NIGHTSTAR™

NightProbe RT User’s Guide

Version 3.1

(RedHawk™ Linux®)

concurrent

0898465-090
July 2006

Preface

Scope of Manual

This guide is designed to assist you in using NightProbe™, a real-time NightStar™ RT tool that provides a graphical user interface to data recording services.

Structure of Manual

This manual consists of fifteen chapters and five appendices. A brief description of the chapters and appendices 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 how to invoke NightProbe.
- Chapter 3 introduces the components of the NightProbe main window, the main control window for NightProbe.
- Chapter 4 describes NightProbe's Target Server dialog.
- Chapter 5 describes NightProbe's timing source configuration options.
- Chapter 6 describes NightProbe's various output methods.
- Chapter 7 describes NightProbe's Program Window.
- Chapter 8 describes NightProbe's PCI Device Window.
- Chapter 9 describes NightProbe's Shared Memory Window.
- Chapter 10 describes NightProbe's Mapped Memory Window.
- Chapter 11 describes how to use NightProbe's Item Browser.
- Chapter 12 describes how to use NightProbe's Item Definition window.
- Chapter 13 describes NightProbe's Item Properties window.
- Chapter 14 describes NightProbe's Spreadsheet Viewer window in detail.
- Chapter 15 documents the NightProbe Application Programming Interface and provides sample programs to demonstrate usage of the data structures and functions in the API.
- Appendix A discusses the NightStar License Manager (NSLM) and how to obtain and install licenses. It also discusses approaches for dealing with a firewall either on the system acting as the license server or on a system hosting the NightStar RT tools.

- Appendix C describes the syntax for specifying full variable names and eligibility rules.
- Appendix D provides a reference table for keyboard shortcuts to activating menu items, controlling data sampling, and traversing dialogs.
- Appendix E consists of tutorials for probing a program and a PCI device.
- Appendix F contains information about customizing the NightProbe graphical user interface.
- Appendix G provides an overview of the user and system configuration requirements that need to be taken into account prior to running NightProbe on a target system.

Syntax Notation

The following notation is used throughout this manual:

italic	Titles of books, reference cards, and items that you must specify appear in *italic* type. Special terms may also appear in *itali*cs.
`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:

0890514	*NightBench™ User's Guide*
0898004	*RedHawk Linux User's Guide*
0898008	*NightStar RT Installation Guide*
0898008	*NightStar RT Tutorial*
0898395	*NightView™ RT User's Guide*
0898398	*NightTrace™ RT User's Guide*
0898458	*NightSim™ RT User's Guide*
0898515	*NightTune™ RT User's Guide*
0898537	*MAXAda for RedHawk Linux Reference Manual*

Contents

Tables

1 Overview

NightProbe is a graphical tool for recording, viewing, and modifying data within a variety of resources:

- executing programs
- shared memory segments
- memory-mapped files and devices
- PCI devices

Data is sampled using non-intrusive techniques to guarantee short response time and minimal impact on the target resources and the target system. The source code of target programs does not need to be modified or recompiled in order to be monitored. Executing programs can be monitored and recorded without being stopped and restarted.

NightProbe can be run on a different processor or system from the target resource, which minimizes NightProbe's impact upon the target system.

Furthermore, NightProbe can probe executable programs which have been stripped of debug and symbol information, such as those in deployed scenarios. In such cases, a copy of the program containing the necessary debug and symbol information must be available for reading on the host system.

Recording and Monitoring

Data recording refers to sampling memory locations in target resources and recording that data for subsequent analysis. Memory locations may be identified by logical address, offset, or by variable name. NightProbe allows you to record data to a file in NightProbe Datastream API format as well as NightTrace data format (see "NightProbe Datastream API" on page 15-1).

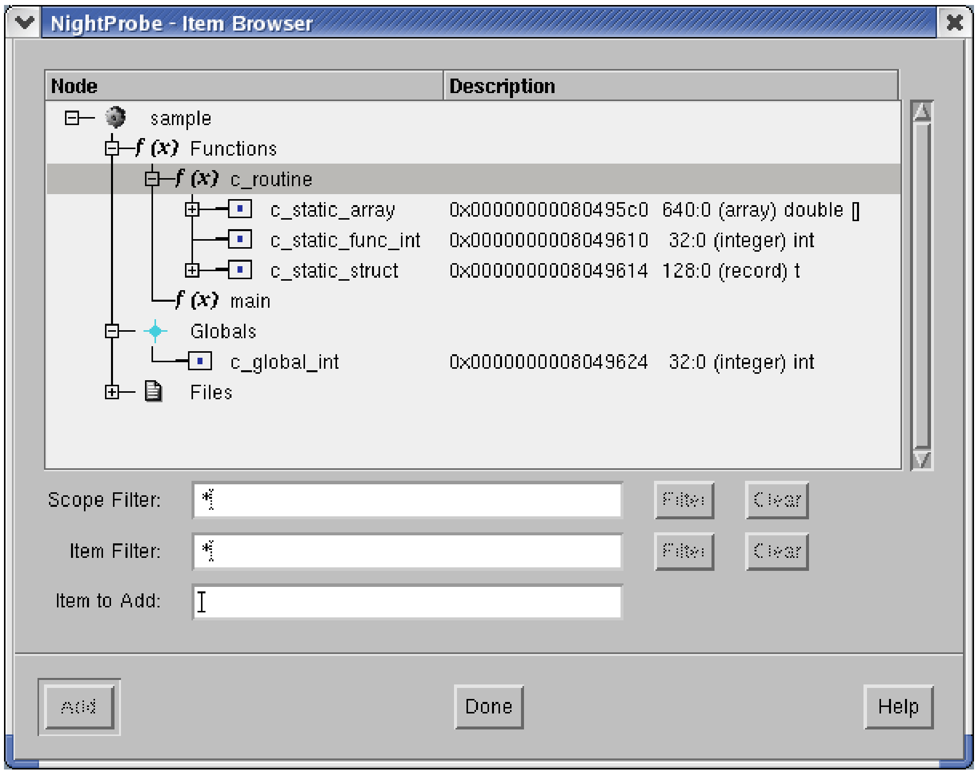
Data monitoring refers to displaying the sampled data for interactive visual inspection and, optionally, modification.

Eligible Variables

Variables with static addresses and layouts may be sampled. Supported languages include C, C++, and Fortran, as well as Ada programs built with the MAXAda compiler.

You can monitor and record any valid memory location in a program's address space, whether that location corresponds to a named variable or not.

Variables are selected from program symbol files using the `Item Browser` dialog as shown in Figure 1-1 (see Chapter 11, "Item Browser" for more information on this dialog):



Node	Description
sample	
f (x) Functions	
f (x) c_routine	
c_static_array	0x00000000080495c0 640:0 (array) double []
c_static_func_int	0x0000000008049610 32:0 (integer) int
c_static_struct	0x0000000008049614 128:0 (record) t
f (x) main	
Globals	
c_global_int	0x0000000008049624 32:0 (integer) int
Files	

Scope Filter: *
Item Filter: *

Figure 1-1. Selecting Variables with the Item Browser

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

Sampling

Data sampling is the process of collecting the values from target memory locations and making the sampled data available to output or viewer processes that can record the data or display the data for interactive monitoring.

Data sampling occurs when a sampling event is triggered by a NightProbe timing source. The rate at which sampling occurs can be regular or irregular, depending on the timing source selected.

NightProbe supports four mechanisms which define the sampling rate:

- on-demand sampling
- iterative sampling driven by the system clock
- iterative sampling controlled by a frequency-based scheduler
- programmed sampling driven by a NightProbe Trigger Client program

See “Timer Selection” on page 3-12 for a more complete discussion of these timing sources.

Using NightProbe

The primary steps in using NightProbe are:

1. Define the target system.
2. Select the target resources you wish to probe.
3. Select variables from target resource programs or create artificial variables to create views into a target resource.
4. Specify the desired sampling timer mechanism.
5. Select output methods.
6. Connect NightProbe to the target system and target resources.
7. Start sampling.

NOTE

On RedHawk® Linux® systems, you must have appropriate file access to the target resource or be granted the `CAP_SYS_RAWIO` capability in order to record, monitor, or modify the resource.

In order to set the CPU bias of the NightProbe server or of the program specified using the Program Output method (see “Program Output” on page 6-14), you must have the `CAP_SYS_NICE` capability.

See “Capabilities” on page G-1 for more information.

2
Invoking NightProbe

The NightProbe tool is available on your system as **`/usr/bin/nprobe`**. The format for executing **`nprobe`** is described below.

To get information about NightProbe:

`nprobe [--help] [--version]`

To use NightProbe to record or monitor variable locations:

```
nprobe [--record=output_file]
       [--trace=key_file]
       [--sheet=config_file]
       [--program=program_file]
       [--list]
       [-Xoption ...]
       [config_file | executable_file ]
```

To translate data files to text:

`nprobe --input=`*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.

`--program=`*progfile*

This option sends sampled data to the executable program *progfile* which is launched when sampling begins. The program should consume the data using the NightProbe Datastream API (see “NightProbe Datastream API” on page 15-1).

`--record=`*output_file*

Activate recording to file *output_file* when sampling begins.

--trace=*key_file*

Activate streaming of recorded data to a NightTrace user daemon using a shared memory buffer designated by *key_file*. The thread name **nprobe** will be used, and several NightTrace configuration files prefixed with the thread name will be generated when you connect, including **nprobe.session**.

--sheet=*layout_file*

Activate a spreadsheet viewer using the layout configuration in file *layout_file*.

--list

Activate a list viewer.

--input=*data_file*

Translate a previously recorded data file from its internal format to printable ASCII text. The **--input** option must be followed by '=' and the path to the data file. The text translation will be written to standard output. No other options should be used with **--input**. The X Window interface will not be started when using the **--input** 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(7x)** 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 NightProbe main window, configure a data recording session, and then select the Save Session As... menu item from the NightProbe menu. Procedures are fully explained in Chapter 3, "NightProbe Main Window".

executable_file

This argument allows you to specify the name of an executable file. It is automatically added to the list of target resources so that you may immediately begin browsing for items within that file.

You may invoke **nprobe** without specifying any options or arguments.

NOTE

`nprobe` requires that your `DISPLAY` environment variable is set appropriately.

Getting Help

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

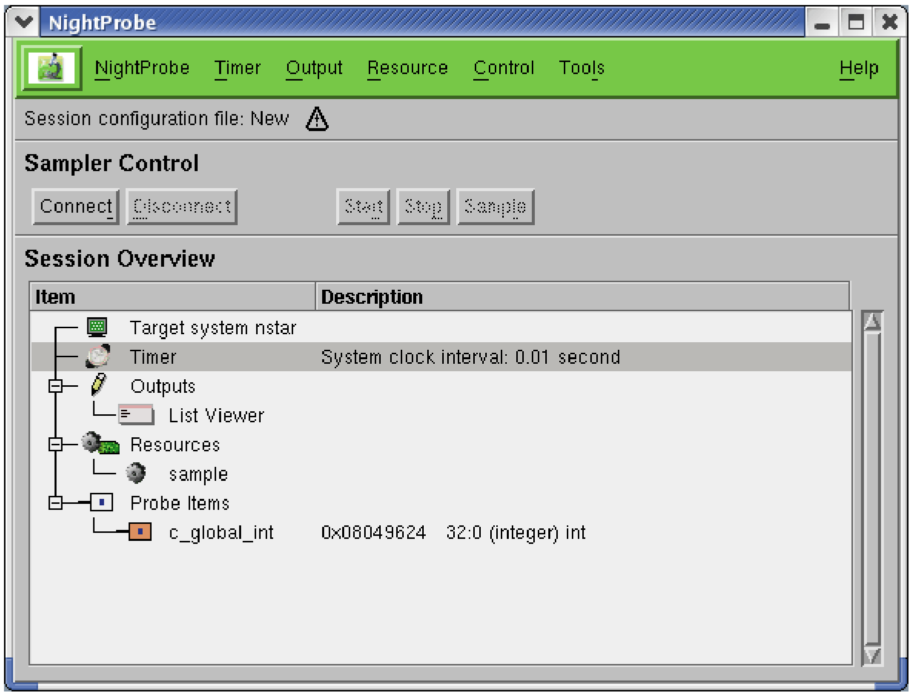
- the `Help` menu on the menu bars of the NightProbe windows (see "Help" on page 3-8)
- the `Help` button on the NightProbe dialogs
- the **`--help`** command line option
- the **`nprobe(1)`** system manual page

3
NightProbe Main Window

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

The NightProbe main window consists of the following components:

- Menu Bar (see “Menu Bar” on page 3-2)
- Session Configuration Status Area (see “Session Configuration Status Area” on page 3-10)
- Sampler Control Area (see “Sampler Control Area” on page 3-10)
- Session Overview Area (see “Session Overview Area” on page 3-12)



Item	Description
Target system nstar	
Timer	System clock interval: 0.01 second
Outputs	
List Viewer	
Resources	
sample	
Probe Items	
c_global_int	0x08049624 32:0 (integer) int

Figure 3-1. NightProbe main window

Menu Bar

The menu bar provides access to session configuration services, additional tools, and help. The activities provided in the pop-up menus in the Session Overview Area are available from the menu bar as well. The menu bar provides the following menus:

- NightProbe (see "NightProbe" on page 3-2)
- Timer (see "Timer" on page 3-4)
- Output (see "Output" on page 3-4)
- Resource (see "Resource" on page 3-5)
- Control (see "Control" on page 3-5)
- Tools (see "Tools" on page 3-6)
- Help (see "Help" on page 3-8)

Each menu is described in the sections that follow.

NightProbe

Mnemonic: Alt+N

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

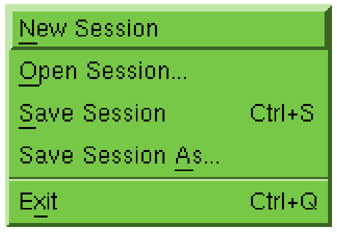


Figure 3-2. NightProbe menu

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

New Session

Mnemonic: N

This option allows you to clear all information from the current session 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 Session...

Mnemonic: O

This option allows you to open a session configuration file that you have previously saved and load all the configuration items into the current session.

If the window contains unsaved changes, NightProbe displays a warning dialog. You may proceed to open the new session, thereby discarding any unsaved changes, or cancel the operation.

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.

Save Session

Mnemonic: S
Accelerator: Ctrl+S

This option allows you to save the configuration data from the current session in the configuration 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 Session As...

Mnemonic: A

This option allows you to specify the name of the file in which you wish the configuration data from the current session to be saved.

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

Exit

Mnemonic: X
Accelerator: Ctrl+Q

This option exits NightProbe. If there are unsaved changes in the current session, a dialog will ask you if you wish to save the session before exiting.

Timer

Mnemonic: Alt+T

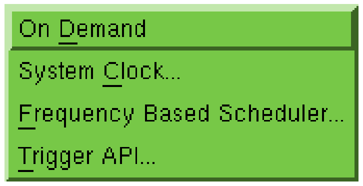


Figure 3-3. Timer menu

These menu items activate the timing selection as described in “Timer Selection” on page 3-12.

Output

Mnemonic: Alt+O

You must select at least one destination for output or NightProbe will not allow connection to the target resource (see “Sampler Control Area” on page 3-10).

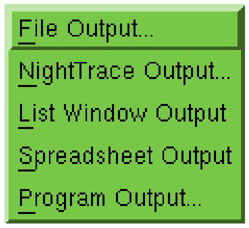


Figure 3-4. Output menu

These menu items activate output selection as described in “Outputs” on page 3-14.

Resource

Mnemonic: Alt+R

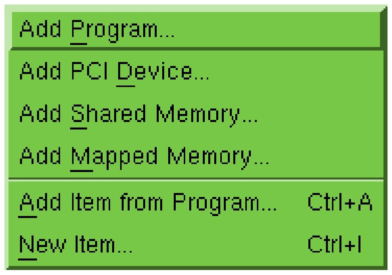


Figure 3-5. Resource menu

These menu items activate resource selection as described in “Resources” on page 3-16.

Control

Mnemonic: Alt+C

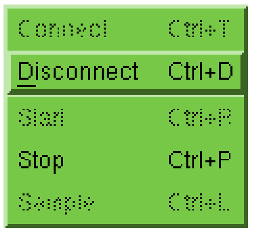


Figure 3-6. Control menu

See “Sampler Control Area” on page 3-10 for a description of these controls.

Tools

Mnemonic: Alt+L

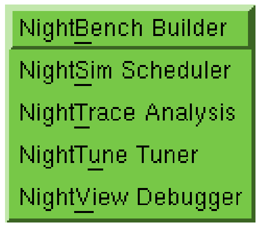


Figure 3-7. Tools menu

The following describe the options on the Tools menu:

NightBench Builder

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 only available on RedHawk systems that have MAXAda installed.

See also:

- *NightBench User's Guide*

NightSim Scheduler

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 RT User's Guide*

NightTrace 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 you 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 RT User's Guide*

NightTune Tuner

Mnemonic: U

Opens the NightTune Tuner. NightTune is a graphical tool for analyzing the status of the system in terms of processes, interrupts, context switches, interrupt CPU affinity, processor shielding and hyperthreading control as well as network and disk activity. NightTune can adjust the scheduling attributes of individual or groups of processes, including priority, policy, and CPU affinity.

See also:

- *NightTune RT User's Guide*

NightView 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 RT User's Guide*

Help

Mnemonic: Alt+H

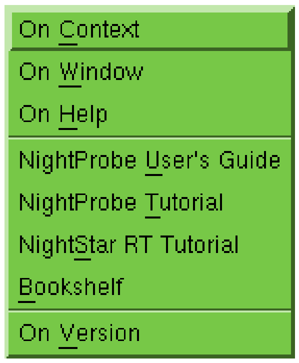


Figure 3-8. Help menu

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: U

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 tutorials which show some of the commonly-used features of NightProbe.

NightStar RT Tutorial

Mnemonic: S

Opens a HyperHelp window containing a tutorial which demonstrates the features of NightSim, NightProbe, NightView, NightTrace, and NightTune in one cohesive example.

Bookshelf

Mnemonic: B

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

On Version

Mnemonic: V

Displays a short description of the current version of NightProbe.

Session Configuration Status Area

The Session Configuration Status Area is located just below the menu bar and contains information on the name of the currently selected session 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 Session... menu option on the File menu or are provided on the command line.

Sampler Control Area

The Sampler Control Area provides buttons for controlling the state of data recording and monitoring as well text fields indicating its current status.

Connect

Accelerator: Ctrl+T

Pressing Connect initiates sampler activity. A sampler process is initiated on the target system. All memory pages associated with the selected variables for each resource are mapped into the sampler process’s address space. Output methods and timer functions are elaborated which may involve creating or opening files, initiating NightTrace API calls, joining a Frequency Based Scheduler, or connecting to a NightProbe Trigger API Client program. If any of these activities cannot be successfully completed, a diagnostic window is shown, all associations with target resources and the target system are released, and the connection process terminates.

This is desensitized unless a resource has been defined (see “Resources” on page 3-16), probe items have been selected or defined (see “Probe Items” on page 3-18), and an output method has been defined (see “Outputs” on page 3-14).

You must be in a connected state to begin data sampling. Note that sampling does not actually begin until Start or Sample is pressed.

Changes to the session configuration are not allowed while in the connected state.

Disconnect

Accelerator: Ctrl+D

Pressing Disconnect terminates sampler activity. If sampling is active, it is stopped. The connection to the target system is terminated after all memory mappings to all target resources are released.

Start

Accelerator: Ctrl+R

Pressing Start initiates sampling. The Start button will be desensitized unless Connect has already been pressed and the selected timing source is not On

Demand. If the timing source is System Clock, sampling will begin at the specified rate.

If the timing source is Frequency Based Scheduler, sampling will begin on the next cycle associated with the scheduler. If the scheduler is not running, sampling will not begin until the scheduler is started or resumed. The scheduler is controlled externally from NightProbe, by either NightSim, the **rtcp(1)** command, or a program using the Frequency Based Scheduler API (see **fbsconfigure(3)**). NightSim can be launched from the Tools menu

If the timing source is Trigger API, sampling will begin on the next receipt of a NightProbe Trigger Client's sample request on the NightProbe Trigger Queue. If the NightProbe Trigger Client is not yet running, sampling will not begin until the Trigger Client program is started.

If NightTrace output has been selected, samples will be discarded if an associated NightTrace daemon is not running. They will begin to be collected when the daemon initiates. A NightTrace daemon can be launched via the **ntraceud(1)** command or from the NightTrace graphical interface which can be launched from the Tools menu.

Stop

Accelerator: Ctrl+P

Pressing Stop halts sampling. The Stop button will be desensitized unless Start has already been pressed. The sampler process remains in a connected state. Sampling will resume when Start is pressed. Note that if the selected timing source is the Frequency Based Scheduler, pressing Stop does not stop the scheduler -- it merely halts sampling. This will not result in scheduler overruns, as the sampler process continues to cycle, but without sampling activity.

Sample

Accelerator: Ctrl+L

Pressing Sample causes a single sample to be obtained from the sampler process. The Sample button will be desensitized unless Connect has already been pressed (see “Connect” on page 3-10) and the selected timing source is On Demand (see “On Demand” on page 3-13).

Session Overview Area

The Session Overview Area provides primary access and control over the five main configurable areas of data recording:

- Target System Selection (see "Target System Selection" on page 3-12)
- Timer Selection (see "Timer Selection" on page 3-12)
- Output Method (see "Outputs" on page 3-14)
- Target Resources (see "Resources" on page 3-16)
- Probe Items (see "Probe Items" on page 3-18)

NOTE

All icons in the Session Overview Area support right-click pop-up menu actions which allow you to add, remove, view, and manipulate the items. These menus are not available when in the connected state; the session configuration may not be modified while connected. In addition, double-click may be used to access properties where applicable.

Target System Selection

To the right of the Target System icon, the name of the target system is displayed.

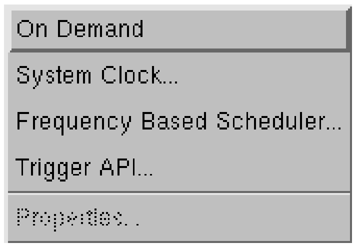
Right-clicking the icon will display a menu which contains Properties.... Selection of that menu option launches the Target Server dialog which allows you to change the target system, the user name, and the scheduling attributes of the sampler process. This dialog may also be launched by double-clicking the Target System icon.

See "Target Server Selection" on page 4-1 for more information.

Timer Selection

To the right of the Timer icon, the description of the selected timing source is displayed.

Right-clicking the Timer icon displays the following menu:



By default, On Demand sampling is selected.

On Demand

The sampler will sample the variables only when the Sample button in the Sampler Control Area is pressed.

System Clock...

Launches the Set System Timer dialog (see “Set System Timer” on page 5-1) which allows you to chose the rate at which sampling will occur.

Frequency Based Scheduler...

Launches the Set FBS Timer dialog (see “Set FBS Timer” on page 5-3) which allows you to chose the scheduler ID and period at which samples should be taken. Use of a frequency-based scheduler as the timing source allows for synchronization of data sampling with the target application, if that application is also scheduled on the same frequency-based scheduler.

Trigger API...

Launches the Set Trigger Timer dialog (see “Set Trigger Timer” on page 5-6) which allows the user to specify the name of the NightProbe Trigger Server Queue on the target system corresponding to that specified in the `np_trigger_open()` call made by the NightProbe Trigger Client. See “NightProbe Trigger API” on page 15-20 for more information.

Selecting the Properties... option from the Timer option menu will launch the appropriate timing source dialog so that you can make adjustments.

Selection of a timing source may also be initiated using the Timer menu (see “Timer” on page 3-4) on the menu bar of the NightProbe main window.

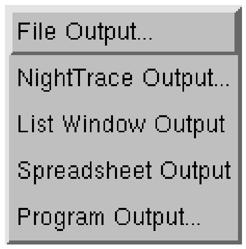
NOTE

The timing source may not be changed while the sampler is in the connected state. The Timer option menus will be disabled while connected.

Outputs

The Outputs icon lists all outputs you select. The list can be expanded or collapsed by clicking the control box to the left of the icon.

Right-clicking the icon will display the following menu:



File Output...

