;

void
c_routine(void)
{
   struct struct_type {
      int   int_component;
      float float_component;
   };

   static struct struct_type c_static_struct = {1, 2.0};
   static float              c_static_array[4] = {0.0,1.0,2.0,3.0};
   static int                c_static_func_int = 50;
   auto   int                c_stack_int = 0;
   auto   int                i;

   c_global_int ++;
   c_static_int += 2;
   c_static_func_int += 3;
   c_stack_int += 4;
   c_static_struct.int_component ++;
   c_static_struct.float_component += 1.1;

   for (i=0; i<4; i++) {
   if (c_static_array[i] > 10000.0) {
      c_static_array[i] /= 1.754;
   } else {
      c_static_array[i] *= 1.754;
   }
   }
}
```

C++ Tutorial

This tutorial demonstrates some of the commonly used features of NightProbe including:

- creating and selecting a program
- variable browsing
- spreadsheet use

The supplied tutorial program defines and implements a C++ class consisting of public and private variables and functions. In addition, functions within the program use static and dynamic variables. Some of the variables are scalars, some are arrays, and some are aggregates.

The tutorial files are in the **/usr/lib/NightProbe/Tutorial** directory. Source listings of these files are in:

- "C++ Sample - cpp_sample.cpp" on page A-22

Creating and Selecting a Program

1. The source code for the sample program used in this tutorialcan be found in the **/usr/lib/NightProbe/Tutorial** directory and is included at the end of this chapter for reference. The sample program contains C++ code.
2. Copy the source file, **cpp_sample.cpp**, found in that same directory and build the program using the following command(s):

   PowerMAX OS and PLDE:

   ```
   ec++ -g -o cpp_sample cpp_sample.cpp
   ```

   RedHawk Linux:

   ```
   g++ -g -o cpp_sample cpp_sample.cpp
   ```

3. Invoke NightProbe with the following command:

   ```
   nprobe &
   ```

   NightProbe displays the Data Recording window.

4. Invoke the sample program with the following command:

   ```
   ./cpp_sample
   ```

5. In the Data Recording window, press the Programs button.

   NightProbe displays the Program Selection window.

6. In the Program Selection window, press the Processes button and select the executing process `cpp_sample` as shown in the list of processes associated with your user. Press OK.
7. In the Program Selection window, press the Add button.

   You will see your program in the Program List display area of the Program Selection window. Press Close to close the Program Selection window.

Variable Browsing

The following sections provide an example of the use of the Variable Browser window. For more information about the Variable Browser window, see Chapter 6, "Using the Variable Browser Window".

Scopes and Variables

1. In the Data Recording window, press the Variables button.

   NightProbe displays the Variable Browser window. When the Variable Browser window first appears, the Current Scope is set to Global.

   The Variable Browser window is separated into two major panes which describe Scopes (the upper pane) and the Variables (the lower pane) within the selected scope. Scopes include source files, functions, C++ classes and an anonymous scope denoted Global which contains all the other outermost scopes and all C++ global variables. Whenever a scope is selected in the upper pane of the Variable Browser window, any scopes nested within it are displayed in the Scopes list within that pane. Additionally, any variables associated with the selected scope are displayed in the Variables list area of the lower pane below.

IMPORTANT

Browsing is most easily accomplished when the Automatic Update checkbox is checked. When this box is checked, selection of a scope causes automatic updating of the lists associated with the selected scope. When the Automatic Update checkbox is cleared, you must press the Update button associated with each list to refresh their contents even if you changes scopes.

Browsing Scopes

1. In the Variable Browser window, enable automatic updates by ensuring the Automatic Update checkbox is checked.
2. The list of scopes nested inside the Global scope include the following:
   a. `increment_counter` which is a function called by `main`

b. `main` which is the first function that executes when the program is run

c. `sample_class` which is an instance of the C++ class `cpp_class` declared in the Global scope

3. Browsing C++ classes

a. Select `sample_class` from the list of Scopes and press the Down button. Alternatively, double-clicking an item in the Scopes list is equivalent to selecting it and pressing Down.