Launches the File Output dialog which allows you to specify a pathname for the file to which data samples will be written. See "File Output" on page 6-1 for more information. A file icon and the name of the selected file will be added to the list of Outputs.

NightTrace Output...

Launches the NightTrace Specification dialog which allows you to configure how NightTrace will capture and possibly display the data. See "NightTrace Output" on page 6-3 for more information. An icon representing NightTrace will be added to the list of Outputs.

List Window Output

Displays a List Viewer window and adds a List Window icon to the list of Outputs. The list viewer displays the names and values of all Probe Items in a scrollable text area. See "List Viewer" on page 6-10 for more information.

Spreadsheet Output

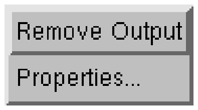
Displays the Spreadsheet Viewer window and adds a Spreadsheet Window icon to the list of outputs. The Spreadsheet Viewer displays variables in a configurable grid and allows for modification of variables as well. See "Spreadsheet Viewer" on page 6-13 for more information.

Program Output

Launches the Program Output dialog which allows you to specify a program that will be used to process the data recording output using the NightProbe Datastream API (see “NightProbe Datastream API” on page 15-1). See “Program Output” on page 6-14 for more information.

Multiple outputs can be added to the list, but only one instance of each output option can be present at any given time.

Once an item is added to the output list, right-clicking the icon representing it will display the following menu:



Remove Output

Removes the selected item from the list.

Properties...

Launches the appropriate output properties window which allows you to modify configurable settings.

Selection of an output may also be initiated using the Output menu (see “Output” on page 3-4) on the menu bar of the NightProbe main window.

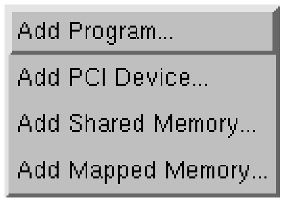
NOTE

Items in the Outputs list can not be removed nor their properties modified while in the connected state. While connected, the output menus are disabled.

Resources

The Resources icon lists all target resources you select. The list can be expanded or collapsed by clicking the control box to the left of the icon.

Right-clicking the icon will display the following menu:



Add Program...

Launches the Program Window which allows you to specify a program file, process name, and process ID. See "Program Window" on page 7-1 for more information.

Add PCI Device...

Launches the PCI Device window which allows you to specify a PCI device which you wish to probe. The device may be specified using the Vendor ID, Device ID, Region Number, and Slot Number or can be selected from a descriptive list. See "PCI Device Window" on page 8-1 for more information.

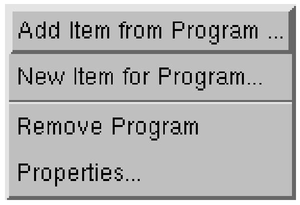
Add Shared Memory...

Launches the Shared Memory window which allows you to select a shared memory segment which you wish to probe. The segment may be selected by IPC Key, IPC Shared memory ID, IPC Key File pathname, or POSIX Shared Memory name. See "Shared Memory Window" on page 9-1 for more information.

Add Mapped Memory...

Launches the Mapped Memory window which allows you to specify a device pathname which will be mapped and probed. The device may be **`/dev/mem`**, another device, or any file that the **`mmap(2)`** system service supports mapping. See "Mapped Memory Window" on page 10-1 for more information.

Once a resource is added to the output list, right-clicking the icon representing it will display a menu similar to the following:



Add Item from *resource*...

Launches the Item Browser window which allows you to choose variables that you want to probe in that program. This menu option is not available for non-program resources. See "Item Browser" on page 11-1 for more information.

New Item for *resource*...

Launches the Item Definition window which allows you to create artificial variables by browsing program files for types, selecting a desired type, and specifying resource addresses or offsets to the new items. These new items are not allocated by NightProbe in the target resource. They merely represent a view of a location that already exists in the resource. See Chapter 12, "Item Definition Window" for more information.

Remove *resource*

Removes the selected resource from the list. Any variables associated with the resource will also be removed from the Probe Items list.

Properties...

Launches the appropriate resource properties window which allows you to modify configurable settings.

Selection of a resource may also be initiated using the Resource menu (see "Resource" on page 3-5) on the menu bar of the NightProbe main window.

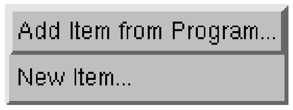
NOTE

Resources can not be removed nor their properties modified while in the connected state. While connected, the resource menus are disabled.

Probe Items

The Probe Items icon lists all target probe items you selected in the order in which they will be sampled. The list can be expanded or collapsed by clicking the control box to the left of the icon.

Right-clicking the icon will display the following menu:



Add Item from Program...

Launches the Item Browser window which allows you to select variables from program files and add them to the list of items to be probed. See "Item Browser" on page 11-1 for more information.

NOTE

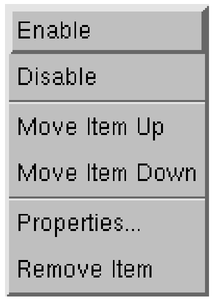
This option is only valid for program resources.

New Item...

Launches the Item Definition window which allows you to create artificial variables by browsing program files for types, selecting a desired type, and specifying resource addresses or offsets to the new items. These new items are not allocated by NightProbe in the target resource, they merely represent a view of a location that already exists in the resource. See Chapter 12, "Item Definition Window" for more information.

Once an item is added to the Probe Items list, its name and description will displayed to the right of the icon representing the item. The description includes the address, bit size, bit offset, the class of type, and the type name or type declaration.

Right-clicking on an item icon will display the following menu:



Enable
Disable

Selection of the Enable or Disable menu options toggles the enabled setting.

If an item in the list is disabled, it is ignored and not included in data samples. Items may become disabled automatically if they are no longer valid with respect to their program file (e.g. a variable entered from a previous configuration file was since removed from the program).

Enabled items are presented with an <u>orange</u> item icon while disabled icons are <u>white</u>.

Multiple items may be disabled or enabled together using multiple selection.

Move Item Up
Move Item Down

Moves the item in the selected direction. The ordering of items in the list controls the order in which they are sampled. In most cases, this ordering is unimportant, but for some hardware devices the order in which items are referenced is critical. Items may also be reordered by selecting an item and using Shift-UpArrow and Shift-DownArrow keys.

Properties...

Launches the Item Properties window which allows you to select the default display format and specify an array slice for array items.

Remove Item

Removes the item from the Probe Items list. Multiple items may be removed together using multiple selection.

NOTE

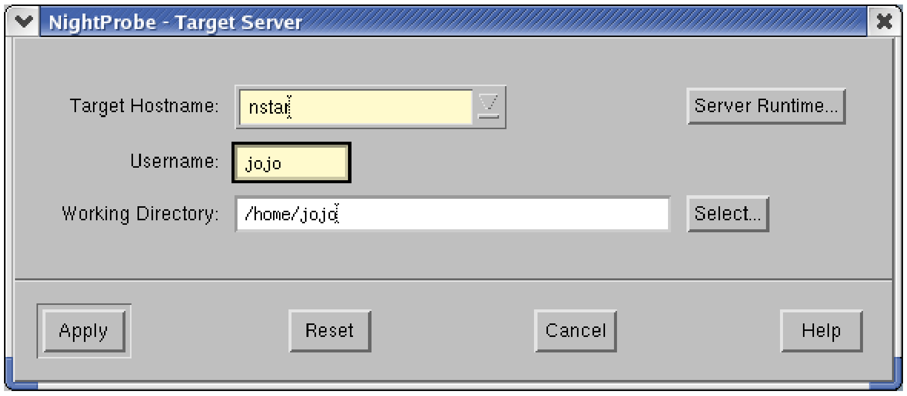
Item menus are disabled while in the connected state.

4
Target Server Selection

The Target Server dialog allows you to specify the system on which the target resources exist, 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. NightProbe uses a TCP/IP socket connection to the NightStar daemon (**`nstar.d`**) on the target system to communicate with the NightProbe server.

The Target Server dialog is launched from the Properties... menu option of the Target icon in the Session Overview Area (see "Session Overview Area" on page 3-12) of the NightProbe main window.

The Target Server dialog is shown in Figure 4-1.



Target Hostname: nstar
Username: jojo
Working Directory: /home/jojo

Figure 4-1. Target Server dialog

Target Hostname

The name of the system to be probed.

Server Runtime...

Presents the Run Time Settings dialog (see "Run Time Settings" on page 4-3) allowing the user to specify the scheduling policy, priority, and CPU bias for programs running on the target system.

Username

The login name of the user connecting to the target system.

Working Directory

The working directory for the NightProbe Server running on the target system.

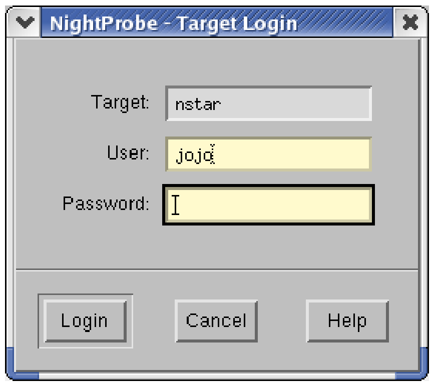
Select

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

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 NightProbe main window is pressed (see "Sampler Control Area" on page 3-10).



Target: nstar
User: jojo
Password:

Figure 4-2. User Authentication dialog

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.

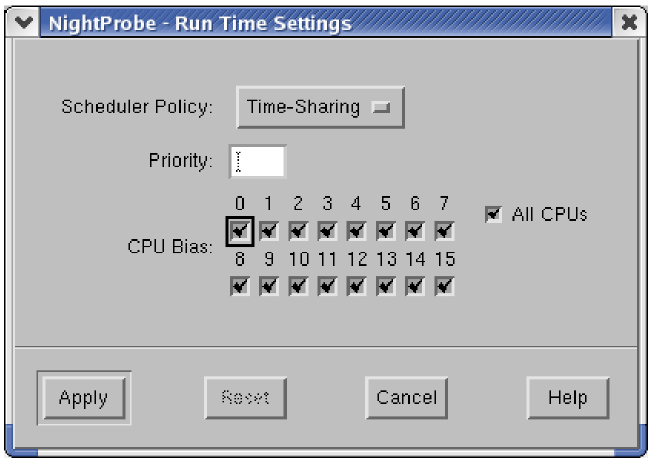
Once authentication has succeeded, NightProbe will no longer require you to re-authenticate even if you disconnect and reconnect, unless you change the target system or user name, or exit NightProbe. Note that NightProbe does not store any password information on disk.

Run Time Settings

The Run Time Settings dialog allows you to specify the scheduling policy, priority, and CPU bias for a particular program and is presented when the user presses:

- the Server Runtime... button on the Target Server dialog (see "Target Server Selection" on page 4-1)
- the Runtime... button on the Program Output dialog (see "Program Output" on page 6-14)

The Run Time Settings dialog is shown in Figure 4-3.


NightProbe - Run Time Settings

Scheduler Policy: Time-Sharing

Priority:

CPU Bias: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 (all checked)

All CPUs (checked)

Figure 4-3. Run Time Settings dialog

Scheduler Policy

This drop-down menu allows you to select the POSIX scheduling policy for the specified program. 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).

FIFO

The FIFO (first–in–first–out) policy (SCHED_FIFO) is associated with the fixed-priority class in which critical processes 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 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 good throughput to CPU-bound processes.

For a full description of the behavior of processes that are scheduled under the respective policies on RedHawk Linux systems, refer to the “Process Scheduling” chapter of the *RedHawk Linux User's Guide*.

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 “Scheduler Policy” above). Higher numerical values correspond to more favorable scheduling priorities.

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

For complete information on scheduling policies and priorities on RedHawk Linux systems, refer to the “Process Scheduling” chapter of the *RedHawk Linux User's Guide*.

CPU Bias

This panel of checkbuttons allows you to select the processor or processors on which a particular program can be scheduled.

You 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 can be scheduled on all available processors.

5
Timing Source Configuration

The selection of the timing source controls when, and at what rate, data sampling will occur.

On Demand sampling (see “On Demand” on page 3-13) means that samples will only be taken when you press the Sample button (see “Sample” on page 3-11) on the NightProbe main window.

This chapter discusses the dialogs relating to the selection of timing sources:

- Set System Timer (see “Set System Timer” on page 5-1)
- Set FBS Timer (see “Set FBS Timer” on page 5-3)
- Set Trigger Timer (see “Set Trigger Timer” on page 5-6)

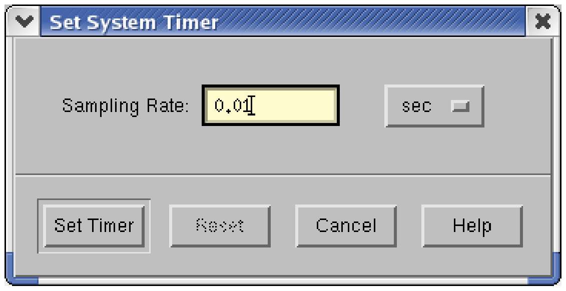
Selecting the Properties... menu option from the Timer icon menu in the Session Overview Area (see “Session Overview Area” on page 3-12) of the NightProbe main window will launch the appropriate timing source dialog so that you can make adjustments. Double-clicking the Timer icon has the same effect.

See “Timer Selection” on page 3-12 for more information.

Set System Timer

Selecting the System Clock... menu option from the Timer icon in the Session Overview Area (see “Session Overview Area” on page 3-12) or from the Timer menu of the NightProbe main window launches the Set System Timer dialog.

Selection of this timing source means the sampler will use the system clock as the timing source at the interval you select in the dialog.



Sampling Rate: 0.01

Figure 5-1. Set System Timer dialog

Sampling Rate

Choose a unit of time measurement using the drop-down menu to the right of the Sampling Rate text field and then enter the amount of time that should pass between samples. The interval you select is displayed to the right of the Timer icon in the Session Overview Area (see “Session Overview Area” on page 3-12).

You can change the interval subsequently by selecting the Properties... option from the option menu on the Timer icon in the Session Overview Area of the NightProbe main window. See “Timer Selection” on page 3-12 for more information.

Set FBS Timer

Selecting the Frequency Based Scheduler... menu option from the Timer icon menu in the Session Overview Area (see “Session Overview Area” on page 3-12) or from the Timer menu of the NightProbe main window launches the Set FBS Timer dialog.

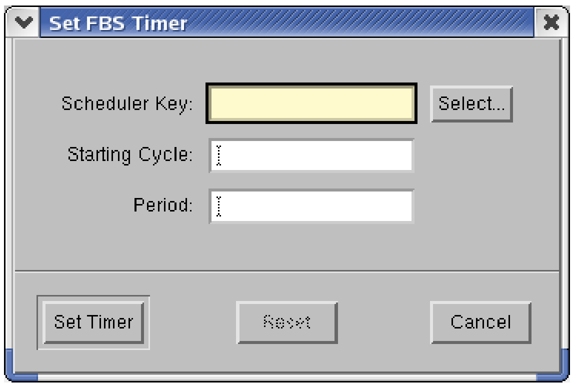


Figure 5-2. Set FBS Timer dialog

Selecting this option means that the NightProbe will take samples as directed by a frequency-based scheduler. This allows for synchronization of data sampling with the application. This dialog allows the user to configure the sampling interval.

NOTE

The frequency-based scheduler that you specify must be running by the time you connect NightProbe.

Scheduler Key

A positive integer value that identifies an existing frequency-based scheduler.

Select...

Presents the Select Frequency-Based Scheduler dialog (see “Select Frequency Based Scheduler” on page 5-4) to select a scheduler that has already been set up and configured.

Starting Cycle

This field allows you to specify the first *minor cycle* in which the specified program is to be wakened in each *major frame*.

The minor cycle is the smallest unit of frequency maintained by the frequency–based scheduler. A minor cycle has associated with it a duration, which is the time that elapses between interrupts generated by the timing source that is attached to the scheduler.

A major frame is defined as one pass through all of the minor cycles with which a frequency–based scheduler is configured.

Period

Specifies the frequency with which a specified program is to be wakened in each major frame. A period of one indicates that the program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles; and so on.

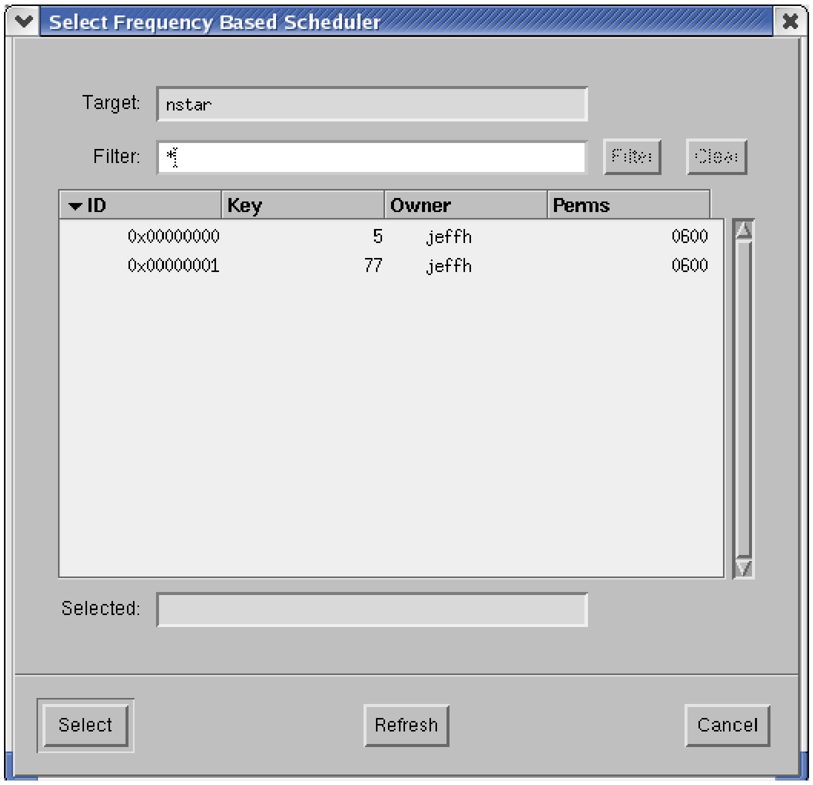
See "Timer Selection" on page 3-12 for more information.

Select Frequency Based Scheduler

The Select Frequency Based Scheduler dialog is presented:

- when the user presses the Select... button for the Scheduler Key field on the Set FBS Timer dialog (see "Set FBS Timer" on page 5-3)
- when the user presses the Select... button for the Key field on the Program Output dialog (see "Program Output" on page 6-14)

The Select Frequency Based Scheduler dialog is shown in Figure 5-3:



Target: nstar

Filter: *

ID	Key	Owner	Perms
0x00000000	5	jeffh	0600
0x00000001	77	jeffh	0600

Selected:

Figure 5-3. Select Frequency Based Scheduler dialog

Choosing the desired scheduler frosm the list populates the Selected field appropriately; pressing the Select button propagates the choice to the parent dialog.

Set Trigger Timer

Selecting the `Trigger API...` menu option from the `Timer` icon in the Session Overview Area (see "Session Overview Area" on page 3-12) or from the `Timer` menu of the NightProbe main window (see "Timer" on page 3-4) launches the `Set Trigger Timer` dialog.

The NightProbe Trigger API provides a means for synchronizing data capture in a probed application. While it may be convenient to synchronize samples from the probed application, the NightProbe Trigger Client can be any application running on the target system, including, for example, a custom cyclic scheduler, or a program which monitors interrupts from a device on the system. The Trigger Client opens a connection to the NightProbe Trigger Server Queue via a call to `np_trigger_open()` and "triggers" samples using the `np_trigger()` call. See "NightProbe Trigger API" on page 15-20 for more information.

The `Set Trigger Timer` dialog allows the user to specify the name of the NightProbe Trigger Server Queue on the target system corresponding to the name specified in the `np_trigger_open()` call in the NightProbe Trigger Client.

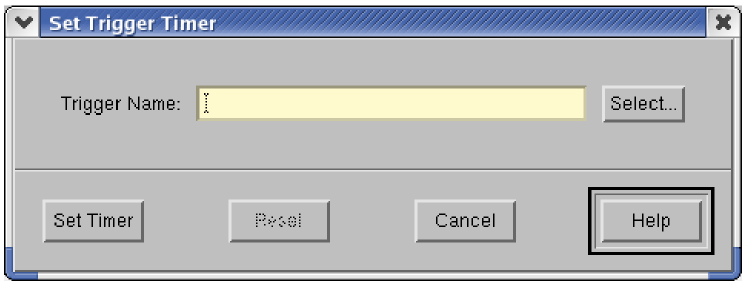


Figure 5-4. Set Trigger Timer dialog

Trigger Name

The name of the NightProbe Trigger Server Queue on the target system which corresponds to the name specified in the `np_trigger_open()` call in the NightProbe Trigger Client. See "NightProbe Trigger API" on page 15-20 for more information.

Select...

Presents the `Select Synchronized Trigger` dialog allowing the user to select the desired Trigger Server Queue from the target system.

6 Output Configuration

The selection of the output methods controls where sampled data is sent or viewed.

There are five output methods available:

- File output (see “File Output” on page 6-1)
- NightTrace output (see “NightTrace Output” on page 6-3)
- List Viewer output (see “List Viewer” on page 6-10)
- Spreadsheet Viewer output (see “Spreadsheet Viewer” on page 6-13)
- Program output (see “Program Output” on page 6-14)

File Output

The File Output dialog is launched by selecting the File Output... menu item from the Outputs icon menu (see “Outputs” on page 3-14) or from the Output menu (see “Output” on page 3-4) on the NightProbe main window.

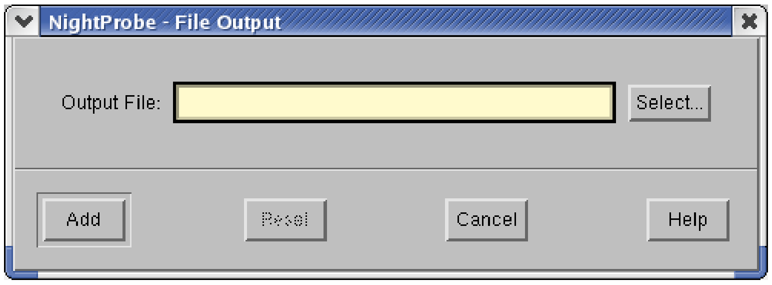


Figure 6-1. File Output dialog

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

Output File

Pathname of the file to be used as a destination for the output of the sampler.

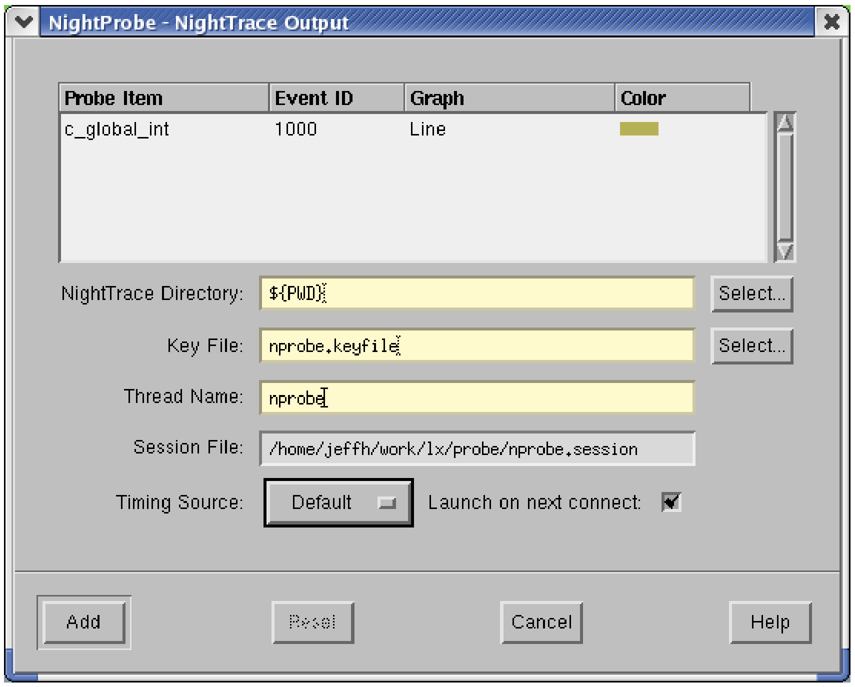
Select...

Presents a file selection dialog to select a new or existing file.

The data recording output file subsequently can be processed using the NightProbe Datastream API (see "NightProbe Datastream API" on page 15-1), can be viewed using the `List Window` dialog from within NightProbe (see "List Viewer" on page 6-10), or can be displayed with the **`nprobe --input`** option (see page 2-2).

NightTrace Output

The NightTrace Output dialog is launched by selecting the NightTrace Output... menu item from the Outputs icon menu (see “Outputs” on page 3-14) or from the Output menu (see “Output” on page 3-4) on the NightProbe main window.



Probe Item	Event ID	Graph	Color
c_global_int	1000	Line	🟨

NightTrace Directory: ${PWD}
Key File: nprobe.keyfile
Thread Name: nprobe
Session File: /home/jeffh/work/lx/probe/nprobe.session
Timing Source: Default
Launch on next connect: ✓

Figure 6-2. NightTrace Output dialog

This output option allows you 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 NightTrace. 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.

The top portion of this dialog contains a list of items being probed. For each item, the following information is displayed:

Probe Item

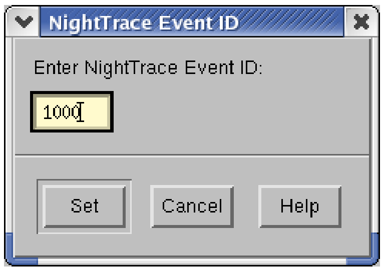
The name of the variable being monitored.

Event ID

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

Double-clicking on the event ID displays the NightTrace Event ID dialog allowing the user to change the value of the unique identifier for the trace events associated with this particular variable.

The NightTrace Event ID dialog is shown in Figure 6-3:



Enter NightTrace Event ID: 1000

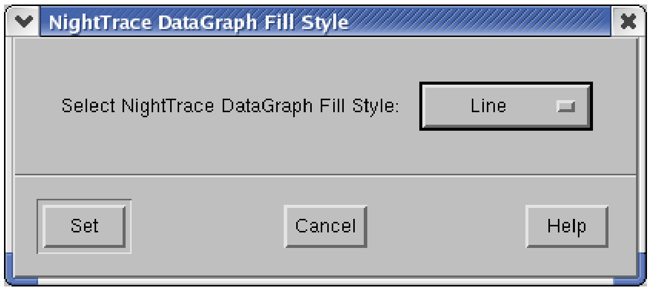
Figure 6-3. NightTrace Event ID dialog

Graph

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.

Double-clicking in this column displays the NightTrace Graph Style dialog which allows the user to select the graph style for this particular variable.

The NightTrace Graph Style dialog is shown in Figure 6-4:



Select NightTrace DataGraph Fill Style: Line

Figure 6-4. NightTrace Graph Style dialog

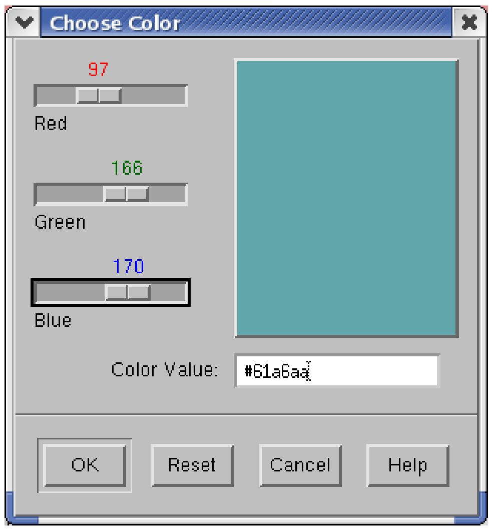
Color

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

Double-clicking on the color presents the `Choose Color` dialog allowing the user to change the color of the lines or bars associated with this particular variable in the Data Graph on the NightTrace display page.

The color can be selected using sliders that control the amount of each of the red, green, or blue components. Alternately, a color value (e.g. `#b22222`) or name (e.g. `firebrick`) can be entered in the `Color Value` field.

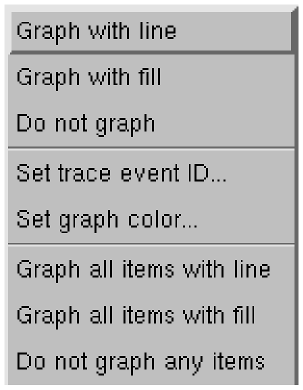
The `Choose Color` dialog is shown in Figure 6-5:



Color Value: #61a6aa

Figure 6-5. Choose Color dialog

Right-clicking on one of the items in this top portion displays the following pop-up menu for that item:



Graph with line

Values for the particular Probe Item associated with this pop-up menu will be graphed as lines of varying height in the Data Graph on the NightTrace display page associated with this session of NightProbe. The height of the line reflects the value of the variable.

Graph with fill

Values for the particular Probe Item associated with this pop-up menu will be graphed as bars of varying height in the Data Graph on the NightTrace display page associated with this session of NightProbe. The height of the bar reflects the value of the variable.

Do not graph

Specifies that the values for the particular Probe Item associated with this pop-up menu will *not* be graphed in the Data Graph on the NightTrace display page associated with this session of NightProbe.

Set trace event ID...

Displays the NightTrace Event ID dialog (see Figure 6-3) allowing the user to change the value of the unique identifier for the trace events associated with the particular Probe Item associated with this pop-up menu.

Set graph color...

Displays the Choose Color dialog allowing the user to change the color of the lines or bars in the Data Graph on the NightTrace display page associated with the particular Probe Item associated with this pop-up menu.

Graph all items with line

Values for all `Probe Items` will be graphed as lines of varying height in the Data Graph on the NightTrace display page associated with this session of NightProbe. The height of the line reflects the value of the variable.

Graph all items with fill

Values for all `Probe Items` will be graphed as bars of varying height in the Data Graph on the NightTrace display page associated with this session of NightProbe. The height of the bar reflects the value of the variable.

Do not graph any items

Specifies that none of the values for any `Probe Items` will be graphed in the Data Graph on the NightTrace display page associated with this session of NightProbe.

The bottom portion of the dialog contains the following fields:

NightTrace 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.

Key File

The `Key File` helps to synchronize the transfer of data from NightProbe to Night-Trace. 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.

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 `Thread Name` for data being streamed to NightTrace from NightProbe is `nprobe`.

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.

Timing Source

Allows the user to select between the system clock and the RCIM tick clock as the timing mechanism used for trace record timestamps.

Launch on next connect

When this option is selected, NightTrace will be started the next time that NightProbe connects to the target application (see "Sampler Control Area" on page 3-10).

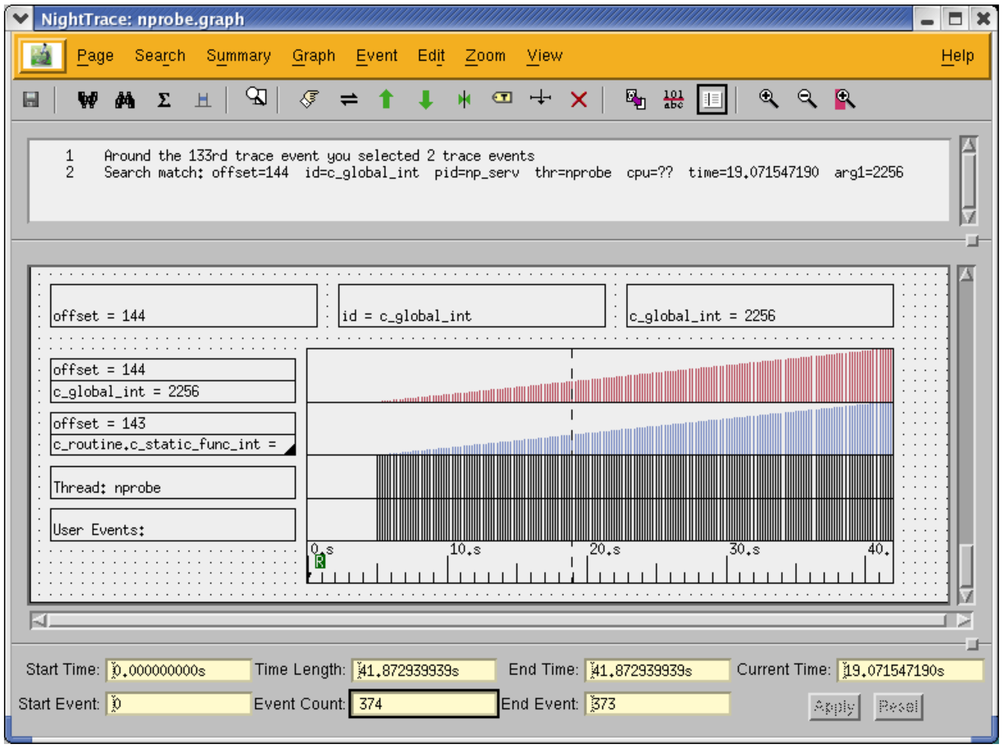
You may then start the daemon either through the NightTrace GUI or by invoking the user daemon **`ntraceud(1)`** from the command line.

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 Analysis` menu item from the `Tools` menu on the NightProbe main window (see "Tools" on page 3-6). See the *NightTrace User's Guide* for more information on this tool.

Figure 6-2 shows a NightTrace display page generated using the `NightTrace Output` method. Data Graphs for two variables appear on the grid. The Data Graph associated with the variable `c_global_int` contains red bars of varying heights; the Data Graph associated with the variable `c_static_int` consists of blue lines of varying heights.



```
1    Around the 133rd trace event you selected 2 trace events
2    Search match: offset=144  id=c_global_int  pid=np_serv  thr=nprobe  cpu=??  time=19.071547190  arg1=2256
```

offset = 144 | id = c_global_int | c_global_int = 2256

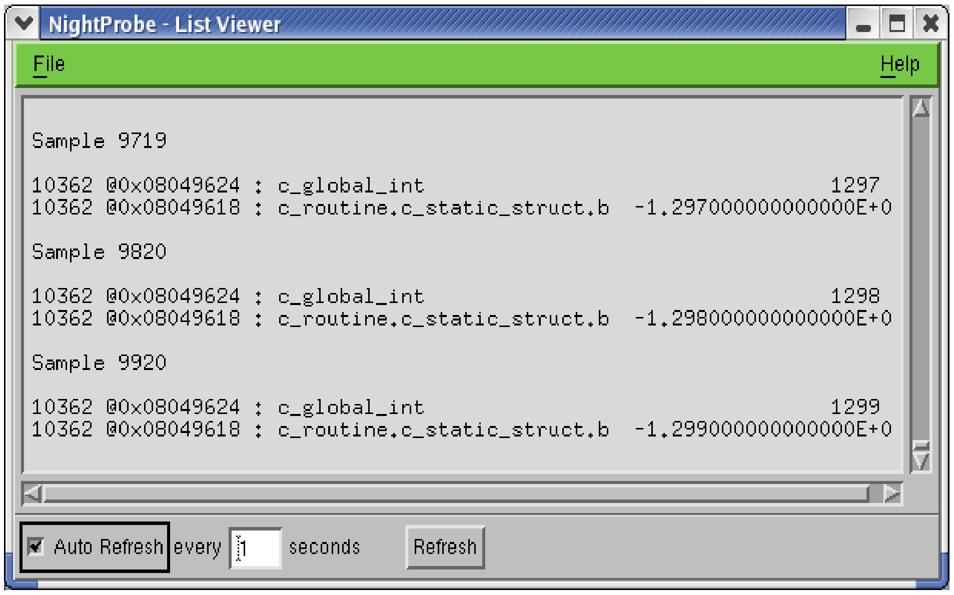
offset = 144
c_global_int = 2256
offset = 143
c_routine.c_static_func_int =
Thread: nprobe
User Events:

Start Time: 0.000000000s Time Length: 41.872939939s End Time: 41.872939939s Current Time: 19.071547190s
Start Event: 0 Event Count: 374 End Event: 373

Figure 6-6. NightTrace display page

List Viewer

The List Viewer window is launched by selecting the List Window Output menu item from the Outputs menu (see "Outputs" on page 3-14) or from the Output menu (see "Output" on page 3-4) on the NightProbe main window.



```
Sample 9719

10362 @0x08049624 : c_global_int                              1297
10362 @0x08049618 : c_routine.c_static_struct.b  -1.297000000000000E+0

Sample 9820

10362 @0x08049624 : c_global_int                              1298
10362 @0x08049618 : c_routine.c_static_struct.b  -1.298000000000000E+0

Sample 9920

10362 @0x08049624 : c_global_int                              1299
10362 @0x08049618 : c_routine.c_static_struct.b  -1.299000000000000E+0
```

Figure 6-7. List Viewer window

The List Viewer window (shown in Figure 6-7) 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:

- Menu Bar (see "Menu Bar" on page 6-11)
- Viewing Area
- Control Area (see "Control Area" on page 6-12)

Menu Bar

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

File

Mnemonic: F

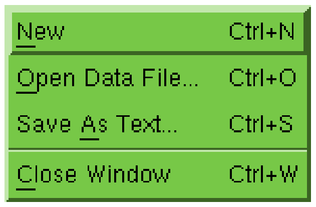


Figure 6-8. File menu

New

Mnemonic: N
Accelerator: Ctrl+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: Ctrl+O

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

Save As Text...

Mnemonic: A
Accelerator: Ctrl+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: Ctrl+W

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

Help

Mnemonic: H

The Help menu operates exactly like the menu provided in the NightProbe main 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 2-3 for details.

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. When using the On Demand timing source (see "On Demand" on page 3-13), sample data is displayed each time a new sample is recorded. Because intermediate refreshes of the display are not necessary when using On Demand timing, the Control Area is unavailable while connected. The Control Area will remain available using any other timing source.

Auto Refresh

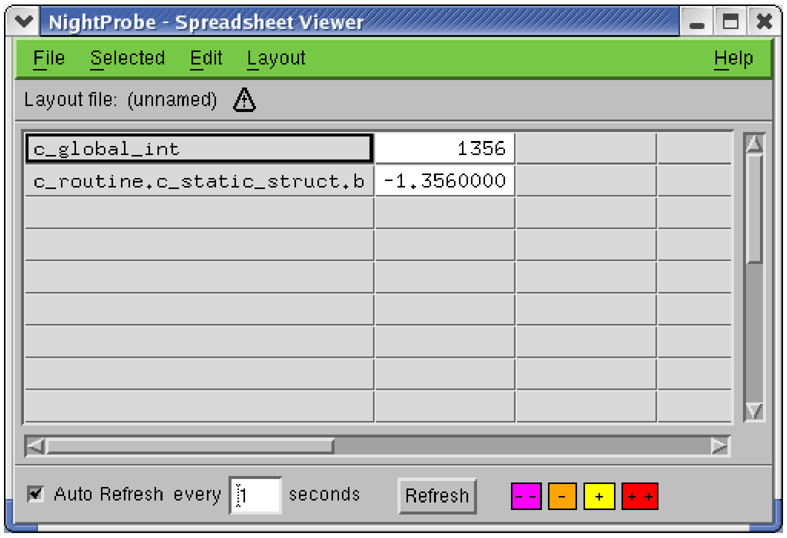
The Auto Refresh checkbox and text entry field control how often the sampled values are displayed. The List Viewer is designed for displaying values at human-readable rates, not necessarily displaying all sampled values (especially if the sampling rate is extremely fast).

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.*

Spreadsheet Viewer

The Spreadsheet Viewer window is launched by selecting the Spreadsheet Output menu item from the Outputs icon menu (see “Outputs” on page 3-14) or from the Output menu (see “Output” on page 3-4) on the NightProbe main window.



c_global_int	1356		
c_routine.c_static_struct.b	-1.3560000		

Auto Refresh every 1 seconds

Figure 6-9. Spreadsheet Viewer window

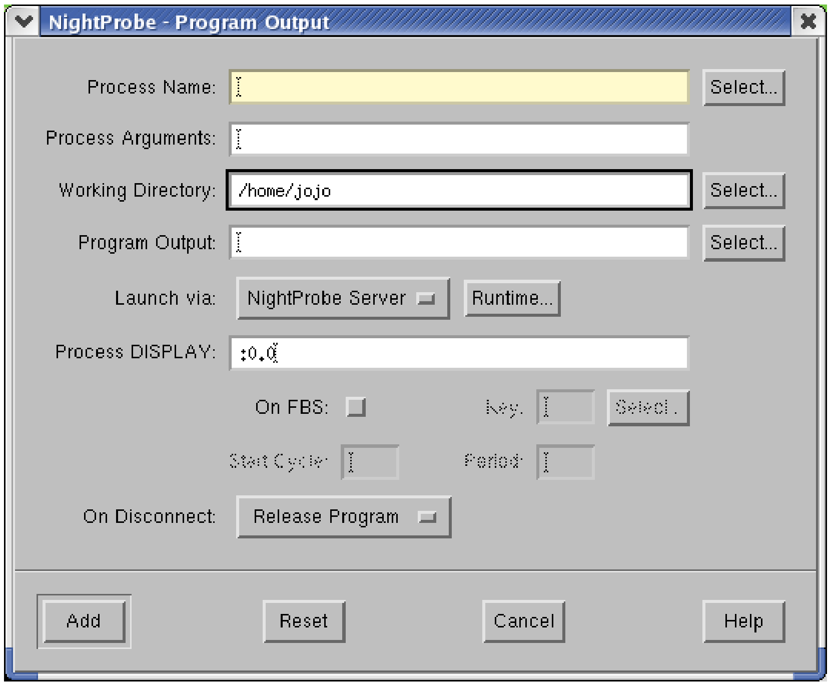
The Spreadsheet Viewer as shown in Figure 6-9, allows you to monitor and modify variables while the program is running. The Spreadsheet Viewer window has many configuration options and is described in detail in its own chapter, Chapter 14, “Spreadsheet Viewer”.

Program Output

Mnemonic: P

The Program Output dialog is launched by selecting the Program Output menu item from the Outputs icon menu (see “Outputs” on page 3-14) or from the Output menu (see “Output” on page 3-4) on the NightProbe main window.

This dialog allows you to specify a program that will be used to process the data recording output using the NightProbe Datastream API (see “NightProbe Datastream API” on page 15-1).


NightProbe - Program Output

Process Name:

Process Arguments:

Working Directory: /home/jojo

Program Output:

Launch via: NightProbe Server

Process DISPLAY: :0.0

On FBS: ☐

Key:

Start Cycle:

Period:

On Disconnect: Release Program

Figure 6-10. Program Output dialog

Process 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.

Process Arguments

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

Working Directory

The current working directory for the program.

Select...

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

NOTE

Pathnames are relative to the host system.

Program Output

If specified, **`stdout`** from the program used to process the data recording output is redirected to this file.

Select...

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

NOTE

Pathnames are relative to the host system.

Launch via

Indicates where the program used to process the data recording output should be executed:

NightProbe GUI

where the GUI portion of NightProbe is running

NightProbe Server

where the probed application is running

Runtime...

Presents the Run Time Settings dialog (see "Run Time Settings" on page 4-3) allowing the user to specify the scheduling policy, priority, and CPU bias for the program used to process the data recording output streamed from NightProbe.

Process DISPLAY

When an output program uses the X Window System for display purposes, a display device is required. This field allows the user to specify the X display to be used by the output program on the target system. The value entered here will be set in the `DISPLAY` environment variable of the program being invoked.

This field is not relevant to output programs which do not use the X Window System.

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.

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 Select Frequency Based Scheduler dialog (see "Select Frequency Based Scheduler" on page 5-4) allowing you 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.

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 “Sampler Control Area” on page 3-10).

Terminate Program

Terminates with `SIGTERM` the program which processed the data streamed from NightProbe after NightProbe has disconnected from the target program (see “Sampler Control Area” on page 3-10).

7
Program Window

The Program Window specifies a program to be probed.

It is launched by selecting the Add Program... menu item from the Resources icon menu (see “Resources” on page 3-16) or from the Resource menu (see “Resource” on page 3-5) on the NightProbe main window.

The Program Window is shown in Figure 7-1.

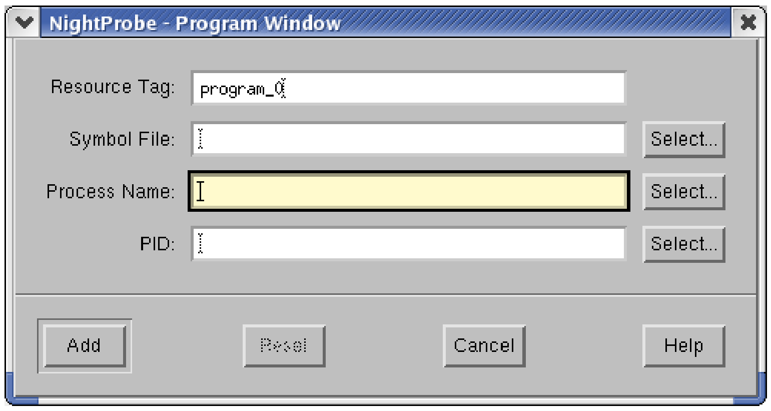


Figure 7-1. Program Window

Resource Tag

A Resource Tag acts as an identifier for the resource. It is used to maintain relationships between variables and resources in the session configuration file so that configurations may be saved and reloaded in the future.

A resource tag begins with an alphabetic character and is followed by a contiguous series of non-blank characters.

A unique resource tag is assigned when the window is launched. The default value of this tag corresponds to the basename of the value used in the Process Name field when the Add button is pressed.

If the tag name is modified, it will be adjusted if it conflicts with another resource. In such cases, NightProbe will append an underscore and number to the value of the Resource Tag field when the Add button is pressed and pop up a dialog indicating the conflict and tag adjustment.

Furthermore, if a resource tag containing illegal characters is entered, NightProbe will construct a legitimate alternative and ask the user for approval.

Symbol File

A Symbol File is an executable file which contains symbolic and debug information that allows NightProbe to identify and list variables that can be probed.

A Symbol File is not strictly required for probing a process, but without it, you can only define artificial variables using the Item Definition dialog.

If no Symbol File is specified, but a filename is specified for the Process Name, NightProbe will automatically attempt to use that filename as the symbol file.

If you specify a file name for the Symbol File and have not specified a value for the Process Name, NightProbe automatically fills in the Process Name with the specified file name.

If specified, the Symbol File must exist, be accessible from the host system, and be a valid executable program file containing symbolic debugging information.

Select...

Presents the Select Symbol File for Resource dialog, a file selection dialog allowing the user to navigate to the directory where the symbol file is located and select it.

Process Name

A Process Name is required in order to probe programs. The Process Name is used to identify running processes when no PID is specified, or when the specified PID cannot be found.

When attempting to connect, if the specified PID is not found, NightProbe will attempt to locate a process that matches the Process Name. If a matching process can be located, NightProbe pops up a dialog indicating the situation and asks if the connection should proceed or be terminated.

Normally, the Process Name and Symbol File are the same but they do not have to be. The Process Name could be a file name of a stripped executable. The Process Name file does not have to exist on the host system.

The Process Name is the only required piece of information that must be entered in this dialog. It is normally sufficient to just type in the name of the executable file the Process Name text field or select it using the Select... file dialog and then click Add to exit the dialog.

Select...

Presents the Select Pathname for Resource dialog, a file selection dialog allowing the user to navigate to the directory where the executable is located and select it.