You see that the Current Scope is `sample_class` and that there are two additional scopes nested inside `sample_class`:

1. the aggregate `cpp_class_struct`
2. the aggregate `cpp_class_union`

Structures in C++, as well as unions, are treated as scopes.

Also note the list of variables contained in the `sample_class` scope which are displayed in the Variables list in the lower pane.

b. Select the `cpp_class_struct` scope from the list of Scopes and press the Down button.

You see that the Current Scope is now `sample_class.cpp_class_struct`.

Note the list of variables contained in the `cpp_class_struct` scope which are displayed in the Variables list in the lower pane.

c. Press the Up button.

You see that the Current Scope is once again `sample_class`.

d. Select the aggregate `cpp_class_union` and press the Down button.

You see that the Current Scope is now `sample_class.cpp_class_union`.

Note the two members of `cpp_class_union` which are displayed in the Variables list in the lower pane.

Note that no further nested scopes exist inside `sample_class.cpp_class_union`.

e. Press the Up button twice so that the Current Scope is once again Global.

4. Browsing C++ functions

C++ functions can declare static data items whose addresses are statically determined and are therefore eligible for monitoring. When browsing such functions, remember that only static items nested within these routines will be displayed. Other items either belong to the Global scope, another routine's scope, a file scope

(e.g. C++ static variables declared outside of a function), or are stack-based entities which don't have a static address.

a. Select function `increment_counter` and press the Down button.

   See that the Current Scope is now `increment_counter` and that the list of nested scopes within it include the aggregate `counter_static_struct` (which is a structure of type `counter_struct_type`).

b. Select the aggregate `counter_static_struct` and press the Down button

   See that the Current Scope is now `increment_counter.counter_static_struct` and that the components of that structure are displayed in the Variables pane below.

c. Press the Up button twice so that the Current Scope is once again Global.

d. Select function `main` and press the Down button.

   See that the Current Scope is now `main` and that no further nested scopes exist inside `main`.

   Also, note `main_static_int` and `control` listed in the Variables pane below.

e. Press the Up button so that the Current Scope is once again Global.

Selecting Variables

Selection of variables can be done by typing in the fully expanded name of the variable or by browsing scopes and using the lists to select displayed items.

1. Selection of various variables in the C++ class `sample_class`

   a. Starting from the Global scope, locate the `sample_class` from the list of nested scopes and double-click the item

   b. The Current Scope is now `sample_class` and the Variables list in the lower pane includes `cpp_class_int` object. Select `cpp_class_int` from that list.

      See that the Selection text field now has `cpp_class_int` displayed and the Info description provides its type, size, and address.

   c. Press the Add button

      See that an entry for `sample_class.cpp_class_int` has been added to the Variable List in the Data Recording window.

   d. Add the variable `cpp_class_enum` by typing its name into the Selection text field in the Variables pane of the Variable Browser and press the Add button.

The text entered in the Selection text field of the Variables pane is associated with the Current Scope. Therefore, since the Current Scope was `sample_class`, the variable `sample_class.cpp_class_enum` was added to the Variable List of the Data Recording window.

e. Add the variable `cpp_class_private_float` by double_clicking its name in the Variables pane of the Variable Browser.

See that an entry for `sample_class.cpp_class_private_float` has been added to the Variable List in the Data Recording window.

2. Selection of C++ arrays

a. Select the `cpp_class_struct` scope from the list of Scopes and press the Down button.

b. The Current Scope is now `sample_class.cpp_class_struct`. Select the array `struct_float_array` from the Variables list in the lower pane.

See that the array contains 4 components as indicated by `[4]` notation in the Info description field.

If we were to press the Add button, the entire array would be added to the Variable List in the Data Recording window. Instead, we will select components 2 and 3 in the following step.

c. Click in the Selection text field and add `[1:2]` to the end of the array name so that it reads:

```
struct_float_array[1:2]
```

Note that in C++ arrays, the first array element is at location 0 so we specified `[1:2]` for the 2nd and 3rd entries.

d. Click the Add button.

See that an entry for `sample_class.cpp_class_struct.struct_float_array` with a Slice of `1..2` has been added to the Variable List in the Data Recording window.

3. Selection of C++ static variables

a. Press the Up button in the Scopes pane twice so that the Global scope is selected.

b. Select function `increment_counter` and press the Down button.

c. Select the aggregate `counter_static_struct` and press the Down button

d. Add the C++ static variable `counter_struct_state` to the Variable List in the Data Recording window by double-clicking it in the Variables list in the bottom pane.

e. Press the Up button in the Scopes pane twice so that the Global scope is selected.

f. Double-click on the function `main` in the Scopes pane.

   The Current Scope is now `main`.

g. Double-click on the variable `control` in the Variables list.

   See that an entry for `main.control` has been added to the Variable List in the Data Recording window.

4. Press the Close button to close the Variable Browser window

Using the Spreadsheet

This section provides an example of the use of the Spreadsheet Viewer window. For more information about the Spreadsheet Viewer window, see "Using the Spreadsheet Viewer Window" on page 8-5.

From the Data Recording window's Output menu, select To Spreadsheet to create a Spreadsheet Viewer window.

Quickly Adding Multiple Variables

1. In the Spreadsheet Viewer window, click on the uppermost left hand cell.

   The selected cell gets a black outline and an I-beam cursor.

2. From the Selected menu, select Place Variables.

   The Spreadsheet Variables window appears. The Variable List is identical to the one in the Data Recording window.

3. In the Spreadsheet Variables window, select all six entries by depressing and holding down the left mouse button on the top entry and dragging the cursor down through the last entry and then releasing the left mouse button. All six items should be highlighted.

4. Change the Cell Layout selection to Vertical.

5. Change the Label Position selection to Left.

6. Press the OK button

   The spreadsheet cells starting with the uppermost left cell now describe the six variables you selected. The left hand column is a label field which includes the simple name of the variable. This field can be edited. The right hand column initially contains the same text, but will be replaced by the value of the associated variable when actual data sampling occurs.

Handling Array Variables

In the section above, we added an entry for the array slice `sample_class.cpp_class_struct.struct_float_array`. During that selection we did not select Array Expansion so all components associated with that entry will be displayed within a single cell (limited by the width of the cell). In this section we will add the same array slice but allow it to expand to utilize individual cells for individual components.

1. Click on an empty cell in the left-most column
2. From the Selected menu, select Place Variables.
3. As before, the Spreadsheet Variables window appears. Select the `sample_class.cpp_class_struct.struct_float_array` item from the list. Change the Array Expansion from None to Horizontal. Press the OK button.

   See that the left-most selected cell contains the name of the selection and the two columns to the right will contain the individual components associated with that selection, once data sampling begins.

4. Resize the columns so that the information displayed can be fully seen. Select the first three columns by using the left mouse button (selecting the cells in the uppermost row is sufficient). From the Layout menu use Column Width and change it to 25 and press OK. (You may need to resize the Spreadsheet Viewer window so that you can see all three columns.)

Start Data Sampling

1. In the Data Recording window, choose System Clock... from the Timer menu. Ensure that the Sampling Rate is set to 1 second and press OK.
2. Connect to the target process by pressing the Connect button in the Probe pane of the Data Recording window.
3. Begin sampling data by pressing the Start button in the Sampling pane of the Data Recording window.

   See the values of the variables displayed in the non-label cells of the Spreadsheet Viewer window. The `cpp_sample` program changes the values of its variables once per second. Note that the frequency in which values are updated in the spreadsheet is unrelated to the frequency at which the sampling occurs. The bottom pane of the Spreadsheet Variables window controls the frequency of spreadsheet refreshes. This is especially important in situations in which NightProbe is being used to record and log data to a file at high-frequency rates but at the same time is used interactively to peek and poke at variables.

Changing Display Cell Attributes

1. Changing the default format base for an integer object

   a. Select the cell which is displaying the value of the variable `cpp_class_int` by clicking once inside the cell

   b. From the Selected menu choose Cell Attributes

   c. Change the Format from Decimal to Hexadecimal

   d. Press the OK button

2. Setting low and high threshold limits

   a. Select the cell which is displaying the value of the variable `cpp_class_enum` by clicking once inside the cell

   b. From the Selected menu choose Cell Attributes

   c. Set the Low Caution value to 2, the Low Danger value to 1, the High Caution value to 5, and the High Danger value to 6.

   d. Press the OK button

   e. Click on an unrelated, empty cell in the spreadsheet.

   The background color of the cell associated with the value of `cpp_class_enum` will remain white until the value drops below or above the threshold settings just applied. The pink and orange colors correspond to Low Danger and Low Caution, respectively, while the yellow and red colors correspond to High Caution and High Danger (see "Cell Color Legend" on page 8-14).

Modifying the Value of Variables

Variables can be modified directly through the spreadsheet by typing in new values into their cells.

1. Modifying a component of an array

   a. Click the first data value cell associated with `struct_float_array` in the 2nd column of the last row of the spreadsheet.

   The value moves to the left-hand side of the cell and stops updating but the remainder of the spreadsheet continues to be refreshed with data samples.

   b. Type in a new value for the cell by backspacing over the existing text and specifying 200.0. Press the Enter key.

   The value of `struct_float_array[1]` has been modified and is displayed. The program multiplies the value of `struct_float_array[1]` by 0.995 each second, so the value will slowly begin to reduce in magnitude.

2. Controlling execution of the program

   The `main` program uses the variable `control` to control execution of the program in the following manner:

```
while control > 0 loop
   case control is
     when 1 => update public class variables
               and private class variables
     else   => do nothing
   end case ;
   increment counter;
   sleep 1 second
end loop ;
```

   a. Click on the cell showing the value of the variable `control`.

   b. Backspace over the existing value and type in 2 and press the Enter key

      This causes the program to skip updating variables -- notice how the values in the spreadsheet no longer change even though sampling is still active.

      Also notice how the `counter_struct_state` continues to change as the `increment_counter` function is called each time through the loop regardless of the value of `control`.

   c. Click on the same cell and change the value to 1 which causes the program to begin continuously updating the spreadsheet once again.

   d. Click on the same cell and change the value to 0 which causes the program to exit.

NOTE

When the program exits, NightProbe presents an error dialog to the user stating that it is no longer able to read the items that it was monitoring.

Conclusion

This concludes the tutorial. We hope that we have given you a sufficient overview so that you can get started using NightProbe. Reference the *NightProbe User's Guide* or use the context-sensitive help for the product if you have any questions.

C++ Sample - cpp_sample.cpp

```
#include <stdio.h>
#include <unistd.h>

#define ARRAY_SIZE 4

class cpp_class {

   public:

      cpp_class();

      void cpp_procedure(void);

      enum states {none, init, freeze, start, stop, in_flight, crash, flames};

      struct struct_type {
         int    struct_int;
         float  struct_float;
         float  struct_float_array[ARRAY_SIZE];
         states struct_enum;
      } cpp_class_struct;

      union union_type {
         int   union_int;
         float union_float;
      } cpp_class_union;

      int    cpp_class_int;
      float  cpp_class_float;
      states cpp_class_enum;

      float get_private_float(void);
      void  set_private_float(float new_value);


   private:

      int   cpp_class_private_int;
      float cpp_class_private_float;


};

cpp_class::cpp_class()
{

  // Initialize variables in cpp_class

   cpp_class_int    = 0;
   cpp_class_float  = 0.0;
   cpp_class_enum   = none;

   cpp_class_struct.struct_int   = 0;
   cpp_class_struct.struct_float = 0.0;
   cpp_class_struct.struct_enum  = none;

   for (int i = 0; i < ARRAY_SIZE; i++) {
      cpp_class_struct.struct_float_array[i] = i * 100.0;
   }

   cpp_class_private_int    = 100;
```

```
   cpp_class_private_float  = 100.0;

}


void
cpp_class::cpp_procedure()
{

   // Increment variables in cpp_class

   cpp_class_int   += 1;
   cpp_class_float += 2.0;

   switch (cpp_class_enum) {
   case flames:
      cpp_class_enum = none;
      break;
   default:
      cpp_class_enum = (states)(cpp_class_enum + 1);
      break;
   }

   cpp_class_struct.struct_int   += 1;
   cpp_class_struct.struct_float += 2.0;

   switch (cpp_class_enum) {
   case flames:
      cpp_class_struct.struct_enum = none;
      break;
   default:
      cpp_class_struct.struct_enum =
         (states)(cpp_class_struct.struct_enum + 1);
      break;
   }

   for (int i = 0; i < ARRAY_SIZE; i++) {
      cpp_class_struct.struct_float_array[i] *= .995;
   }

}

float
cpp_class::get_private_float(void)
{
   return cpp_class_private_float;
}

void
cpp_class::set_private_float(float new_value)
{
   cpp_class_private_float = new_value;
}

void
increment_counter(void)
{

   enum state {even, odd};

   struct counter_struct_type {
      int   counter_struct_int;
      state counter_struct_state;
   };
```