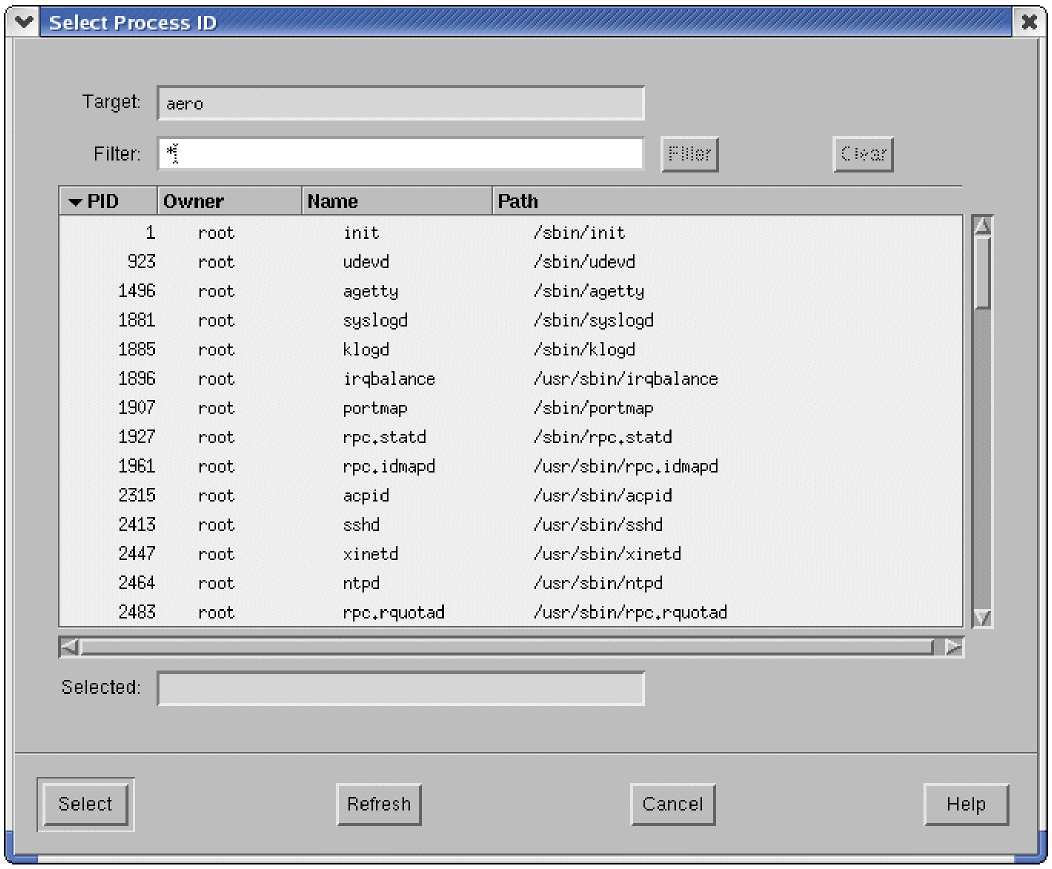
PID

The PID is an optional field which specifies a specific process ID to probe.

If the PID is left blank or if the specified PID is not running when you connect, NightProbe will automatically attempt to locate a running process that matches the Process Name. If the PID was blank, NightProbe silently proceeds to connect using the process ID it located. If the PID was not blank, NightProbe will pop up a dialog indicating the newly located PID and ask whether the connection should proceed or be terminated.

Select...

The Select... button brings up the Select Process ID dialog, which presents a list of the process IDs, owner user names, and process names running on the target system and allows you to select one for monitoring. Selecting a file in the Select Process ID dialog automatically fills in the Resource Tag, Symbol File, Process Name and PID in the Program Window.



Target: aero

Filter: *

PID	Owner	Name	Path
1	root	init	/sbin/init
923	root	udevd	/sbin/udevd
1496	root	agetty	/sbin/agetty
1881	root	syslogd	/sbin/syslogd
1885	root	klogd	/sbin/klogd
1896	root	irqbalance	/usr/sbin/irqbalance
1907	root	portmap	/sbin/portmap
1927	root	rpc.statd	/sbin/rpc.statd
1961	root	rpc.idmapd	/usr/sbin/rpc.idmapd
2315	root	acpid	/usr/sbin/acpid
2413	root	sshd	/usr/sbin/sshd
2447	root	xinetd	/usr/sbin/xinetd
2464	root	ntpd	/usr/sbin/ntpd
2483	root	rpc.rquotad	/usr/sbin/rpc.rquotad

Selected:

Figure 7-2. Select Process ID dialog

Filters are patterns constructed using standard regular expression syntax as defined by **regex(7)**. The default Filter is “*” which means that all processes are displayed. An empty filter shows all processes as well.

Clicking on the PID, Owner, or Name headings will sort the list of processes using that field as the sort key. Or, if that field already was the sort key, it will reverse the order of the sort.

When you use the Select Process ID dialog to select a program, NightProbe uses information available in the target's **/proc** file system to obtain the full pathname of the executable program on the target file system. If this pathname is not the same on the host file system, you may need to modify the selected pathname.

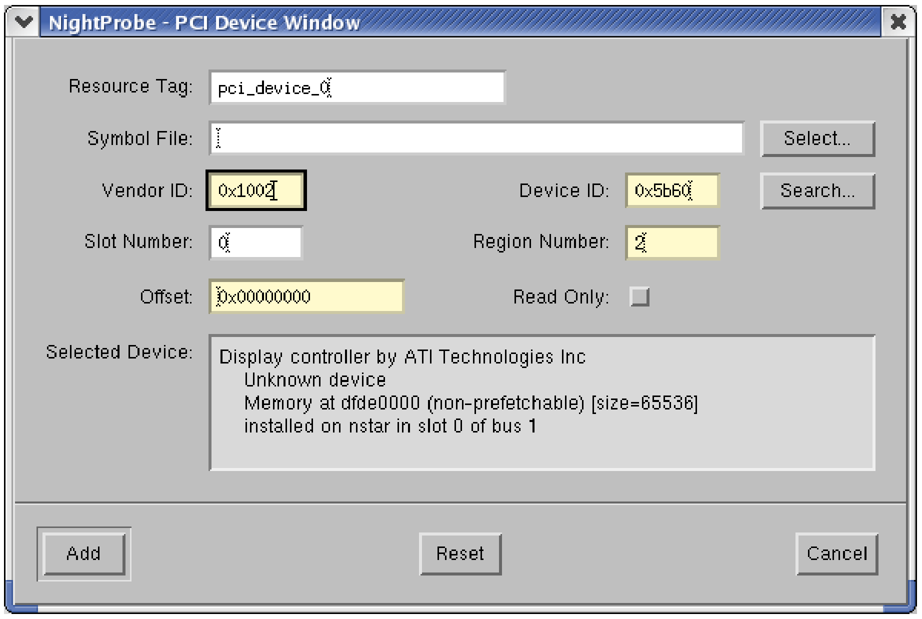
If the executable file selected has no symbolic debug information within (for example, if it is a stripped executable file), you will need to specify a symbol file pathname, relative to the host system, containing the additional symbolic debugging information needed to load symbol information.

8
PCI Device Window

The PCI Device Window specifies a PCI device to be monitored or recorded.

It is launched by selecting the Add PCI Device... menu item from the Resources icon menu (see “Resources” on page 3-16) or from the Resource menu (see “Resource” on page 3-5) on the NightProbe main window.

The PCI Device Window is shown in Figure 8-1.



Resource Tag: pci_device_0
Symbol File:
Vendor ID: 0x1002
Device ID: 0x5b60
Slot Number: 0
Region Number: 2
Offset: 0x00000000
Read Only: ☐
Selected Device:

```
Display controller by ATI Technologies Inc
    Unknown device
    Memory at dfde0000 (non-prefetchable) [size=65536]
    installed on nstar in slot 0 of bus 1
```

Figure 8-1. PCI Device WIndow

The PCI Device Window is only available for target systems running RedHawk Linux. PCI device mapping depends on the Base Address Register file system extension to **/proc/bus/pci**, which is only available under RedHawk Linux. (**bar_scan_open(3)**)

Resource Tag

A Resource Tag acts as an identifier for the resource. It is used to maintain relationships between variables and resources in the session configuration file so that configurations may be saved and reloaded in the future.

A resource tag begins with an alphabetic character and is followed by a contiguous series of non-blank characters.

A unique resource tag is assigned when the window is launched.

If the tag name is modified, it will be adjusted if it conflicts with another resource. In such cases, NightProbe will append an underscore and number to the value of the Resource Tag field when the Add button is pressed and pop up a dialog indicating the conflict and tag adjustment.

Furthermore, if a resource tag containing illegal characters is entered, NightProbe will construct a legitimate alternative and ask the user for approval.

Symbol File

A Symbol File is an executable file which contains symbolic and debug information that allows NightProbe to identify and list types that can be associated with artificial variables you associate with the PCI device.

A Symbol File is not required for probing a PCI device, but without it, you can only define artificial variables using basic numeric and character types in the Item Definition dialog.

If specified, the Symbol File must exist, be accessible from the host system, and be a valid executable program file.

PCI Device Specification

PCI devices are probed by mapping regions of PCI memory into the address space of the sampler process.

Since the memory addresses of devices on the PCI bus are dynamic in nature, the specification of the device itself is the preferred approach.

Alternatively, if the address is known, you can use the Memory Mapped dialog using **/dev/mem** as the device.

NightProbe can only probe PCI devices with mappable register or memory regions. It cannot probe I/O ports.

Region numbers start at zero for each PCI device and are monotonically increasing.

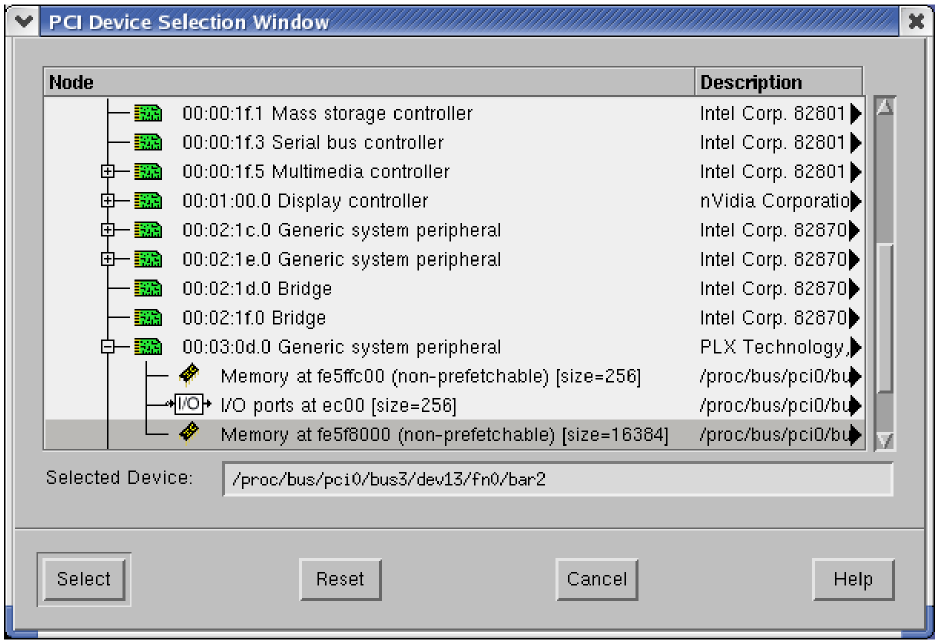
To specify a PCI device, the following pieces of information are required: Vendor ID, Device ID, Region Number, and Slot Number.

These fields only accept numeric input and are limited to values in the range 0x0 .. 0xffff. A Region Number of zero corresponds to the first mappable region for the device.

The Slot Number is not always required. It is required only to differentiate between two or more devices having identical Vendor and Device IDs installed on the same system. If it is left blank, when connecting and their are multiple such devices, NightProbe will allow you to select the appropriate device using the PCI Device Selection window. If you choose not to select a device, NightProbe will issue a diagnostic and terminate the connection.

If these values are readily available, you can enter them in the text fields.

Alternatively, pressing the Search... button will launch the PCI Device Selection window:



Node	Description
00:00:1f.1 Mass storage controller	Intel Corp. 82801
00:00:1f.3 Serial bus controller	Intel Corp. 82801
00:00:1f.5 Multimedia controller	Intel Corp. 82801
00:01:00.0 Display controller	nVidia Corporatio
00:02:1c.0 Generic system peripheral	Intel Corp. 82870
00:02:1e.0 Generic system peripheral	Intel Corp. 82870
00:02:1d.0 Bridge	Intel Corp. 82870
00:02:1f.0 Bridge	Intel Corp. 82870
00:03:0d.0 Generic system peripheral	PLX Technology,
Memory at fe5ffc00 (non-prefetchable) [size=256]	/proc/bus/pci0/bu
I/O ports at ec00 [size=256]	/proc/bus/pci0/bu
Memory at fe5f8000 (non-prefetchable) [size=16384]	/proc/bus/pci0/bu

Selected Device: /proc/bus/pci0/bus3/dev13/fn0/bar2

Figure 8-2. PCI Device Selection Window

The PCI Device Selection Window presents an interactive PCI device browser containing a list of PCI devices and their mappable regions organized in a tree.

The tree contains a list of all PCI devices installed on the target system.

PCI devices having mappable registers, or regions, are shown with an expandable control box to the left of their icon. Click the control box to see the mappable regions for each device.

The Select button will remain desensitized until you select a mappable region node. Since devices without mappable regions are not able to be probed by NightProbe, the Select button will remain desensitized if you select a PCI device node.

If the Vendor ID, Device ID, or Slot Number text fields of the PCI Device Window contain data when the Search... button is pressed, a message dialog asks whether you wish to filter the list of PCI devices by the values of those fields or would prefer to see the complete list of PCI devices on the target system.

If you select the filtering option, but no PCI devices match the supplied values, a diagnostic will be issued and the entire list of PCI devices will be presented in the tree.

Offset

By default, the Offset value field is set to zero. This value defines the base offset from the beginning of the selected PCI device memory region for all artificial variables to be associated with the region. The Offset in the Item Definition dialog is added to the base offset defined here.

Read Only

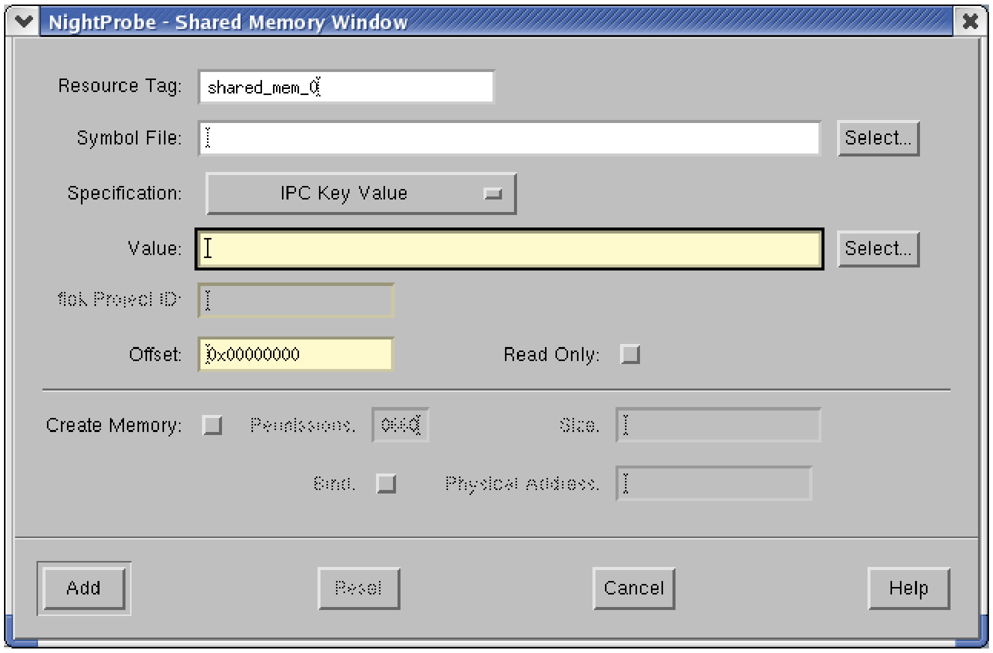
Optionally, the Read Only checkbox may be activated. If selected, the mapping to the PCI device's memory region will be done in read-only mode to prevent accidental modification when using the Spreadsheet Viewer.

9
Shared Memory Window

The Shared Memory Window specifies a shared memory segment to be monitored or recorded.

It is launched by selecting the Add Shared Memory... menu item from the Resources icon menu (see “Resources” on page 3-16) or from the Resource menu (see “Resource” on page 3-5) on the NightProbe main window.

The Shared Memory Window is shown in Figure 9-1.



Resource Tag: shared_mem_0
Symbol File:
Specification: IPC Key Value
Value:
ftok Project ID:
Offset: 0x00000000
Read Only: ☐
Create Memory: ☐
Permissions: 0660
Size:
Bind: ☐
Physical Address:

Figure 9-1. Shared Memory WIndow

Resource Tag

A Resource Tag acts as an identifier for the resource. It is used to maintain relationships between variables and resources in the session configuration file so that configurations may be saved and reloaded in the future.

A resource tag begins with an alphabetic character and is followed by a contiguous series of non-blank characters.

A unique resource tag is assigned when the window is launched.

If the tag name is modified, it will be adjusted if it conflicts with another resource. In such cases, NightProbe will append an underscore and number to the value of the Resource Tag field when the Add button is pressed and pop up a dialog indicating the conflict and tag adjustment.

Furthermore, if a resource tag containing illegal characters is entered, NightProbe will construct a legitimate alternative and ask the user for approval.

Symbol File

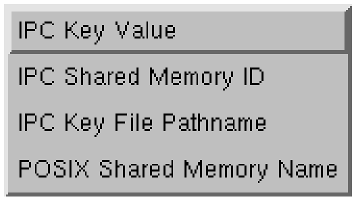
A Symbol File is an executable file which contains symbolic and debug information that allows NightProbe to identify and list types that can be associated with artificial variables you create and associate with the shared memory segment.

A Symbol File is not required for probing a shared memory segment, but without it, you can only define artificial variables using basic numeric and character types in the Item Definition dialog.

If specified, the Symbol File must exist, be accessible from the host system, and be a valid executable program file.

Specification

The Specification field contains a menu list describing the method for identifying the shared memory segment:

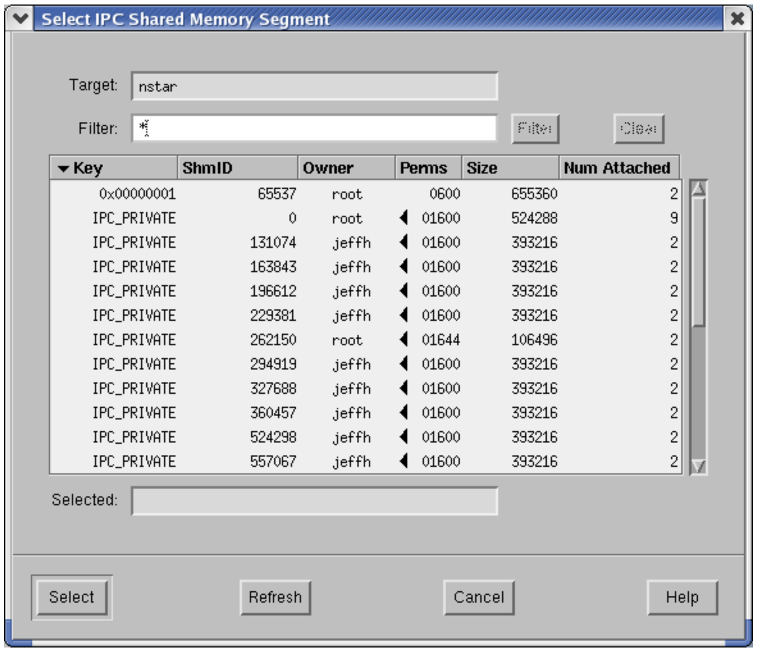


IPC Key Value

When this menu option is selected, the value supplied in the Value text field will be interpreted as a shared memory key as supplied to the **`shmget(2)`** system call. Specification of an IPC_PRIVATE key value is inappropriate and is disallowed.

The Select... button to the right of the Value text field presents the Select IPC Shared Memory Segment dialog which provides a list of all existing shared memory segments on the target system.

The Select IPC Shared Memory Segment dialog is shown in Figure 9-2.



Target: nstar

Filter: *

Key	ShmID	Owner	Perms	Size	Num Attached
0x00000001	65537	root	0600	655360	2
IPC_PRIVATE	0	root	01600	524288	9
IPC_PRIVATE	131074	jeffh	01600	393216	2
IPC_PRIVATE	163843	jeffh	01600	393216	2
IPC_PRIVATE	196612	jeffh	01600	393216	2
IPC_PRIVATE	229381	jeffh	01600	393216	2
IPC_PRIVATE	262150	root	01644	106496	2
IPC_PRIVATE	294919	jeffh	01600	393216	2
IPC_PRIVATE	327688	jeffh	01600	393216	2
IPC_PRIVATE	360457	jeffh	01600	393216	2
IPC_PRIVATE	524298	jeffh	01600	393216	2
IPC_PRIVATE	557067	jeffh	01600	393216	2

Selected:

Figure 9-2. Select IPC Shared Memory Segment dialog

NOTE

Clicking on the Key, ShmID, Owner, Perms, Size or Num Attached headings will re-sort the list of processes using that field as the sort key. Or, if that field already was the sort key, it will reverse the order of the sort.

Filters are patterns constructed using standard regular expression syntax as defined by **`regex(7)`**. The default Filter is "*" which means that all shared memory segments are displayed. An empty filter shows all shared memory segments as well.

IPC Shared Memory ID

When this menu option is selected, the value supplied in the Value text field will be interpreted as a shared memory identifier as returned from the **`shmget(2)`** system call.

The Select… button to the right of the Value text field presents the Select IPC Shared Memory Segment dialog (see Figure 9-2) which provides a list of all existing shared memory segments on the target system.

IPC Key File Pathname

When this menu option is selected, the value supplied in the Value text field will be interpreted as a pathname. The shared memory segment will be identified using the **ftok(3)** service using the specified pathname and ftok Project ID value in the text field below the Value text field.

The Select... button to the right of the Value text field presents the Select IPC Key File Pathname for Resource dialog which provides a standard file selection dialog to specify the pathname.

NOTE

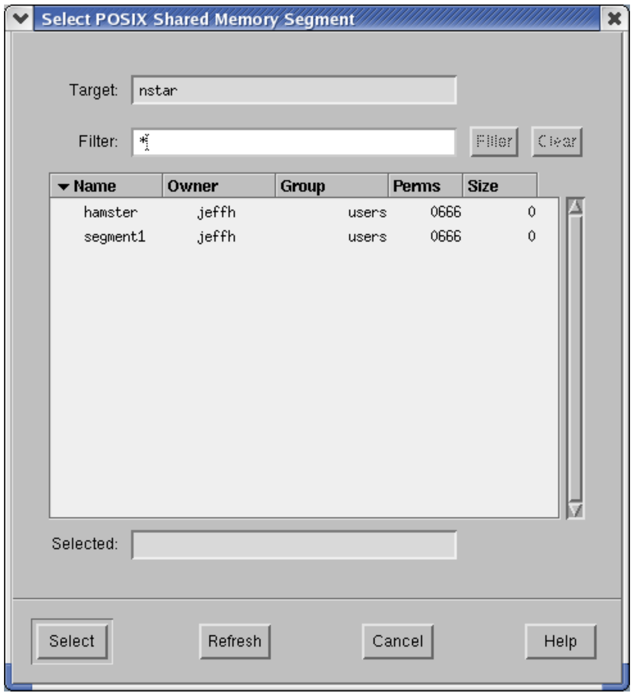
The filenames provided in the file selection dialog are relative to the host system, even though the selected file must be accessible from the target system when NightProbe connects to the target system.

POSIX Shared Memory Name

When this menu option is selected, the value supplied in the Value text field will be interpreted as POSIX shared memory object as created by the **shm_open(3)** service.

The Select... button to the right of the Value text field presents the Select POSIX Shared Memory Segment dialog which provides a list of all existing named POSIX shared memory objects.

The Select POSIX Shared Memory Segment dialog is shown in Figure 9-3.



Target: nstar

Filter: *

Name	Owner	Group	Perms	Size
hamster	jeffh	users	0666	0
segment1	jeffh	users	0666	0

Figure 9-3. Select POSIX Shared Memory Segment dialog

Filters are patterns constructed using standard regular expression syntax as defined by **`regex(7)`**. The default Filter is "*" which means that all POSIX shared memory objects are displayed. An empty filter shows all POSIX shared memory objects as well.

Value

The contents of this field will be interpreted based on the selection from the Specification drop-down as described above.

Select...

Pressing this button presents a dialog allowing the user to select an existing shared memory object or to specify a key file pathname (based on the selection of the Specification as described above).

ftok Project ID

When IPC Key File Pathname is selected in the Value drop-down, this field allows the user to provide a `proj_id` parameter for the **`ftok(3)`** key translation in order to generate a System V IPC key.

The least significant 8 bits (1..255) are used to designate a project ID. Its value must be non-zero. The user is allowed to specify it here because their application may use a specific value of `proj_id` and it is not possible to generate the same key unless the same `proj_id` is used.

The defult value is 1.

Offset

The Offset value defaults to zero. This value defines the base offset from the beginning of the shared memory segment for all variables to be associated with this shared memory segment. The Offset in the Item Definition dialog is added to the base offset defined here.

Read Only

If the Read Only checkbox is activated, the attachment of the shared memory region to the sampler process's address space will be done in read-only mode, preventing modification of the region via the Spreadsheet Viewer. Selection of the Read Only checkbox precludes selection of the Create Memory option.

Create Memory

If the Create Memory checkbox is selected, the Size of the shared memory segment must be specified. When the sampler process connects to the target system, it will create the shared memory segment if it does not already exist. If it already exists and is of insufficient size, the connection will fail with a diagnostic.

Permissions

When the Create Memory checkbox is selected, this field allows the user to specify the permissions with which to create the shared memory segment.

Size

When the Create Memory checkbox is selected, this field allows the user to specify the size (in bytes) of the shared memory segment.

Bind

If the Create Memory option is selected, you may also select the Bind option and specify a physical memory address to which to bind the created shared memory segment. If this option is specified, the address will be passed to the **`shmbind(2)`** system service during connection.

Physical Address

When the Bind checkbox is selected, this field is used to specify the physical memory address to which to bind the created shared memory segment.

WARNING

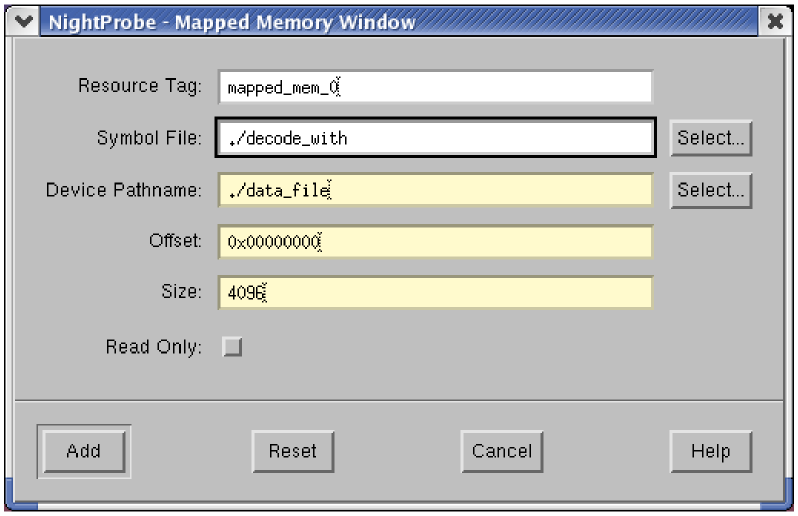
Extreme care is recommended when choosing physical addresses. Probing inappropriate physical addresses can cause system instability or crashes.

10
Mapped Memory Window

The Mapped Memory Window specifies a device or file to monitored or recorded via memory mapping.

It is launched by selecting the Add Mapped Memory... menu item from the Resources icon menu (see “Resources” on page 3-16) or from the Resource menu (see “Resource” on page 3-5) on the NightProbe main window.

The Mapped Memory Window is shown in Figure 10-1.



Resource Tag: mapped_mem_0
Symbol File: ./decode_with
Device Pathname: ./data_file
Offset: 0x00000000
Size: 4096
Read Only: ☐

Figure 10-1. Mapped Memory WIndow

Resource Tag

A Resource Tag acts as an identifier for the resource. It is used to maintain relationships between variables and resources in the session configuration file so that configurations may be saved and reloaded in the future.

A resource tag begins with an alphabetic character and is followed by a contiguous series of non-blank characters.

A unique resource tag is assigned when the window is launched.

If the tag name is modified, it will be adjusted if it conflicts with another resource. In such cases, NightProbe will append an underscore and number to the value of the Resource Tag field when the Add button is pressed and pop up a dialog indicating the conflict and tag adjustment.

Furthermore, if a resource tag containing illegal characters is entered, NightProbe will construct a legitimate alternative and ask the user for approval.

Symbol File

A Symbol File is an executable file which contains symbolic and debug information that allows NightProbe to identify and list types that can be associated with artificial variables you create and associate with the mapped memory segment.

A Symbol File is not required for probing a mapped memory segment, but without it, you can only define artificial variables using basic numeric and character types in the Item Definition dialog.

If specified, the Symbol File must exist, be accessible from the host system, and be a valid executable program file.

Device Pathname

A Device Pathname must be specified. It identifies the device or file that will be mapped into the sampler process's memory via the **`mmap(2)`** system service.

The specified pathname does not have to be accessible from the host system.

The Select... button to the right of the Device Pathname text field provides a standard file selection dialog to locate specify the pathname.

NOTE

The pathnames provided in the file selection dialog are relative to the host system, even though the selected file must be accessible from the target system when NightProbe connects to the target system.

System memory may be probed by specifying **`/dev/mem`** as the Device Pathname.

WARNING

Extreme care is recommended in probing system memory as this can easily destabilize or crash the system

Offset

An Offset must be specified. By default, the Offset value field is set to zero. This value defines the base offset from the beginning of the mapped memory segment for all variables to be associated with this mapped memory segment. The Offset in the Item Definition dialog is added to the base offset defined here.

Size

The Size of the memory segment must be specified. When NightProbe connects to the target system, if the specified size exceeds the size of the device specified in the Device Pathname text field, the connection will fail with a diagnostic.

Read Only

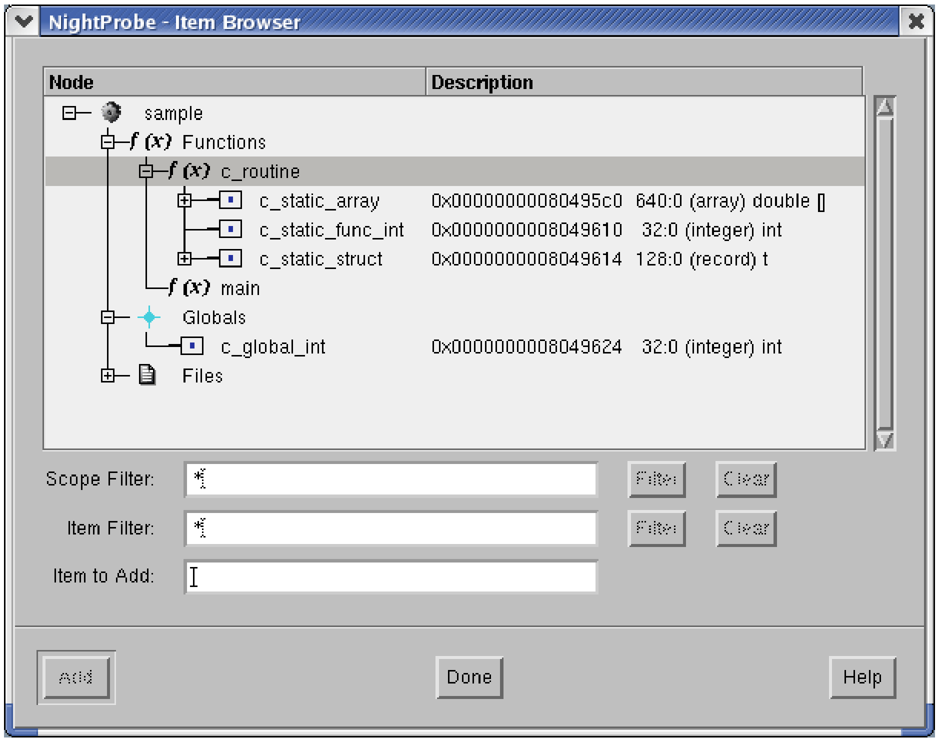
The Read Only checkbox may be activated. If activated, the memory mapping will be done in a read-only mode preventing modification of the probed file or device from the Spreadsheet Viewer window.

11
Item Browser

The Item Browser allows you to navigate the list of variables in a program symbol file and select variables to be probed.

It is launched by selecting the Add Items from Program... menu item from the Probe Items icon menu (see “Probe Items” on page 3-18) or from the Resource menu (see “Resource” on page 3-5) on the NightProbe main window.

The Item Browser is shown in Figure 11-1:



NightProbe - Item Browser

```
Node                                Description
⊟ sample
  ⊟ f(x) Functions
    ⊟ f(x) c_routine
      ⊞ c_static_array              0x00000000080495c0  640:0 (array) double []
        c_static_func_int           0x0000000008049610   32:0 (integer) int
      ⊞ c_static_struct             0x0000000008049614  128:0 (record) t
      f(x) main
  ⊟ Globals
      c_global_int                  0x0000000008049624   32:0 (integer) int
  ⊞ Files
```

Scope Filter: * [Filter] [Clear]

Item Filter: * [Filter] [Clear]

Item to Add:

[Add] [Done] [Help]

Figure 11-1. Item Browser

The top portion of the Item Browser consists of the Interactive Variable Browser (see “Interactive Variable Browser” on page 11-3) which provides expandable and collapsible lists of entities organized in a tree containing scopes, variables, and components of composite variables.

The remaining fields are described below:

Scope Filter

The `Scope Filter` allows a regular expression to be applied to the entire tree of lists that contain variables. The expression should adhere to the syntax as defined by **`regex(7)`**. The expression is not automatically bound to the start or end of an item name. Thus to filter scopes that start with "`sched_`", you would enter the regular expression:

```
^sched_.*
```

In addition, an empty filter or the pattern "`*`" show all scopes.

Item Filter

The `Item Filter` allows a regular expression to be applied to all variables in the tree. The expression should adhere to the syntax as defined by **`regex(7)`**. The expression is not automatically bound to the start or end of a name. Thus to filter variables that end with "`_counter`", you would enter the regular expression:

```
.*_counter$
```

Item filtering stops at the variable level, it is not applied to components of composite variables.

In addition, an empty filter or the pattern "`*`" show all items.

Item to Add

Variables may be directly added to the list of `Probe Items` of the NightProbe main window by typing their fully-expanded name in the `Item To Add` text field.

See "Variable Name Notation" on page C-1 for information on how to specify a variable in fully-expanded name format.

For array variables, a subscript or slice may be added to the name. See "Array Slices" on page C-2 for information on array slice syntax.

For record or structure variables, a component may be specified.

When the `Add` button is pressed, the variable is added to the `Probe Items` list in the Session Overview Area of the NightProbe main window. If the variable does not match any existing eligible variable in the symbol file, it will be added in a disabled state and its description will indicate "no matching symbol". Variables in the `Probe Items` list which are in a disabled state have a white icon.

Interactive Variable Browser

The Interactive Variable Browser appears at the top of the Item Browser and provides expandable and collapsible lists of entities organized in a tree containing scopes, variables, and components of composite variables. The root of the tree is a list of all programs that have an associated symbol file.

For each node in the tree that has children, the list can be expanded or collapsed by clicking the control box to the left of the icon.

Keyboard traversal is also supported within the tree:

Up Arrow

Pressing the Up Arrow key will cause the node immediately above the current node in a list of siblings to be selected unless the top node in a list has been reached. If the top node has been reached, the key has no effect.

Down Arrow

Pressing the Down Arrow key will cause the node immediately below the current node in a list of siblings to be selected unless the bottom node in a list has been reached. If the bottom node has been reached, the key has no effect.

Left Arrow

Pressing the Left Arrow key will cause the parent of the current list to be selected. When a root node is reached, the key has no effect.

Right Arrow

Pressing the Right Arrow key will cause automatically expand the current node and cause the first child node to become selected. If the current node is a leaf, the key has no effect.

Space

Pressing the Space toggles the expansion setting of the current node. If the current node has no children, the key has no effect.

For each resource node, there are four main nodes which provide lists of Functions, Globals, and Files, and Packages which contain scopes and variables.

Functions

The Functions list is populated with the names of identifiable functions within the symbol file. Functions that contain scopes or variables that can be recorded will have an expandable control box to their left. Fortran subroutines containing common blocks are an example of a function which contains a nested scope.

Globals

The Globals list is populated with variables that are defined in the global scope. Typically these are C and C++ variables declared as `extern`.

Files

The Files list is populated with all identifiable source files within the symbol file. Files may contain scopes, such as packages or functions, and static variables declared outside of functions.

Packages

The Packages list is populated with all identifiable library level Ada packages within the symbol file. Packages contain other packages or variables that can be probed.

Variables

Variables appear by name with boxes as their icon. If a variable is a composite type (an array, structure, or record), it will appear with an expandable control box. All record and structure components are shown when a record or structure is expanded. Array components can also be expanded, but are limited by the resource `Nprobe.text.maxArrayExpansion`, which defaults to 1000.

To add a variable to the list of Probe Items of the NightProbe main window, select the variable and click Add. Alternatively, pressing the <Enter> key or double-clicking the variable adds the selected variable as well. Pressing the <Enter> key also exits the window immediately after adding the item.

Composite variables may be added as a whole, or individual components may be added.

Variables that have been added to the Probe Items list are indicated with a orange icon.

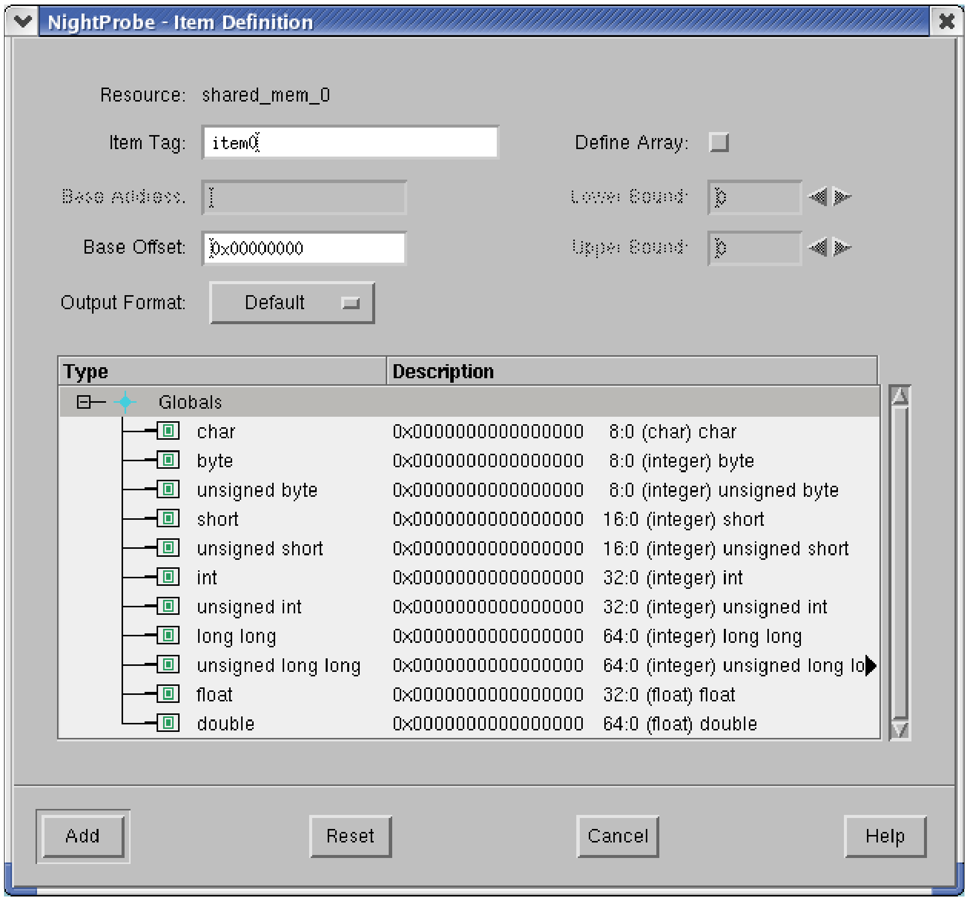
Variables in source files that were compiled without the symbolic debug option (**`-g`**) or are not elligible to be recorded will not appear in the Item Browser.

12
Item Definition Window

The Item Definition window allows you to create an artificial variable which is a view into the associated resource.

It is launched by selecting the New Item... menu item from the Probe Items icon menu (see “Probe Items” on page 3-18) or from the Resource menu (see “Resource” on page 3-5) on the NightProbe main window

The Item Definition window is shown in Figure 12-1:


NightProbe - Item Definition

Resource: shared_mem_0

Item Tag: item0

Define Array: ☐

Base Address:

Lower Bound: 0

Base Offset: 0x00000000

Upper Bound: 0

Output Format: Default

Type	Description
Globals	
char	0x0000000000000000 8:0 (char) char
byte	0x0000000000000000 8:0 (integer) byte
unsigned byte	0x0000000000000000 8:0 (integer) unsigned byte
short	0x0000000000000000 16:0 (integer) short
unsigned short	0x0000000000000000 16:0 (integer) unsigned short
int	0x0000000000000000 32:0 (integer) int
unsigned int	0x0000000000000000 32:0 (integer) unsigned int
long long	0x0000000000000000 64:0 (integer) long long
unsigned long long	0x0000000000000000 64:0 (integer) unsigned long lo
float	0x0000000000000000 32:0 (float) float
double	0x0000000000000000 64:0 (float) double

Figure 12-1. Item Definition Window

Item Tag

An Item Tag is simply a short-hand identifier for the artificial variable being defined by this dialog.

By default, a unique Item Tag is assigned when the window is launched.

If you modify the Item Tag value and it conflicts with an existing Item Tag name, a dialog will pop up and request a change.

Base Address or Offset

The Base Address or Base Offset must be defined for the artificial variable being defined by this dialog.

For artificial variables associated with program resources, the Base Address should be set to the virtual address you wish to probe within the program.

For artificial variables associated with other resources, the Base Offset should be set to the offset you wish to view within the memory region of the resource. This value added to the Base Offset that was supplied in the associated resource selection window to calculate the final offset of the item in the resource.

Output Format

The Output Format can be selected via the drop-down list. The default setting is Default, which will cause variable values to be displayed in their natural format as described in the following table:

Type Class	Format
Signed Integers	Decimal
Unsigned Integers	Hexadecimal
Real	Exponential
Fixed Point	Exponential
Enumerations	Enumeration Image
Character Arrays	String
Pointers	Hexadecimal

Alternatively, you can select a display format from the list.

Define Array

When the Define Array checkbox is selected, the artificial item will be defined as an array of the type that you select in the Interactive Type Browser (see “Interactive Type Browser” on page 12-4).

Lower Bound

Specifies the lower bound of the defined array. The value can be set either by entering the number in the text field or by using the increase/decrease arrows.

Upper Bound

Specifies the upper bound of the defined array. The value can be set either by entering the number in the text field or by using the increase/decrease arrows.

The bottom portion of the Item Definition window consists of the Interactive Type Browser (see "Interactive Type Browser" on page 12-4) which provides expandable and collapsible lists of entities organized in a tree containing scopes, types, and type components.

Interactive Type Browser

The Interactive Type Browser appears in the bottom portion of the Item Definition Window and provides expandable and collapsible lists of entities organized in a tree containing scopes, types, and type components.

For each node in the tree that has children, the list can be expanded or collapsed by clicking the control box to the left of the icon.

Keyboard traversal is also supported within the tree:

Up Arrow

Pressing the Up Arrow key will cause the node immediately above the current node in a list of siblings to be selected unless the top node in a list has been reached. If the top node has been reached, the key has no effect.

Down Arrow

Pressing the Down Arrow key will cause the node immediately below the current node in a list of siblings to be selected unless the bottom node in a list has been reached. If the bottom node has been reached, the key has no effect.

Left Arrow

Pressing the Left Arrow key will cause the parent of the current list to be selected. When a root node is reached, the key has no effect.

Right Arrow

Pressing the Right Arrow key will cause automatically expand the current node and cause the first child node to become selected. If the current node is a leaf, the key has no effect.

Space

Pressing the Space bar toggles the expansion setting of the current node. If the current node has no children, the key has no effect.

For each resource node, there are four main nodes which provide lists of Functions, Globals, and Files, and Packages which contain scopes and types.

Functions

The Functions list is populated with the names identifiable functions within the symbol file. Functions that contain scopes or types that can be recorded will have an expandable control box to their left.

Globals

The Globals list is populated with basic numeric and character types.

Files

The Files list is populated with all identifiable source files within the symbol file. Files may contain scopes, such as packages or functions, and types declared outside of functions.

Packages

The Packages list is populated with all identifiable library level Ada packages within the symbol file. Packages contain other packages or types.

Types

Types appear by name with boxes as their icon. If a type is a composite type (an array or structure or record), it will appear with an expandable control box. All record and structure components are shown when a record or structure is expanded. Array components can also be expanded, but are limited by the resource `Nprobe.text.maxArrayExpansion`, which defaults to 1000.

For resources that do not have symbol files associated with them, only the Globals list will be populated with types.

For resources that have symbol files, types will appear in the tree only if the associated source files were compiled with the symbolic debug option (**`-g`**). Further, depending on the compiler version, types that were not applied to a variable in such source files may be omitted from the tree.

Each type node contains the type name and a description of the type, including any offset from its composite parent, its bit size, bit offset, and type class (e.g. array, structure, integer, or record).

To select a type for the new variable being defined in this dialog, select the desired type node.

A composite type may be added as a whole, or individual components may be added.

If a component of a composite type is added, the offset of the component within the base composite type will be added to the Base Offset value when the new variable is added to the list of Probe Items. This association is maintained in the session configuration..

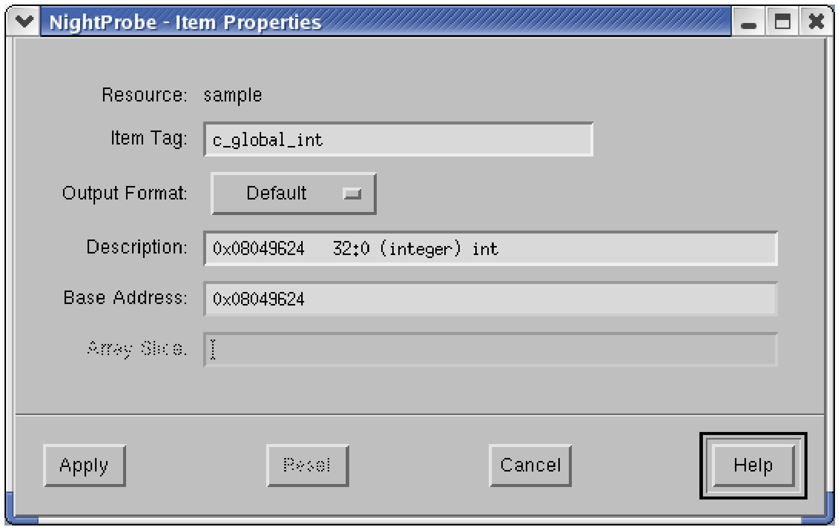
Thus, if a saved configuration is reloaded and the component's offset had changed due to a modified symbol file, the component offset is adjusted accordingly.

13
Item Properties Window

The Item Properties window allows you view and modify some attributes of variables listed in the Probe Items of the NightProbe main window.

It is launched by selecting the Properties... menu item from the pop-up menu for the probe item of interest listed under Probe Items (see “Probe Items” on page 3-18) in the Session Overview Area of the NightProbe main window.

The Item Properties window is shown in Figure 13-1:



Resource: sample
Item Tag: c_global_int
Description: 0x08049624 32:0 (integer) int
Base Address: 0x08049624

Figure 13-1. Item Properties Window

Item Tag

The Item Tag, a short-hand identifier for the item, is displayed here.

Output Format

The setting for the default Output Format is displayed in the drop-down list.

This field can be modified to affect how values of this item are displayed.

The Default setting will cause item value to be displayed in its natural format as described in the following table:

Type Class	Format
Signed Integers	Decimal
Unsigned Integers	Hexadecimal
Real	Exponential
Fixed Point	Exponential
Enumerations	Enumeration Image
Character Arrays	String
Pointers	Hexadecimal

Description

The Description of the item includes:

Item Address or Offset

The first field in the Description is the virtual address of a real program variable or the offset of an artificial variable.

For artificial variables which were defined as components of a composite type, the value displayed here may differ from the value displayed in the Base Offset field. For such artificial variables, the value displayed here is the sum of the Base Offset and any offset associated with this component within the outermost enclosing composite type.