```
   static struct counter_struct_type counter_static_struct = {0, even};

   if (++counter_static_struct.counter_struct_int % 2) {
      counter_static_struct.counter_struct_state = odd;
   }
   else {
      counter_static_struct.counter_struct_state = even;
   }

}


cpp_class sample_class;


int
main (int argc, void **argv)
{
   static int main_static_int = 0;
   float      local_float     = 0.0;
   static int control = 0;

   control = 1;

   while (control) {
      switch (control) {
      case 1:
         sample_class.cpp_procedure();
         local_float = sample_class.get_private_float() + 0.25;
         sample_class.set_private_float(local_float);
         break;
      default:
         break;
      }
      increment_counter();
      sleep(1);
   }

   return main_static_int;
}
```

B
GUI Customization

Before you can use NightProbe, you must set your DISPLAY environment variable. (See “Environment Variables” on page B-1.) If you want to run NightProbe “as is”, then simply invoke it. However, if you want to customize your NightProbe session, read on.

Environment Variables

You must ensure that the value of your DISPLAY environment variable is set to the name of your X server. You may determine whether the value of this variable is already set by typing the following at the system command prompt.

```
echo $DISPLAY
```

If the value of the variable has been set, output similar to the following is displayed:

```
eagle:0.0
```

If the value has not been set, you may set it from the command line. The format for specifying the name of your X server is as follows:

[*host*]:*server*[.*screen*]

Host specifies the name of your X server; *server* specifies the server number; and *screen* specifies the screen number. In most cases, the server number and screen number are zero.

Table B-1 shows how to set the DISPLAY environment variable from the command line by using different versions of the shell.

Table B-1. Setting the Display Variables

Type of Shell	Command Line Entry
Bourne Shell	**DISPLAY=eagle:0.0** **export DISPLAY**
C Shell	**setenv DISPLAY eagle:0.0**
Korn Shell	**export DISPLAY=eagle:0.0**

X Window System Resources

The graphical user interface (GUI) for NightProbe is based on OSF/Motif, and it runs in the environment of the X Window System. All X applications may be customized using X resources. Resources specify application attributes such as fonts, colors, screen layout, button and label names, mnemonics, and accelerators.

NightProbe provides default values for its X resources. Each user may override any X resources with personal preferences, or a site may provide for different defaults. These new settings can appear in the following places:

- In your **.Xdefaults** file
- On the **nprobe** invocation line (See "Command-Line Options" on page B-2.)
- In a resource file that the **xrdb(1)** X resource database manager reads

If you specify the same X resource on the **nprobe** invocation line and in your **.Xdefaults** file, the setting on the invocation line overrides the one in the file.

This appendix contains information that you need if you want to customize the graphical user interface.

NightProbe's behavior may be modified by specifying resources. Resources can be specified in many ways. A complete discussion of this topic is beyond the scope of this text. For more information on setting X11 client resources, refer to the *X Window System User's Guide*, to the *OSF/Motif Style Guide*, or to the **X(1)** and **xrdb(1)** man pages.

The following files in the **/usr/lib/X11/app-defaults** directory contain NightProbe's default resources:

Nprobe	Basic application default resources file
Nprobe-color	Application default resources file for color displays
Nprobe-mono	Application default resources file for monochrome displays

You can look in these files for examples of ways to customize NightProbe's appearance and behavior. To see all the NightProbe resources, use the **editres(1)** tool.

One way to specify resources is to copy the default resource files to your home directory and change your versions of NightProbe's resource files. That is the method used in this appendix.

Command-Line Options

NightProbe has its own set of command-line options. (See "Invoking NightProbe" on page 2-6.) When you invoke NightProbe, you may also specify any standard X tool kit command-line option. Such options include **-bg** *color* to set the color for the window

background; **-fg** *color* to set the color to use for text or graphics; and **-xrm** *resources-tring* to set selected resources. For example:

```
nprobe -xrm "*drecWindow*geometry:-0+0" &
```

would put the Data Recording window in the upper right corner. For a complete list of these options, refer to the **X(1)** man page.

Application Resources

In addition to the standard resources associated with an X11 or Motif program, NightProbe defines special *application resources* you can use to customize NightProbe's appearance and behavior. These resources affect the entire NightProbe graphical user interface; they are "global" to the application.

There are two categories of application resources used by NightProbe. One set of application resources applies to all products that are part of the NightStar tool set. In addition to these, NightProbe has its own application resources.

NightStar Resources

NightProbe is part of the NightStar tool set. To provide a consistent appearance among these tools and to provide an easy way for you to change the default appearance, special application resources exist that define fonts and colors. They allow you to change one resource (instead of many) to affect the font or color for a set of window components that have similar characteristics. These resources are applied only to certain window components; many of NightProbe's window components are unaffected by the NightStar resources.

For example, some textual display areas show only program output and some areas accept input only from you. Different colors are used for these areas to distinguish them. If you want to change the color for input fields, for example, you need to change only one resource in NightProbe's color resource file. See "Color Resources" on page B-5. The next time you run NightProbe, the color of all the input fields has the new setting.

Changing the `inputBackground` line in your **Nprobe-color** file to:

```
*inputBackground:          Yellow
```

causes the background color for all input areas to be yellow.

NightProbe Resources

NightProbe resources are not shared by other NightStar tools. A list of NightProbe resources follows.

`autoUpdate` — Controls the automatic updates of the Scope list and Variable list of the Variable Browser window and the display on the Spreadsheet Viewer window. See **`-autoupdate`** in "Invoking NightProbe" on page 2-6. See the Auto Update check box in "Components Affecting the Entire Window" on page 6-3, "Using the Control Area" on page 8-4, and "Using the Control Area" on page 8-14.

`lowCautionColor` — Control the color of the Low Caution field of the Spreadsheet Viewer window. The default value is orange. (See "Using the Control Area" on page 8-14.)

`lowDangerColor` — Control the color of the Low Danger field of the Spreadsheet Viewer window. The default value is magenta. (See "Using the Control Area" on page 8-14.)

`highCautionColor` — Control the color of the High Caution field of the Spreadsheet Viewer window. The default value is yellow. (See "Using the Control Area" on page 8-14.)

`highDangerColor` — Control the color of the High Danger field of the Spreadsheet Viewer window. The default value is red. (See "Using the Control Area" on page 8-14.)

Font Resources

Your X terminal vendor supplies you with vendor-specific directories and files that pertain to fonts. The programs **`xlsfonts(1)`** and **`xfontsel(1)`** can be used to help you find font names. NightProbe's font resources are in the file **`/usr/lib/X11/app-defaults/Nprobe`**. This section describes the special font resources available for NightStar tools in general and NightProbe in particular.

NightStar tools use proportional-width fonts except in areas that depend on text alignment; in these instances a fixed-width font is important for readability. If you decide to change fonts, make sure that you choose another fixed-width font for the font resources that have `fixed` in their names.

NightStar font resources include:

`smallFontList`	Used for areas that require a smaller font. NightProbe does not currently use this font.
`infoFontList`	Used for areas that display informational messages, warnings, errors. NightProbe uses this font for text fields.
`fixedFontList`	Used for areas that depend on text alignment. NightProbe uses this font for lists and the viewing area of the Spreadsheet Viewer window.
`smallFixedFontList`	Used for areas that depend on text alignment but require a smaller font. NightProbe does not currently use this font.

NightProbe uses a *default font* for most of the textual display in the windows. This proportional-width font is specified as the value of the standard Motif `fontList` resource. This font is used by window components that do not have a font specified for them. For example, changing the `fontList` line in your **`Nprobe`** file to:

```
*fontList:          9x15
```

would cause NightProbe to use the `9x15` font for the default font of most textual displays.

TIP:
Fonts can take up a lot of memory in your X server. If you are running low on server memory, you might want to set up your resources so that you use fewer fonts.

Color Resources

Your X terminal vendor supplies you with a vendor-specific file that pertains to colors. An example file of available colors is **`/usr/lib/X11/rgb.txt`**. NightProbe's color resources are in the **`Nprobe-color`** and **`Nprobe-mono`** files in the directory **`/usr/lib/X11/app-defaults`**. This section describes the special color resources available for NightStar tools in general and NightProbe in particular.

NightStar tools use the same color scheme to indicate that they are part of the same tool set and to provide cues about the usage of different areas in the windows. Each NightStar tool uses a unique color for its menu bars.

The following NightStar color application resources are defined:

`outputBackground` `outputForeground`	Used for the background and foreground colors in output-only areas.
`inputBackground` `inputForeground`	Used for the background and foreground colors in areas that accept user input.
`distinctBackground` `distinctForeground`	Used for the background and foreground colors in areas that require user input.

NightProbe uses a *default color* for most of the window areas. This color is specified as the value of the standard X11 `background` resource. This color is used by window components that do not have a color specified for them.

NightProbe determines whether you are using a monochrome or color display and automatically loads the appropriate NightProbe monochrome or color application defaults file. This means that you do not have to specify an X11 `customization` resource explicitly. If you do specify this resource (using either `-color` or `-mono` for the value), NightProbe still loads the appropriate application defaults file and uses its resource values.

Monochrome Display

The file **`/usr/lib/X11/app-defaults/Nprobe-mono`** has examples of monochrome resource specifications that were chosen to help distinguish certain fields using standard X Window System bitmaps.

If you want NightProbe to have white text on a black background, you can add these resources to your **`Nprobe-mono`** or **`Nprobe`** file.

```
*background: black
*foreground: white
```

Color Display

The file **`/usr/lib/X11/app-defaults/Nprobe-color`** has examples of color resource specifications. These resources include the default color and NightStar colors. See "Color Resources" on page B-5. The colors in this file were chosen to help distinguish certain fields and to emphasize areas that accept user input.

If you want to make changes to the colors, change your copy of the **`Nprobe-color`** file.

Internationalization

The following files in the **`/usr/lib/locale/C/LC_MESSAGES`** directory contain NightProbe's messages:

`NightProbe-help`	English language help message texts
`NightProbe-text`	Other English language message texts

To make these files available to NightProbe in another language:

1. Make the directory **`/usr/lib/locale/`***lang*`/LC_MESSAGES`** where *lang* is the name of your locale, if it does not already exist.
2. Copy the two NightProbe message files to this directory, and edit them so that they contain the translations for your locale. Be sure to leave unchanged all lines that begin with "//".

3. Before running NightProbe, establish your locale according to **`environ(5)`** and **`setlocale(3C)`**. Normally, this requires you to set one or several of the environment variables `LC_MESSAGES`, `LC_ALL`, or `LANG` to the name of your locale.

Labels for window titles, buttons, fields, menus, etc. are in the **`/usr/lib/X11/app-defaults/Nprobe`**. file. You can translate these resource values too.

For more information about internationalization, see the *OSF/Motif Programmer's Guide* and **`X(1)`** man page, especially text pertaining to the `XFILESEARCHPATH` environment variable.

C
PowerMAX OS Requirements

This chapter provides an overview of the system configuration requirements that need to be taken into account prior to installing and running NightProbe on a PowerMAX OS system.

System Configuration Requirements

The default kernel configuration shipped with the system should be sufficient for NightProbe use. However, the system administrator should be aware of the following kernel options.

Table C-1 describes the kernel options that NightProbe requires.

Table C-1. Required Kernel Options

Kernel Option	Description
`fbs`	Frequency-based scheduler. Required only if you use the frequency-based scheduler in conjunction with NightProbe.
`ipc`	Inter-process communications, including shared memory. For more information about shared memory, see "Interprocess Communication" in the *PowerMAX OS Programming Guide*.
`procfs`	**`/proc`** file system.

See the *NightSim User's Guide* for additional information about establishing the environment for NightSim, an optional real-time tool that provides access to the frequency-based scheduler and performance monitor.

NightProbe can produce information that can be analyzed by the NightTrace performance analysis tool. See the *NightTrace Manual* for information about establishing the environment for NightTrace.

Index

D

E

F

G

H

I

K

N

O

P

R

S

T

U

V

W

X

Spine for 1/2” Binder

Product Name: 0.5" from top of spine, Helvetica, 36 pt, Bold

Volume Number (if any): Helvetica, 24 pt, Bold

Volume Name (if any): Helvetica, 18 pt, Bold

Manual Title(s): Helvetica, 10 pt, Bold, centered vertically within space above bar, double space between each title

Bar: 1" x 1/8" beginning 1/4" in from either side

Part Number: Helvetica, 6 pt, centered, 1/8" up