Bit Size and Bit Offset

The second field in the Description is the Bit Size and Bit Offset, displayed together, separated by a colon.

Type Class

The third field in the Description is the Type Class, displayed in parentheses. Type classes include:

- Enumeration
- Floating Point
- Fixed Point
- Integer
- Record
- Array
- Character

- Pointer
- (Fortran) Complex

Type Name

The last field in the Description is the Type Name, or short-hand type declaration if no specific type name is available.

Base Address or Offset

The Base Address of program items or the Base Offset of artificial items is shown here.

This value may be modified only for artificial variables created with the Item Definition window.

The Base Offset may differ from the offset displayed in the Description field. This occurs only for artificial items which were defined as components of a composite type. The final offset of the item is defined by the Base Offset plus the component's offset within the outermost containing composite type.

Array Slice

If the item is an array, the current slice information is shown and may be modified.

If a slice is specified, only those array components are sampled.

See "Array Slices" on page C-2 for more information on array slice syntax.

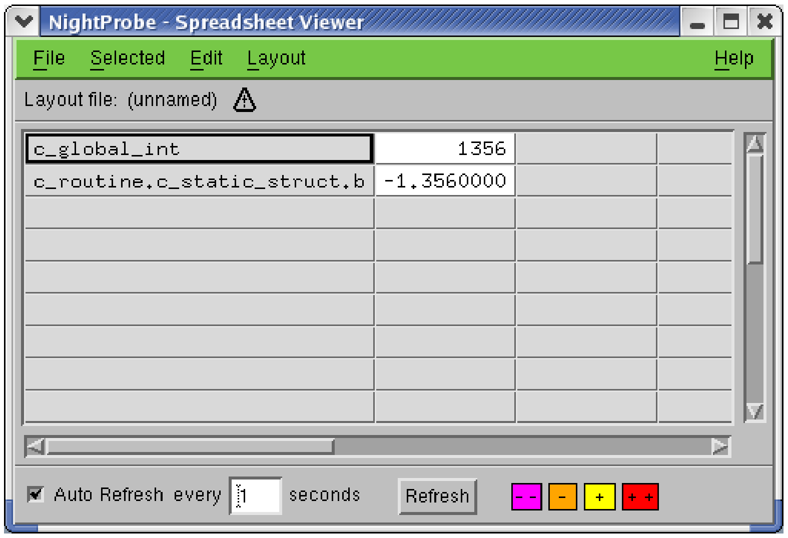
14
Spreadsheet Viewer

The Spreadsheet Viewer window provides for both viewing and modifying sampled data.

It is launched by selecting the Spreadsheet Output menu item from the Outputs icon menu (see “Outputs” on page 3-14) or from the Output menu (see “Output” on page 3-4) on the NightProbe main window.

The Spreadsheet Viewer window is shown in Figure 14-1:

.



c_global_int	1356		
c_routine.c_static_struct.b	-1.3560000		

Figure 14-1. Spreadsheet Viewer window

The Spreadsheet Viewer window:

- 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 E-5 for an example using a C++ and C and Ada program.

The Spreadsheet Viewer window consists of the following components:

- Menu Bar (see "Menu Bar" on page 14-2)
- Layout Configuration Status Area (see "Layout Configuration Status Area" on page 14-11)
- Spreadsheet Viewing Area (see "Spreadsheet Viewing Area" on page 14-12)
- Control Area (see "Control Area" on page 14-12)

Menu Bar

The Spreadsheet Viewer window menu bar contains the following menus.

- File (see "File" on page 14-2)
- Selected (see "Selected" on page 14-4)
- Edit (see "Edit" on page 14-8)
- Layout (see "Layout" on page 14-9)
- Help (see "Help" on page 14-11)

Each menu is described in the sections that follow.

File

Mnemonic: Alt+F

The File menu on the Spreadsheet Viewer window contains items for creating new layout files, opening existing layout files, and saving the current layout file.

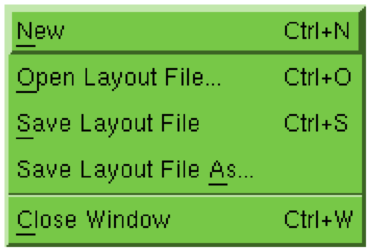


Figure 14-2. File menu

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

New

Mnemonic: N
Accelerator: Ctrl+N

This option allows you to clear the cells in the spreadsheet and the layout configuration.

This item is desensitized when you are connected to the program being monitored.

Open Layout File...

Mnemonic: O
Accelerator: Ctrl+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 from which to choose a filename.

This item is desensitized when you are connected to the program being monitored.

Save Layout File

Mnemonic: S
Accelerator: Ctrl+S

This option allows you to save the spreadsheet layout configuration to the current layout file. If the spreadsheet viewer is not currently associated with a layout configuration file name, this option is the same as Save Layout File As.

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 14-5).

Close Window

Mnemonic: C
Accelerator: Ctrl+W

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

This item is desensitized when you are connected to the program being monitored.

Selected

Mnemonic: Alt+S

The Selected menu on the Spreadsheet Viewer window contains items for placing varibles in the spreadsheet, changing the attributes of cells in the spreadsheet, enabling/disabling of updates, and the alignment of the content with the cells.

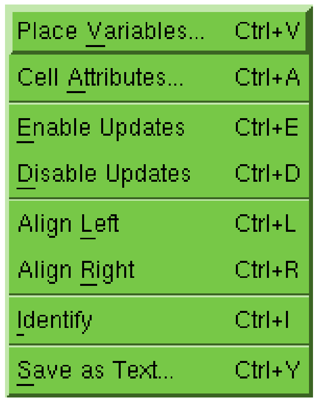


Figure 14-3. Selected menu

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: Ctrl+V

Selecting the Place Variables menu option displays the Spreadsheet Variables window (see "Spreadsheet Variables" on page 14-6) which contains controls to place variable cells onto a spreadsheet.

Cell Attributes

Mnemonic: A
Accelerator: Ctrl+A

Selecting the Cell Attributes menu option displays the Cell Attributes dialog (see "Cell Attributes" on page 14-7) which allows you to view and change various attributes associated with a spreadsheet cell.

NOTE

The window is only active when a Variable cell is selected.

Enable Updates

Mnemonic: E
Accelerator: Ctrl+E

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

Disable Updates

Mnemonic: D
Accelerator: Ctrl+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: Ctrl+L

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

Align Right

Mnemonic: R
Accelerator: Ctrl+R

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

Identify

Mnemonic: I
Accelerator: Ctrl+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: Ctrl+Y

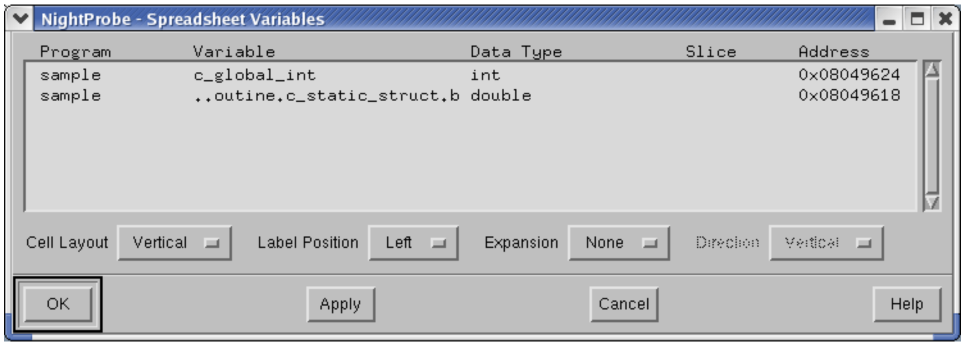
Writes as text information to a file the image of the currently selected cells. You will be presented with a file selection dialog with which to choose a file name.

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 14-2).

Spreadsheet Variables

The Spreadsheet Variables dialog is presented when the user selects the Place Variables... menu item from the Selected menu of the Spreadsheet Viewer window (see "Spreadsheet Viewer" on page 14-1).

The Spreadsheet Variables dialog is shown in Figure 14-4:



Program	Variable	Data Type	Slice	Address
sample	c_global_int	int		0x08049624
sample	..outine.c_static_struct.b	double		0x08049618

Cell Layout: Vertical · Label Position: Left · Expansion: None · Direction: Vertical

Figure 14-4. Spreadsheet Variables dialog

To use this dialog, 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 list of variables are four option menus for controlling placement:

Cell Layout

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).

Label Position

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).

Expansion

The Expansion menu specifies what to do with variables that represent composite types. Selecting None will place all array elements in a single cell. Selecting Records will automatically expand the record allocating a cell for each individual component. Selecting Arrays will automatically expand array components as individual cells. Selecting Both will expand arrays and records. Expansion is limited to a single level; components which are themselves composites will not be expanded.

Direction

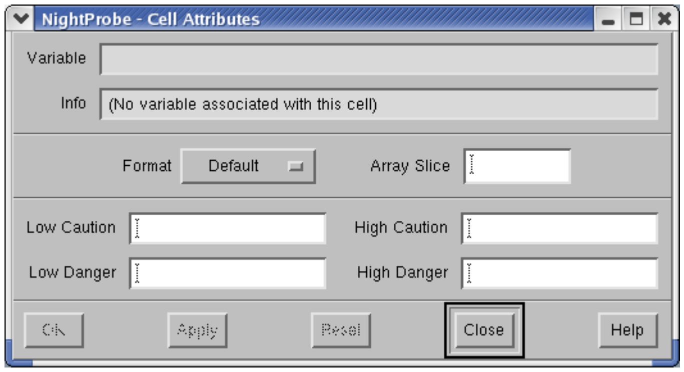
The Direction menu is sensitized when composite expansion is selected. Selecting Horizontal or Vertical will place each composite in its own cell, laid out in the specified direction.

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

The Cell Attributes dialog is presented when the user selects the Cell Attribues... menu item from the Selected menu of the Spreadsheet Viewer window (see “Spreadsheet Viewer” on page 14-1).

The Cell Attributes dialog is shown in Figure 14-5:



(No variable associated with this cell)

Figure 14-5. Cell Attributes dialog

Variable
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 C-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 F-2.

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).

Edit

Mnemonic: Alt+E

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

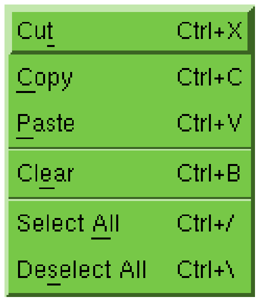


Figure 14-6. Edit menu

Cut

Mnemonic: T
Accelerator: Ctrl+X

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

This item is desensitized when you are connected to the program being monitored.

Copy

Mnemonic: C
Accelerator: Ctrl+C

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

This item is desensitized when you are connected to the program being monitored.

Paste

Mnemonic: P
Accelerator: Ctrl+V

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

This item is desensitized when you are connected to the program being monitored.

Clear

Mnemonic: E
Accelerator: Ctrl+B

Clears the selected cells.

This item is desensitized when you are connected to the program being monitored.

Select All

Mnemonic: A
Accelerator: Ctrl+/

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

Deselect All

Mnemonic: S
Accelerator: Ctrl+\

Puts all cells into the “unselected” state.

Layout

Mnemonic: Alt+L

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

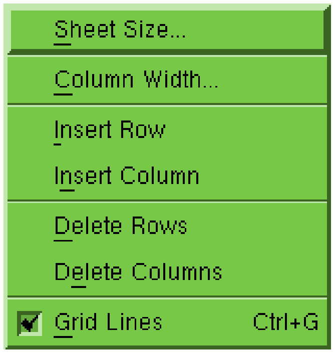


Figure 14-7. Layout menu

Sheet Size

Mnemonic: S

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

This item is desensitized when you are connected to the program being monitored.

Column Width

Mnemonic: C

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

NOTE

Cell width is limited to 255 characters.

Insert Row

Mnemonic: I

Inserts one row above the current selection point.

This item is desensitized when you are connected to the program being monitored.

Insert Column

Mnemonic: N

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

This item is desensitized when you are connected to the program being monitored.

Delete Rows

Mnemonic: D

Deletes the selected rows.

This item is desensitized when you are connected to the program being monitored.

Delete Columns

Mnemonic: E

Deletes the selected columns.

This item is desensitized when you are connected to the program being monitored.

Grid Lines

Mnemonic: G

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

Help

Mnemonic: Alt+H

The Help menu operates exactly like the menu provided in the NightProbe main 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 2-3 for more information.

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.

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 “Control Area” on page 14-12).

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 target system, 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, enumeration constants as identifiers, or character strings when enclosed in double quotes (").

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. When using the On Demand timing source (see “On Demand” on page 3-13), sample data is displayed each time a new sample is recorded. Because intermediate refreshes of the display are not necessary when using On Demand timing, the Control Area is unavailable while connected. The Control Area will remain available using any other timing source. In addition, the Control Area contains a legend indicating the two caution and two danger colors (see “Cell Color Legend” on page 14-13).

Auto Refresh

The Auto Refresh 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 2-1 and “NightStar Resources” on page F-2.)

Refresh

The Refresh button gets the most recent sample taken and displays it in the Spreadsheet Viewer window. The Refresh 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 dialog (see “Cell Attributes” on page 14-4).

You can define the colors of these cells with resources described in “NightStar Resources” on page F-2.

15 NightProbe API

NightProbe provides two sets of APIs for use in applications. The NightProbe Datastream API (see "NightProbe Datastream API" on page 15-1) provides a basic interface to the data produced by NightProbe. These data structures and functions allow the user to process the data sampled by NightProbe either in real-time or via a previously recorded file.

The NightProbe Trigger API (see "NightProbe Trigger API" on page 15-20) provides an interface to the NightProbe Trigger Server Queue allowing an application to control the sampling of data by NightProbe in a synchronized manner.

NightProbe Datastream API

The NightProbe Datastream 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 `File Output` and `Program Output` methods (see "File Output" on page 6-1 and "Program Output" on page 6-14).

The following sections describe the general format of the data generated by NightProbe sampling (see "NightProbe Data Format" on page 15-1) as well as the data structures and functions (see "Data Structures" on page 15-2 and "Functions" on page 15-6) that comprise the NightProbe Datastream API.

Sample programs using the NightProbe Datastream API are also provided (see "Sample Programs" on page 15-13).

NightProbe Data Format

This section describes the general format of data generated by NightProbe sampling. This format is used when you select either the `File Output` or `Program Output` method (see "File Output" on page 6-1 and "Program Output" on page 6-14).

The NightProbe Datastream API allows you 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*.

When the `File Output` method is selected (see "File Output" on page 6-1), NightProbe writes the data to a file, and a user program opens that file and passes the file descriptor to the NightProbe Datastream API calls to decode the data.

When the `Program Output` method is selected (see "Program Output" on page 6-14), 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 passes the **`stdin`** file descriptor to the NightProbe Datastream API calls to decode the data.

The following diagram describes the general layout of a datastream:

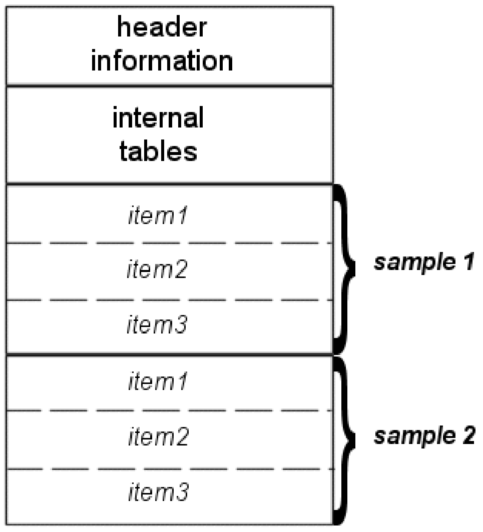


Figure 15-1. Structure of NightProbe datastream

The NightProbe Datastream API functions 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 samples. See "Functions" on page 15-6 for more information about these functions.

Data Structures

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

- `np_endian_type` (see "np_endian_type" on page 15-3)
- `np_handle` (see "np_handle" on page 15-3)
- `np_header` (see "np_header" on page 15-3)
- `np_item` (see "np_item" on page 15-4)
- `np_process` (see "np_process" on page 15-4)
- `np_type` (see "np_type" on page 15-5)

See "Functions" on page 15-6 for information about the functions available in the NightProbe Datastream API.

np_endian_type

`np_endian_type` is used to represent the endian order of the NightProbe data and of the host system.

```
typedef enum np_endian_type_code {
   NP_LITTLE_ENDIAN,      /* Addresses designate the Least
                             Significant Byte of a value.  */
   NP_BIG_ENDIAN,         /* Addresses designate the Most
                             Significant Byte of a value.  */
} np_endian_type ;
```

See “Data Structures” on page 15-2 for other data structures included in the NightProbe Datastream API.

np_handle

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

```
typedef int np_handle;
```

See “Data Structures” on page 15-2 for other data structures included in the NightProbe Datastream 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_endian_type   sample_endian;
   np_process     * processes;
   np_item        * items;
} np_header ;
```

See “Data Structures” on page 15-2 for other data structures included in the NightProbe Datastream API.

SEE ALSO

- “np_endian_type” on page 15-3
- “np_process” on page 15-4
- “np_item” on page 15-4
- “np_read()” on page 15-8

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
   unsigned      bit_offset; // bit offset within each sample
   unsigned      bit_size;   // atomic size in bits
   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 `bit_size` bits beginning at `bit_offset` within the sample.

See “Data Structures” on page 15-2 for other data structures included in the NightProbe Datastream API.

SEE ALSO

- “np_process” on page 15-4
- “np_type” on page 15-5

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 15-2 for other data structures included in the NightProbe Datastream API.

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 15-2 for other data structures included in the NightProbe Datastream API.

SEE ALSO

- “np_item” on page 15-4

Functions

The following functions are part of the NightProbe Datastream API:

- `np_open()` (see "np_open()" on page 15-6)
- `np_avail()` (see "np_avail()" on page 15-7)
- `np_read()` (see "np_read()" on page 15-8)
- `np_close()` (see "np_close()" on page 15-9)
- `np_format()` (see "np_format()" on page 15-10)
- `np_error()` (see "np_error()" on page 15-11)

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 `File Output` output method (see "File Output" on page 6-1) which contains the data recording output If data recording output is streamed directly from NightProbe using the `Program Output` output method (see "Program Output" on page 6-14), *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 diagnostic message describing the failure

IMPORTANT

> If you call `np_open()` on a data file that was produced on a target architecture having a different endian order from the host (e.g. big-endian data file/little-endian host, or vice versa), all data returned to you by `np_read()` will be of the data file's orientation. In order to obtain meaningful information about the probe samples in the file in such situations, you must first convert the format of the data in the sample buffer to the proper endian format for the host before calling `np_format()`.

See “Functions” on page 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- “np_header” on page 15-3
- “np_handle” on page 15-3
- “np_read()” on page 15-8
- “np_error()” on page 15-11

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 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- "np_open()" on page 15-6
- "np_read()" on page 15-8
- "np_error()" on page 15-11

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 15-13 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.

IMPORTANT

If you call `np_open()` on a data file that was produced on a target architecture having a different endian order from the host (e.g. big-endian data file/little-endian host, or vice versa), all data returned to you by `np_read()` will be of the data file's orientation. In order to obtain meaningful information about the probe samples in the file in such situations, you must first convert the format of the data in the sample buffer to the proper endian format for the host before calling `np_format()`.

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 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- “np_header” on page 15-3
- “np_item” on page 15-4
- “np_open()” on page 15-6
- “np_avail()” on page 15-7
- “np_error()” on page 15-11
- “np_format()” on page 15-10

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 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- “np_open()” on page 15-6
- “np_error()” on page 15-11

np_format()

Return an allocated string representation of the specified `np_item` value from the given sample. The caller is responsible for freeing the memory associated with the returned string once it is no longer needed.

SYNTAX

```
char * np_format (np_handle handle,
                  np_item * i,
                  void * sample,
                  int which);
```

PARAMETERS

handle	value (obtained from `np_open()`) which identifies the NightProbe datastream of interest
i	a pointer to an `np_item` descriptor denoting a single item within a data sample. The `np_item` is part of the `np_header` obtained from the previous call to `np_open()`.
sample	a pointer to the contents of a single sample obtained from a call to `np_read()`
which	for items with multiple atoms (i.e. *i->count* > 1), *which* determines the atom to be formatted. A *which* value of 1 indicates the first atom for the item.

RETURN VALUES

non-NULL	value represents a textual representation of the specified data in a format based on the `np_type` of the item
NULL	a parameter was invalid, or the NightProbe Datastream API was unable to allocate memory for the result. np_error may be called to obtain a diagnostic message describing the failure.

See “Functions” on page 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- “np_header” on page 15-3
- “np_item” on page 15-4
- “np_error()” on page 15-11

np_host_endian()

Returns the `np_endian_type` value denoting the endian order of the host system.

SYNTAX

```
np_endian_type np_host_endian (void);
```

See “Functions” on page 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- “np_endian_type” on page 15-3

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 15-6 for other functions included in the NightProbe Datastream API.

SEE ALSO

- “np_open()” on page 15-6
- “np_avail()” on page 15-7
- “np_read()” on page 15-8

Sample Programs

The following programs are given as examples of how to use the NightProbe Datastream API (see “NightProbe Datastream API” on page 15-1).

program_output_test.c

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

program_output_fbs_test.c

This program uses the NightProbe Datastream 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.

program_output_test.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <nprobe.h>



int    cycles   = 0 ;
int    overruns = 0 ;
char * sample;


// Perform the work of consuming a single Data Recording sample from NightProbe.
//
int
work (FILE * ofile, np_handle h, np_header * hdr) {
   np_item * i;
   int status;
   int which ;

   // Read one sample, which may contain data for multiple processes
   // and variables.
   //
   status = np_read (h, sample);
   if (status <= 0)  {
      return status;
   }

   cycles++ ;

   fprintf (ofile, "Sample %d\n", cycles);
   for (i = hdr->items; i; i = i->link) {
      char buffer [1024] ;
      sprintf (buffer, "item: %s:", i->name) ;
      fprintf (ofile, "%-30s", buffer) ;        // Nice formatting :-)

      // Display the value of each item.
      // For arrays, format each individual item.
      //
      for (which = 1; which <= i->count; ++which) {
         char *   image = np_format (h, i, sample, which) ;

         if (image != NULL) {
            fprintf (ofile, " %s", image) ;
         } else {
            fprintf (ofile, "\n<error: %s>\n", np_error (h)) ;
            return -1 ;
         }

         free (image) ;
      }
      fprintf (ofile, "\n");
   }
   fflush (ofile) ;

   return 1 ;
}




int
main (int argc, char *argv[])
{
   np_handle h;
   np_header hdr;
   np_process * p;
   np_item * i;
   int fd;
   int status;
   FILE *   ofile = stdout ;


   fd = 0 ; // stdin

   status = np_open (fd, &hdr, &h);
   if (status) {
      fprintf (stderr, "%s\n", np_error(h));
      exit(1);
   }

   sample = (char *) malloc(hdr.sample_size);
   if (sample == NULL) {
      fprintf (stderr, "insufficient memory to allocate sample buffer\n");
      exit(1);
   }

   for (p = hdr.processes; p; p = p->link) {
      if (p->pid >= 0) {
         fprintf (ofile, "process: %s (%d)\n", p->name, p->pid);
      } else {
         fprintf (ofile, "resource: %s (%s)\n", p->name, p->label);
      }
   }
   fprintf (ofile, "\n");

   for (i = hdr.items; i; i = i->link) {
      fprintf (ofile, "item: %s (%s), size=%d bits, count=%d, type=%d\n",
              i->name, i->process->name, i->bit_size, i->count, i->type);
   }
   fprintf (ofile, "\n");

   for (;;) {
      status = work (ofile, h, &hdr)  ;
      if (status <= 0) break ;
   }

   fprintf (ofile, "Data Recording done: %d cycles fired, %d overruns\n",
            cycles, overruns) ;

   if (ofile != stdout) {
      fclose (ofile) ;
   }

   if (status < 0) {
      fprintf (stderr, "%s\n", np_error(h));
   }

   np_close (h);

   // At this point, file descriptor 0 remains open, but is no
   // longer a NightProbe Data File/Stream.
}
```

program_output_fbs_test.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <nprobe.h>
#ifdef linux
#include <fbsched.h>
#else
#include <fbslib.h>
#endif



int    cycles   = 0 ;
int    overruns = 0 ;
char * sample;


// Perform the work of consuming a single Data Recording sample from NightProbe.
//
// This function is called once every time the fbswait() system call returns
// successfully.
//
int
work (FILE * ofile, np_handle h, np_header * hdr) {
   np_item * i;
   int status;
   int n;
   char * ptr;


   // 0, 1, or >1 trigger events may have occurred since we last work()ed.
   //
   // Check whether data is available, and process it as long as new
   // data is already available within this work cycle.
   //
   // A more sophisticated program would limit the number of np_read() calls
   // per work cycle based upon how much time is left in the current cycle.
   //
   while (np_avail (h)) {

      // Read one sample, which may contain data for multiple processes
      // and variables.
      //
      status = np_read (h, sample);
      if (status <= 0)  {
         return status;
      }

      cycles++;

      fprintf (ofile, "\n");
      for (i = hdr->items; i; i = i->link) {
         fprintf (ofile, "item: %25s :", i->name);
         // Calculate the address of the item within the sample buffer.
         // This formula calculates the address of the first byte of
         // data corresponding to the item.
         //
         // The first bit is at i->bit_offset % 8 within that byte.
         //
         ptr = sample + (i->bit_offset/8);

         for (n = 0; n < i->count; ++n) {

            // Note that this simple example assumes type/format from
            // the size of the data item.  The 'i->type' field should
            // be taken into account for a more accurate means of
            // determining the data format.
            //
            if (i->bit_offset % 8) {
               fprintf (ofile, " <size=%d bits, offset=%d bits>",
                                i->bit_size, i->bit_offset % 8) ;
            } else {
               switch (i->bit_size) {
               case 8:
                  fprintf (ofile, " 0x%1x", ((char*)ptr)[n]);
                  break;
               case 16:
                  fprintf (ofile, " 0x%1x", ((unsigned short*)ptr)[n]);
                  break;
               case 32:
                  fprintf (ofile, " 0x%1x", ((unsigned*)ptr)[n]);
                  break;
               case 64:
                  fprintf (ofile, " %lf", ((double*)ptr)[n]);
                  break;
               default:
                  fprintf (ofile, " <size=%d bits>", i->bit_size) ;
               }
            }
         }
         fprintf (ofile, "\n");
      }
      fflush (ofile);
   }

   return 1;
}




int
main (int argc, char *argv[])
{
   np_handle h;
   np_header hdr;
   np_process * p;
   np_item * i;
   int fd;
   int status;
   FILE *   ofile = stdout ;
```

```
#ifdef linux
   if (!fbsavail()) {
      fprintf (ofile, "fbsavail() reports No FBS on this target\n") ;
      fclose (ofile) ;
      exit (1) ;
   }
#endif



   fd = 0 ; // stdin

   status = np_open (fd, &hdr, &h);
   if (status) {
      fprintf (stderr, "%s\n", np_error(h));
      exit(1);
   }

   sample = (char *) malloc(hdr.sample_size);
   if (sample == NULL) {
      fprintf (stderr, "insufficient memory to allocate sample buffer\n");
      exit(1);
   }

   for (p = hdr.processes; p; p = p->link) {
      if (p->pid >= 0) {
         fprintf (ofile, "process: %s (%d)\n", p->name, p->pid);
      } else {
         fprintf (ofile, "resource: %s (%s)\n", p->name, p->label);
      }
   }
   fprintf (ofile, "\n");

   for (i = hdr.items; i; i = i->link) {
      fprintf (ofile, "item: %s (%s), size=%d bits, count=%d, type=%d\n",
               i->name, i->process->name, i->bit_size, i->count, i->type);
   }
   fprintf (ofile, "\n");

   for (;;) {

      // We wait till the Concurrent FBS wakes us up at the time which is
      // appropriate for performing data recording.  This program must be
      // scheduled on the FBS, but doing so allows the scheduling of data
      // recording activity at a time that won't disturb other critical
      // application cycles.
      //
      int stat = fbswait() ;

      // Diagnose the return value from fbswait()
      if (stat < 0) {
         switch (stat) {
         case -1:
            if (errno == ENOENT) {
               fprintf (ofile,
                        "%s has been removed from the scheduler\n", argv[0]) ;
            } else {
               fprintf (ofile, "fbs_wait(3) failed on cycle %d: "
                              "errno is %d (%s)\n",
                         cycles, errno, strerror (errno)) ;
            }
            break ;
         default:
            fprintf (ofile, "fbs_wait(3) returned unexpected %d on cycle %d\n",
                     stat, cycles) ;
            break ;
         }

         break ;
      }

      switch (stat) {
      case 0:
         break ;
      case 1:
         fprintf (ofile, "fbstrig(2) caused sim to fire: cycle %d\n", cycles) ;
         break ;
      case 2:
         fprintf (ofile, "soft overrun %d detected on cycle %d\n",
                  ++overruns, cycles) ;
         break ;
      }

      status = work (ofile, h, &hdr);
      if (status <= 0) {
         break;
      }
   }

   fprintf (ofile, "Data Recording done: %d cycles fired, %d overruns\n",
            cycles, overruns) ;

   if (ofile != stdout) {
      fclose (ofile) ;
   }

   if (status < 0) {
      fprintf (stderr, "%s\n", np_error(h));
   }

   np_close (h);

   // At this point, file descriptor 0 remains open, but is no
   // longer a NightProbe Data File/Stream.
}
```

NightProbe Trigger API

The NightProbe Trigger API provides an interface to the NightProbe Trigger Server Queue allowing an application (the NightProbe Trigger Client) to cause the NightProbe Server to sample data in a synchronized manner.

The name of the Trigger Server Queue can be specified to NightProbe using the `Set Trigger Timer` dialog (see "Set Trigger Timer" on page 5-6).

The following sections describe the data structures and functions (see "Data Structures" on page 15-20 and "Functions" on page 15-21) that comprise the NightProbe Trigger API.

A sample program using the NightProbe Trigger API is also provided (see "Sample Program" on page 15-25).

Data Structures

The following data structure is part of the NightProbe Trigger Application Programming Interface:

- `np_trigger_handle` (see "np_trigger_handle" on page 15-20)

See "Functions" on page 15-21 for a list of functions included in the NightProbe Trigger API.

np_trigger_handle

`np_trigger_handle` is a unique integer value denoting a connection to a NightProbe Trigger Server Queue. The `np_trigger_handle` may be used to request sampling events and/or obtain further information about a failure.

```
typedef int np_trigger_handle;
```

Functions

The following functions are part of the NightProbe Trigger API:

- `np_trigger_open()` (see “np_trigger_open()” on page 15-21)
- `np_trigger()` (see “np_trigger()” on page 15-22)
- `np_trigger_close()` (see “np_trigger_close()” on page 15-23)
- `np_trigger_error()` (see “np_trigger_error()” on page 15-23)

np_trigger_open()

`np_trigger_open()` is used to open a connection to a NightProbe Trigger Server Queue, allowing the caller to control when NightProbe samples are captured.

SYNTAX

```
int np_trigger_open (char *name, np_trigger_handle *sampler);
```

PARAMETERS

name	a unique identifier for the NightProbe Trigger Server Queue on the target system. This is the same name assigned to the trigger in the `Set Trigger Timer` dialog (see “Set Trigger Timer” on page 5-6) *name* must be a legal filename containing no '`/`' characters. The length of name is restricted to at most (`MAXNAMELEN`-15).
sampler	returns an `np_trigger_handle` designating the active NightProbe Trigger Sampler Queue connection.

RETURN VALUES

0	indicates successful completion
-1	indicates a failure `np_trigger_error()` may be used to obtain further information about the reason for the failure.

See “Functions” on page 15-21 for other functions included in the NightProbe Trigger API.

SEE ALSO

- “np_trigger_handle” on page 15-20
- “np_trigger_error()” on page 15-23

np_trigger()

`np_trigger()` requests a sample be taken by the NightProbe sampler. If there is an active NightProbe Trigger Server connected to the queue, `np_trigger()` sends a trigger request to the NightProbe Trigger Server Queue and returns.

SYNTAX

```
int np_trigger (np_trigger_handle sampler);
```

PARAMETERS

sampler	an `np_trigger_handle` specifying the active NightProbe Trigger Sampler Queue connection

RETURN VALUES

0	indicates successful completion
1	no NightProbe Trigger Server was connected. This is not necessarily an error, but the result is provided so that the NightProbe Trigger API Client can determine when a server disconnects. Note that the server may later re-connect, and subsequent `np_trigger()` calls will again return 0.
-1	indicates a failure `np_trigger_error()` may be used to obtain further information about the reason for the failure.

See “Functions” on page 15-21 for other functions included in the NightProbe Trigger API.

SEE ALSO

- “np_trigger_handle” on page 15-20
- “np_trigger_open()” on page 15-21
- “np_trigger_error()” on page 15-23

np_trigger_close()

`np_trigger_close()` is used to disconnect from the NightProbe Trigger Server Queue.

SYNTAX

```
int np_trigger_close (np_trigger_handle sampler);
```

PARAMETERS

sampler	an `np_trigger_handle` specifying the active NightProbe Trigger Sampler Queue connection to be closed *sampler* is no longer valid after this call.

RETURN VALUES

0	indicates successful completion
-1	indicates a failure `np_trigger_error()` may be used to obtain further information about the reason for the failure.

See “Functions” on page 15-21 for other functions included in the NightProbe Trigger API.

SEE ALSO

- “np_trigger_handle” on page 15-20
- “np_trigger_open()” on page 15-21
- “np_trigger_error()” on page 15-23

np_trigger_error()

`np_trigger_error()` returns an error message describing the most recent failure detected by the NightProbe Trigger API functions.

SYNTAX

```
char *np_trigger_error (np_trigger_handle sampler);
```

PARAMETERS

sampler	an `np_trigger_handle` specifying the active NightProbe Trigger Sampler Queue connection

RETURN VALUES

Returns the error message describing the most recent failure detected by the NightProbe Trigger API functions.

Returns ‘`No error`’ if no errors have occurred.

See “Functions” on page 15-21 for other functions included in the NightProbe Trigger API.

SEE ALSO

- “np_trigger_handle” on page 15-20
- “np_trigger_open()” on page 15-21

Sample Program

The following program is given as an example of how to use the NightProbe Trigger API (see “NightProbe Trigger API” on page 15-20).

nprobe_trigger_test.c

This program uses the NightProbe Trigger API.

nprobe_trigger_test.c

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

#include "nprobe_trigger.h"

int
main (int argc, char * argv[])
{
   np_trigger_handle h;
   int status;
   int i ;

   if (argc != 2) {
      fprintf (stderr, "Usage: nprobe_trigger_test trigger_name\n");
      exit(1);
   }

   printf ("Trigger %s: connecting...\n", argv[1]);
   status = np_trigger_open (argv[1], &h);
   if (status) {
      fprintf (stderr, "%s\n", np_trigger_error(h));
      exit(1);
   }
   printf ("Trigger %s: conected to trigger server queue\n", argv[1]);

   for (i = 0 ; i >= 0 ; i++) {
      sleep (1) ;
      if ((status = np_trigger (h)) < 0) {
         fprintf (stderr, "%s\n", np_trigger_error(h));
         exit(1);
      } else if (status == 0) {
         printf ("Trigger %s: triggered %d\n", argv[1], i);
      } else {
         printf ("Trigger %s: dropped %d\n", argv[1], i);
      }
   }

   np_trigger_close (h);
   printf ("Trigger %s: closed.\n", argv[1]);

   exit (0);
}
```

A
NightStar Licensing

NightStar RT uses the NightStar License Manager (NSLM) to control access to the NightStar RT tools.

License installation requires a licence key provided by Concurrent.The NightStar RT tools request a licence (see “License Requests” on page A-2) from a license server (see “License Server” on page A-2).

Two license modes are available, fixed and floating, depending on which product option you purchased. Fixed licenses can only be served to NightStar RT users from the local system. Floating licenses may be served to any NightStar RT user on any system on a network.

Tools are licensed per system, per concurrent user. A single license is shared among any or all of the NightStar RT tools for a particular user on a particular system. The intent is to allow *n* developers to fully utilize all the tools at the same time while only requiring *n* licenses. When operating the tools in remote mode, where a tool is launched on a local system but is interacting with a remote system, licenses are required only from the host system.

You can obtain a license report which lists all licenses installed on the local system, current usage, and expiration date for demo licenses (see “License Reports” on page A-3).

The default configuration includes a strict firewall which interferes with floating licenses. See “Firewall Configuration for Floating Licenses” on page A-3 for information on handling such configurations.

See “License Support” on page A-4 for information on contacting Concurrent for additional assistance with licensing issues.

License Keys

Licenses are granted to specific systems to be served to either local or remote clients, depending on the license model, fixed or floating.

License installation requires a license key provided by Concurrent. To obtain a license key, you must provide your system identification code. The system identification code is generated by the **nslm_admin** utility:

```
nslm_admin --code
```

System identification codes are dependent on system configurations. Reinstalling Linux on a system or replacing network devices may require you to obtain new license keys.

To obtain a license key, use the following URL:

http://www.ccur.com/NightStarRTKeys

Provide the requested information, including the system identification code. Your license key will be immediately emailed to you.

Install the license key using the following command:

```
nslm_admin --install=xxxx-xxxx-xxxx-xxxx-xxxx
```

where *xxxx-xxxx-xxxx-xxxx-xxxx* is the key included in the license acknowledgment email.

License Requests

By default, the NightStar RT tools request a license from the local system. If no licenses are available, they broadcast a license request on the local subnet associated with the system’s hostname.

You can control the license requests for an entire system using the **/etc/nslm.config** configuration file.

By default, the **/etc/nslm.config** file contains a line similar to the following:

```
:server @default
```

The argument @default may be changed to a colon-separated list of system names, system IP addresses, or broadcast IP addresses. Licenses will be requested from each of the entities found in the list, until a license is granted or all entries in the list are exhausted.

For example, the following setting prevents broadcast requests for licenses, by only specifying the local system:

```
:server localhost
```

The following setting requests a license from **server1**, then **server2**, and then a broadcast request if those fail to serve a license:

```
:server server1:server2:192.168.1.0
```

Similarly, you can control the license requests for individual invocations of the tools using the **NSLM_SERVER** environment variable. If set, it must contain a colon-separated list of system names, system IP addresses, or broadcast IP addresses as described above. Use of the **NSLM_SERVER** environment variable takes precedence over settings defined in **/etc/nslm.config**.

License Server

The NSLM license server is automatically installed and configured to run when you install NightStar RT.

The **`nslm`** service is automatically activated for run levels 2, 3, 4, and 5. You can check on these settings by issuing the following command:

```
/sbin/chkconfig --list nslm
```

In rare instances, you may need to restart the license server via the following command:

```
/sbin/service nslm restart
```

See **`nslm(1)`** for more information.

License Reports

A license report can be obtained using the **`nslm_admin`** utility.

```
nslm_admin --list
```

lists all licenses installed on the local system, current usage, and expiration date (for demo licenses). Use of the **`--verbose`** option also lists individual clients to which licenses are currently granted.

Adding the **`--broadcast`** option will list this information for all servers that respond to a broadcast request on the local subnet associated with the system’s hostname.

See **`nslm_admin(1)`** for more options and information.

Firewall Configuration for Floating Licenses

RedHawk does not support a firewall configuration by default, because iptables support is disabled. However, it is possible to build a custom kernel with iptables support enabled. If that is done, and floating licenses are used, the iptables firewall rules must be configured to allow the license requests and responses to pass.

If the system with iptables support and firewall rules is serving licenses, then the firewall rules must be arranged to allow license requests on UDP port 25517 and TCP port 25517 from any systems that will make license requests. For example, in a simple firewall, rules like the following, inserted before any `DROP` or `REJECT` rules, might work:

```
iptables -A INPUT -p udp -m udp -s subnet/mask --dport 25517 -j ACCEPT
iptables -A INPUT -p tcp -m tcp -s subnet/mask --dport 25517 -j ACCEPT
```

If the system with iptables support and firewall rules is running NightStar RT tools and receiving floating licenses, then the firewall rules must be arranged to allow license responses on UDP port 25517 from any system serving licenses. For example, in a simple firewall, rules like the following, inserted before any `DROP` or `REJECT` rules, might work:

```
iptables -A INPUT -p udp -m udp -s subnet/mask --sport 25517 -j ACCEPT
```

License Support

For additional aid with licensing issues, contact the Concurrent Software Support Center at our toll free number 1-800-245-6453. For calls outside the continental United States, the number is 1-954-283-1822. The Software Support Center operates Monday through Friday from 8 a.m. to 5 p.m., Eastern Standard Time.

You may also submit a request for assistance at any time by using the Concurrent Computer Corporation web site at **http://www.ccur.com/isd_support_contact.asp** or by sending an email to **support@ccur.com**.

B
Kernel Dependencies

Concurrent’s RedHawk kernel provides features and performance gains that are critical for the full operation of the NightStar RT tools.

The NightStar RT tools can operate in a host-only mode on Red Hat systems without Concurrent’s RedHawk kernel, cross-targeting to RedHawk systems. Additionally, the NightStar RT tools can function on Red Hat systems without the RedHawk kernel, but will lack the numerous advantages afforded by running with it.

The following sections describe the additional functionality and capabilities of the NightStar RT tools when running Concurrent’s RedHawk kernel.

Advantages for NightView

The following advantages are afforded NightView when Concurrent’s RedHawk kernel is running:

- Application speed conditions

  Provides “execution-speed” patches, conditions, and ignore counts.

- Hot operations

  Users of NightView gain the ability to read and write to a particular process without having to stop it. Thus, all eventpoints can be applied and modified during application program execution without stopping the process. User variables also can be read and modified without stopping the process.

- Signal handling

  Allows NightView to pass signals directly to a particular process, avoiding context switching.

NOTE

NightView may not operate at all on older versions of Red Hat without the RedHawk kernel.

Advantages for NightTrace

The following advantage is afforded NightTrace when Concurrent's RedHawk tracing kernel is running:

- Kernel tracing

  Users of NightTrace gain the ability to obtain kernel trace data and combine that with user trace data. Kernel tracing is an incredibly powerful feature that not only provides insight into the operating system kernel but also provides useful information relating to the execution of user applications.

The RedHawk kernel is provided in three flavors:

- Tracing
- Debug
- Plain

The Tracing and Debug flavors provide the features required for NightTrace kernel tracing. These kernels can be selected at boot-time from the boot-loader menu.

Advantages for NightProbe

The following advantages are afforded NightProbe when Concurrent's RedHawk kernel is running:

- Minimal intrusion

  Allows NightProbe to read and write variables without stopping the process for each sample or write operation.

- Sampling performance

  Allows NightProbe to use direct memory fetches for data sampling (as opposed to programmed I/O) which is important for high-rate data acquisition.

- Concurrent debugging/probing

  Allows NightProbe to probe programs already under the control of a debugger or another NightProbe session.

- PCI Device probing

  Allows NightProbe to probe PCI device memory via the Base Address Register (BAR) file system.

Advantages for NightTune

The following advantage is afforded NightTune when Concurrent’s RedHawk kernel is running:

- Context switch rate

  Allows NightTune user to display the context switch counts per CPU instead of for the overall system.

- CPU shielding

  Individual CPUs can be shielded from interrupts and processes allowing CPUs to be dedicated solely to specific interrupts and processes that are bound to the CPU.

- CPU sibling interference

  Individual CPUs can be marked down to avoid interfering with hyperthreaded sibling CPUs and dual-core sibling CPUs. Hyperthreaded CPUs share all the resources of their sibling CPU. Dual-core CPUs share the CPU cache and a path to memory with their sibling CPU.

Advantages for NightSim

The following advantage is afforded NightSim when Concurrent’s RedHawk kernel is running:

- Scheduling target

  Allows NightSim to schedule processes on the system via Concurrent’s Frequency-Based Scheduler.

C
Variables

This section describes the notation used to reference variables in program resources (see "Variable Name Notation"), and describes the criteria used to determine eligibility of a variable for probing (see “Variable Eligibility for Program Resources” on page C-4).

Variable Name Notation

Variable names may be used to identify memory addresses in C, C++, and Fortran, as well as Ada (using the MAXAda compiler) programs. NightProbe accepts and displays variables with the following syntax.

Syntax

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

Parameters

file	The source file name enclosed in double-quotes.
common	The common block name enclosed in slashes (or // for unnamed common blocks)
scope	The name of the scope. Includes the names of enclosing functions, packages, or composite variables. Each one is separated from the next by a dot (`.`). (See “Composite Types” on page C-2 for information about composite types.)
name	The name of the variable. The variable may be either a scalar, an array, a structure or record, or a component of a variable of a composite type.
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 C-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 E-3 and “Creating a View into the Device” on page E-20.

Composite Types

To NightProbe, arrays, C and C++ structures and unions as well as C++ classes and Ada records are *composite types*. Composite objects may be recorded as a whole or individual components within the object may be recorded.

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]
```

The previous examples are all equivalent.

In C and C++ as well as 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)`.

Variable Eligibility for Program Resources

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

As stated before, variable names may be used to identify memory addresses in 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 option to generate 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, <u>ineligible</u> for data monitoring/recording.

D
Keyboard Traversal

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 F-2 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 Main 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 3-1 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(7x)`** system manual page).

Table 3-1. NightProbe Accelerators

Accelerator	Menu Item	Action
<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> <A>	Resource Ì Add Item	Opens the Item Browser window.
<Control> <I>	Resource Ì New Item	Opens the Item Definition window.
<Control> <T>	Control Ì Connect	Connects to target system and resources.
<Control> <D>	Control Ì Disconnect	Disconnects from target system and resources.
<Control> <R>	Control Ì Start	Starts iterative sampling.
<Control> <P>	Control Ì Stop	Stops iterative sampling.
<Control> <L>	Control Ì Sample	Obtains a single sample
<F1>		Displays help for the component that currently has the focus
<Shift> <F1>		Performs same function as Help -> On Context

E
Tutorials

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

- “Probing Programs Tutorial” on page E-1 demonstrates probing a program written in C++ and C and Ada
- “Probing Devices Tutorial” on page E-18 demonstrates probing a PCI device

Probing Programs 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 records and structures.

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

- “Ada Sample - ada_sample.a” on page E-11
- “C Sample - c_sample.c” on page E-17

Creating and Selecting a Program

1. The source code for the sample program used in this tutorial, as well as the compiled and linked binary, can be found in the **`/usr/lib/NightProbe/Tutorial`** directory and are included at the end of this chapter for reference. The sample program contains C++ and C and Ada code (the latter compiled with the Concurrent MAXAda compiler).

2. Either copy the appropriate binary file for your system from **/usr/lib/NightProbe/Tutorial**, or copy the source files and compile the program:

   For example:

   ```
   /usr/bin/uncompress -c \
         /usr/lib/NightProbe/Tutorial/ada_sample.pentium.Z > ada_sample
   chmod 777 ada_sample
   ```

   - or -

   ```
   /usr/bin/uncompress -c \
         /usr/lib/NightProbe/Tutorial/ada_sample.amd64.Z > ada_sample
   chmod 777 ada_sample
   /usr/bin/uncompress -c \
         /usr/lib/NightProbe/Tutorial/cpp_sample.pentium.Z > sample
   chmod 777 sample
   ```

   - or -

   ```
   /usr/bin/uncompress -c \
         /usr/lib/NightProbe/Tutorial/cpp_sample.amd64.Z > sample
   chmod 777 sample
   ```

   - or -

   ```
   cp /usr/lib/NightProbe/Tutorial/ada_sample.a .
   cp /usr/lib/NightProbe/Tutorial/cpp_sample.cpp .
   cp /usr/lib/NightProbe/Tutorial/c_sample.c .

   cc -g -c c_sample.c
   gcc -g -c -o c_sample.o c_sample.c
   g++ -g -c -o cpp_sample.o cpp_sample.cpp
   g++ -g -o sample cpp_sample.o c_sample.o
   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:

   ```
   /usr/bin/nprobe &
   ```

   NightProbe displays the NightProbe main window.

4. Invoke the sample program with the following command:

   ```
   ./ada_sample
   ./sample
   ```

5. In the NightProbe main window, right-click the Resources icon in the Session Overview Area and select the Add Program... menu option.

   NightProbe displays the Program Window.

6. In the Program Window, press the Select... button to the right of the PID text field to bring up the Select Process ID dialog.

Select the executing process ada_sample as shown in the list of processes associated with your user.

NOTE

> You can narrow your search by entering ada_sample in the Filter field and pressing the Filter button.

Press Select after selecting the desired process.

7. In the Program Window, press the Add button. You will see your program in the Resources list in the Session Overview Area of the NightProbe main window.

Variable Browsing

The following sections provide an example of the use of the Item Browser window. For more information about the Item Browser window, see Chapter 11, “Item Browser”.

Right-click the Probe Items icon in the Session Overview Area of the NightProbe main window and select the Add Item from Program... menu option.

The Item Browser window appears and contains a single root item, ada_sample, representing our program resource. Expand that root item by clicking the control box to the left of the icon. FourThree new items will appear:

- Functions
- Globals
- Files
- Packages

Adding a global C variable

1. Expand the Globals list by clicking the control box to the left of the Globals icon.
2. Select the c_global_int variable by clicking on its icon once.
3. Press the Add button.

   The item now appears in the Probe Items list in the NightProbe main window and its color has turned to orange in the Item Browser to indicate it has been added.

4. Collapse the Globals list by clicking on the control box to the left of the Globals icon.

Adding a static C variable declared inside a function

1. Expand the Functions list by clicking the control box to the left of the Functions icon.
2. Expand the list of variables inside c_routine by clicking the control box to the left of the c_routine icon.
3. Expand the c_static_array component list by clicking the control box to the left of the c_static_array icon.
4. Select two components of the array by clicking on one, then clicking on the next one in the list while holding down the Shift key.
5. Press the Add button.

   The two components now appears in the Probe Items list in the NightProbe main window and their color has turned to orange in the Item Browser to indicate they have been added.

6. Click on the c_static_array icon once.
7. Now press the left arrow key on your keyboard until the Functions list is selected (if no action occurs, make sure that the c_static_array icon is selected and that your NumLock key is not on).
8. Press the Space to collapse the Functions list.

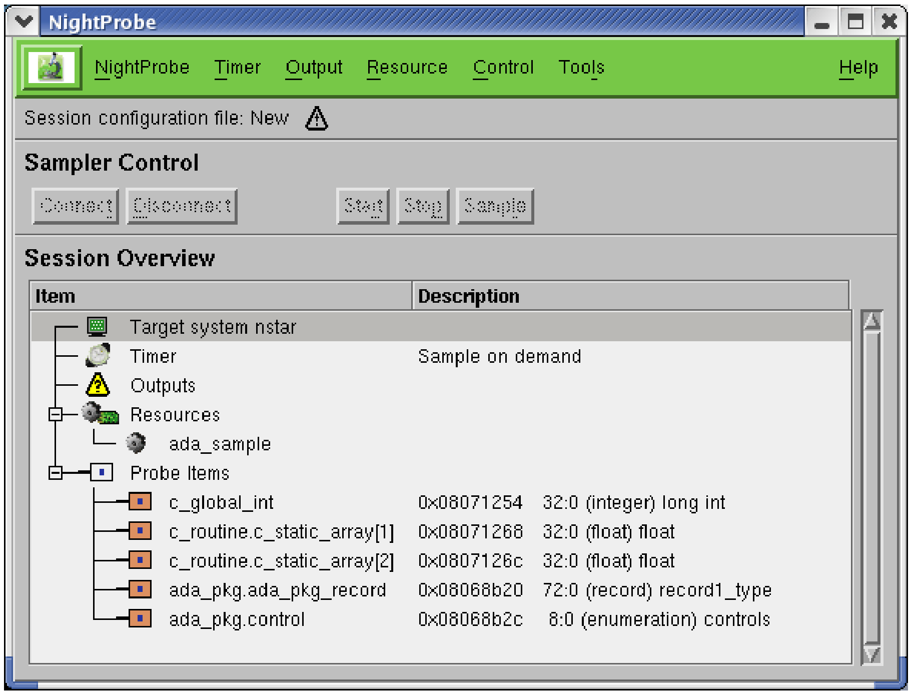
Adding Ada variables declared in packages

1. Now that you're an expert at navigating, browse the list of Packages until the variables in the package ada_pkg are visible.
2. Select the ada_pkg_record icon in the ada_pkg package.
3. Press the Add button to add the entire record to the list of Probe Items.
4. Add the control variable in the ada_pkg package as well.
5. Press the Done button to exit the dialog.

Adding C++ variables declared in classes

1. Expand the Files list by clicking the control box to the left of the Files icon.
2. Click the control box to the left of "cpp_sample.cpp".
3. Select the sample_class entry.
4. Press the Add button to add the entire class to the list of Probe Items.
5. Press the Done button to exit the dialog.

At this point, the NightProbe main window should look like this:


NightProbe
NightProbe Timer Output Resource Control Tools Help
Sampler Control
Connect Disconnect Start Stop Sample

Session configuration file: New

Session Overview

Item	Description
Target system nstar	
Timer	Sample on demand
Outputs	
Resources	
ada_sample	
Probe Items	
c_global_int	0x08071254 32:0 (integer) long int
c_routine.c_static_array[1]	0x08071268 32:0 (float) float
c_routine.c_static_array[2]	0x0807126c 32:0 (float) float
ada_pkg.ada_pkg_record	0x08068b20 72:0 (record) record1_type
ada_pkg.control	0x08068b2c 8:0 (enumeration) controls

If the list of Probe Items differs from the above figure, go back into the Item Browser and add the items shown above to the list.

NOTE

The addresses to the right of each item may differ from the figure above; that is expected and can be ignored.

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 “Spreadsheet Viewer” on page 14-1.

To launch the Spreadsheet Viewer

1. Right-click the Outputs icon in the Session Overview Area of the Night-Probe main window and select the Spreadsheet Output menu option.

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. All the variables from the Probe Items list in the NightProbe main window are shown in the list of variables in this window.

3. In the Spreadsheet Variables window, select all of the entries by 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 of the items should be highlighted.
4. Set the Cell Layout to Vertical.
5. Set the Label Position to Left.
6. Set the Expansion to Both.
7. Set the Direction to Vertical.
8. Press the OK button.

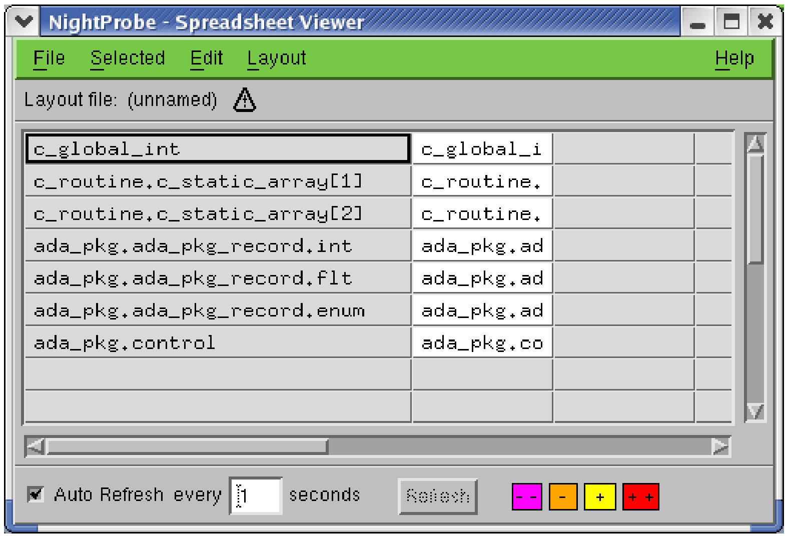
   The spreadsheet cells starting with the uppermost left cell now describe the variables you selected. The left-hand column is a label field which includes the name of the variable. This field can be edited. The right-hand column initially contains the same text, but will be replaced by the value of the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

9. Select a cell in the first column and use the Column Width menu option from the Layout menu to widen the column to 30 characters.
10. Select a cell in the second column and use Column Width menu option from the Layout menu to widen the column to 15 characters.

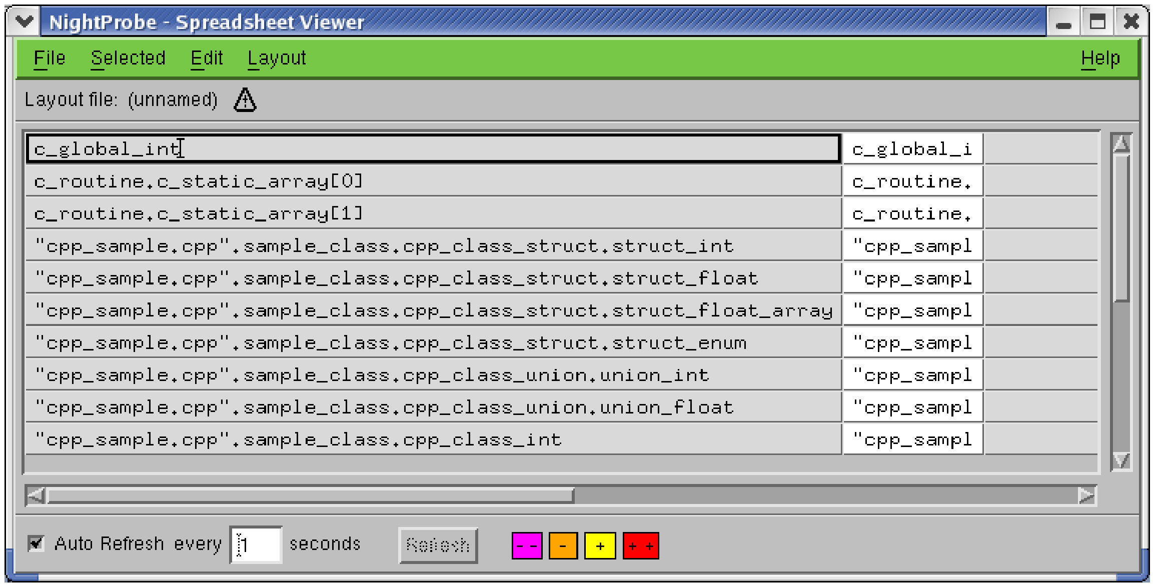
NOTE

You may need to resize the Spreadsheet Viewer window in order to see the expanded cells in their entirety.

At this point, the spreadsheet window should look like this:


NightProbe - Spreadsheet Viewer
File Selected Edit Layout Help
Layout file: (unnamed)
Auto Refresh every 1 seconds
Refresh
- -
-
+
+ +

c_global_int	c_global_i	
c_routine.c_static_array[1]	c_routine.	
c_routine.c_static_array[2]	c_routine.	
ada_pkg.ada_pkg_record.int	ada_pkg.ad	
ada_pkg.ada_pkg_record.flt	ada_pkg.ad	
ada_pkg.ada_pkg_record.enum	ada_pkg.ad	
ada_pkg.control	ada_pkg.co	


NightProbe - Spreadsheet Viewer
File Selected Edit Layout Help
Layout file: (unnamed)
Auto Refresh every 1 seconds
Refresh
- -
-
+
+ +

c_global_int	c_global_i	
c_routine.c_static_array[0]	c_routine.	
c_routine.c_static_array[1]	c_routine.	
"cpp_sample.cpp".sample_class.cpp_class_struct.struct_int	"cpp_sampl	
"cpp_sample.cpp".sample_class.cpp_class_struct.struct_float	"cpp_sampl	
"cpp_sample.cpp".sample_class.cpp_class_struct.struct_float_array	"cpp_sampl	
"cpp_sample.cpp".sample_class.cpp_class_struct.struct_enum	"cpp_sampl	
"cpp_sample.cpp".sample_class.cpp_class_union.union_int	"cpp_sampl	
"cpp_sample.cpp".sample_class.cpp_class_union.union_float	"cpp_sampl	
"cpp_sample.cpp".sample_class.cpp_class_int	"cpp_sampl	

If your spreadsheet differs significantly from the above figure, select the cells with content, remove their content using the Cut menu item from the Edit menu and repeat the steps above.

Selecting a Timing Source

1. In the NightProbe main window, right-click the Timer icon in the Session Overview Area and select the System Clock... menu option.

A Set System Timer window will appear.

2. Ensure the sampling rate is 1 second.
3. Press the Set Timer button.

Start Data Sampling

1. Connect to the target process by pressing the Connect button in the Sampler Control Area of the NightProbe main window.
2. Begin sampling data by pressing the Start button in the Sampler Control Area of the NightProbe main window.

   See the values of the variables displayed in the second column of cells in 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 actually unrelated to the frequency at which the sampling occurs; but by circumstance, they both currently happen to be 1 second.)

   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 being used interactively to peek and poke at variables.

Modifying the Value of Variables

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

1. Click the data value cell associated with `c_global_int` (in the second column of the first 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.

2. Type in a new value for the cell by backspacing over the existing text and entering 200.
3. Press the Enter key.

   The value of `c_global_int` has been modified and is displayed. The program increments the value once per second.

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

```
loop
   case ada_pkg.control is
   when halt   => exit program
   when run    => update variables
   when hold   => do nothing
   end case ;
   sleep 1 second
end loop ;
while (sample_class.control != cpp_class::halt) {
   switch (sample_class.control) {

   case cpp_class::run:
      sample_class.cpp_procedure();
      c_routine();
      local_float = sample_class.get_private_float() + 0.25;
      sample_class.set_private_float(local_float);
      break;

   case cpp_class::hold:
   case cpp_class::halt:
      break;
   }

   increment_counter();
   sleep(1);
}
```

4. Click on the cell showing the value of the `ada_pkg"cpp_sample.cpp".sample_class.control` variable (whose label is displayed in the cell to its left).

5. Backspace over the existing value and type in:

   ```
   hold
   ```

6. 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.

7. Click on the same cell and change the value back to:

   ```
   run
   ```

   which causes the program to resume updating variables.

8. Click on the same cell and change the value to:

   ```
   halt
   ```

   which causes the program to exit.

NOTE

Look at the terminal screen where you invoked the **`ada_sample`** program; it should have exited.

9. Exit NightProbe by choosing the Exit menu option from the NightProbe menu in the NightProbe main window.

   When asked about saving changes to the session, press the No button.

Ada Sample - ada_sample.a

```
package ada_pkg is
--
   type states is (none, init, freeze, start, stop, in_flight, approach, land);
   type controls is (halt, run, hold);

   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        : controls := halt;

   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 := ada_pkg.run;

   loop
      case ada_pkg.control is
      when ada_pkg.halt =>
         exit;
      when ada_pkg.run =>
         ada_routine;
         c_routine;
      when ada_pkg.hold =>
         null;
      end case;
      delay 1.0;
   end loop;
--
end ada_sample;
```

C++ Sample - cpp_sample.cpp

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

#define ARRAY_SIZE 4

class cpp_class {
public:
   enum states {
      none, init, freeze, start, stop, in_flight, approach, land
   };

   enum controls {
      halt, hold, run
   };

   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);

   controls       control;

   cpp_class (void);

   class nested {
   public:
      struct nested_struct {
         struct_type nested_struct_struct;
         int         nested_struct_int_array [ARRAY_SIZE];
      } cpp_nested_class_struct;

      nested (void);
   } nested;



   void cpp_procedure(void);

private:

   int   cpp_class_private_int;
   float cpp_class_private_float;

};


cpp_class::nested::nested(void){
   // Initialize variables in the nested cpp_class::nested class.

   cpp_nested_class_struct.nested_struct_struct.struct_int = 0;
   cpp_nested_class_struct.nested_struct_struct.struct_float = 0;
   cpp_nested_class_struct.nested_struct_struct.struct_enum = none;

   for (int i = 0; i < ARRAY_SIZE; i++) {
      cpp_nested_class_struct.nested_struct_int_array[i] = i * 100;
   }
}


cpp_class::cpp_class (void) {

   // Initialize variable 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;
   }

   control = run;

   cpp_class_private_int    = 100;
   cpp_class_private_float  = 100.0;
}

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
cpp_class::cpp_procedure()
{

   // Increment variables in cpp_class

   cpp_class_int++;
   cpp_class_float += 2.0 ;

   switch (cpp_class_enum) {
   case land:
      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 land:
      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;
   }


   // Increment variables in nested class in cpp_class.
   for (int i = 0 ; i < ARRAY_SIZE ; i++ ) {
      nested.cpp_nested_class_struct.nested_struct_struct.struct_int += 1;
      nested.cpp_nested_class_struct.nested_struct_struct.struct_float += 2.0;

      switch (nested.cpp_nested_class_struct.nested_struct_struct.struct_enum) {
      case land:
         nested.cpp_nested_class_struct.nested_struct_struct.struct_enum = none;
         break ;

      default:
         nested.cpp_nested_class_struct.nested_struct_struct.struct_enum =
            (states) (nested.cpp_nested_class_struct.
                                nested_struct_struct.struct_enum + 1);
         break ;
      }

      nested.cpp_nested_class_struct.nested_struct_int_array[i] += 1;
   }
}


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;
   }

}
```

```
// Driver program.  Continuously loop, calling the modules
// for each language.

cpp_class sample_class;

extern "C" void c_routine(void);

int
main() {

   static int main_static_int = 0;
   float      local_float     = 0.0;

   sample_class.control = cpp_class::run;

   while (sample_class.control != cpp_class::halt) {
      switch (sample_class.control) {

      case cpp_class::run:
         sample_class.cpp_procedure();
         c_routine();
         local_float = sample_class.get_private_float() + 0.25;
         sample_class.set_private_float(local_float);
         break;

      case cpp_class::hold:
      case cpp_class::halt:
         break;
      }

      increment_counter();
      sleep(1);
   }

   return main_static_int;
}
```

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;
   }
   }
}
```

Probing Devices Tutorial

This tutorial demonstrates NightProbe's ability to probe PCI devices. We will probe the `sync_clock` timer on the Real-Time Clock and Interrupt Module (RCIM).

NOTE

This tutorial is only applicable to systems running RedHawk Linux that have an RCIM installed.

This tutorial requires that you run as the `root` user or that your user has the `CAP_SYS_RAW_IO` system capability as described in "Capabilities" on page G-1.

Selecting the RCIM

We will use some type structures from a compiled program file to aid in viewing the RCIM device.

1. Copy the **`rcim.c`** source file from:

   ```
   /usr/lib/NightProbe/Tutorial/rcim.c
   ```

   and compile and link it in a working directory:

   ```
   cp /usr/lib/NightProbe/Tutorial/rcim.c .
   g++ -g -o rcim rcim.c
   ```

2. Invoke NightProbe:

   ```
   /usr/bin/nprobe &
   ```

3. Right-click the Resources icon and select the Add PCI Device... menu option in the Session Overview Area of the NightProbe main window.
4. Press the Search... button on the PCI Device Window dialog to launch the PCI Device Selection Window.
5. Scroll through the list of PCI devices until you see one labeled:

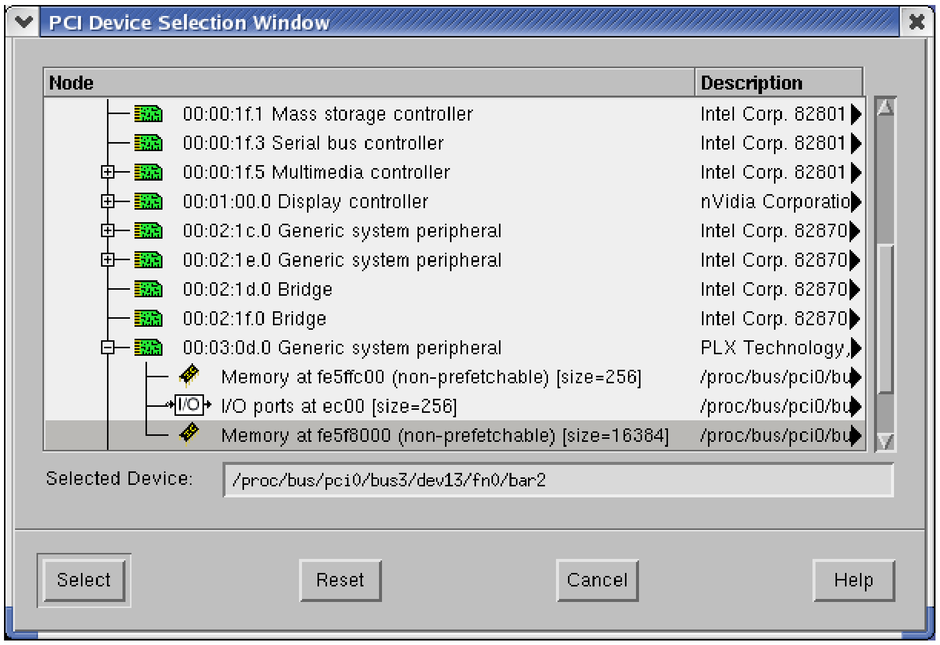
   PLX Technology, Inc. RCIM Realtime Clock and Interrupt Module

   or

   Concurrent Computer Corporation unknown device

6. Expand the list of regions for that device by clicking the control box to the left of the PCI card icon on that row.

The window contents should look similar to the following, with the actual set of PCI devices dependent on your specific system:



Node	Description
00:00:1f.1 Mass storage controller	Intel Corp. 82801
00:00:1f.3 Serial bus controller	Intel Corp. 82801
00:00:1f.5 Multimedia controller	Intel Corp. 82801
00:01:00.0 Display controller	nVidia Corporatio
00:02:1c.0 Generic system peripheral	Intel Corp. 82870
00:02:1e.0 Generic system peripheral	Intel Corp. 82870
00:02:1d.0 Bridge	Intel Corp. 82870
00:02:1f.0 Bridge	Intel Corp. 82870
00:03:0d.0 Generic system peripheral	PLX Technology,
Memory at fe5ffc00 (non-prefetchable) [size=256]	/proc/bus/pci0/bu
I/O ports at ec00 [size=256]	/proc/bus/pci0/bu
Memory at fe5f8000 (non-prefetchable) [size=16384]	/proc/bus/pci0/bu

Selected Device: /proc/bus/pci0/bus3/dev13/fn0/bar2

7. Select the third region for the RCIM Device, the Memory region listed after the I/O ports by clicking it once.

8. Press the Select button.

   The PCI Device Selection Window will exit and the Vendor ID, Device ID, Slot, and Region Number fields in the PCI Device Window will be populated.

9. Change the Offset text field in the lower portion of the PCI Device Window to the value `0x1000` (4096 decimal) which is the offset of the `sync_clock` timer on the RCIM.

10. In the Symbol File text field of the PCI Device Window, type in the name of the **`rcim`** program we built in the steps above:

    `./rcim`

11. Change the Resource Tag text field to `rcim` for clarity.

12. **STOP!**

    Make sure that the Offset field in Step 9 is set to 0x1000.

    Proceed.

13. Press the Add button.

The selected PCI device memory region will now appear in the Resources list in the Session Overview Area of the NightProbe main window.

Creating a View into the Device

In order to view locations within the PCI device, we need to create artificial variables to which we assign offsets and types. These variables are not allocated by NightProbe in the PCI device; they merely represent a view into a memory location that already exists in the device.

1. Right-click the PCI card icon in the Resources list in the Session Overview Area of the NightProbe main window and select the New Item for PCI Device... menu option.

   The Item Definition window appears. In the list area, you should see three root items representing Functions, Globals, and Files. The lists can be expanded so that you can browse for types to apply to the artificial variable you are creating within this dialog.

   NOTE

   If the only root item in the list is Globals, you forgot to specify a Symbol File value in the PCI Device Window. Exit this dialog, right-click on the PCI card icon in the NightProbe main window, choose Properties... and type in **`./rcim`** in the Symbol File text field and then return to this dialog.

2. Expand the Files list by clicking the control box to the left of the Files icon.
3. Expand the list of types inside rcim.c by clicking the control box to the left of the rcim.c icon.
4. Select the struct rcim_timer type by clicking on its icon once; do not expand the list of components.
5. Change the Item Tag text field to the value sync_clock for clarity in subsequent displays.
6. Press the Add button.

   The artificial variable sync_clock has been added to the Probe Items list in the Session Overview Area of the NightProbe main window.

Selecting the List Viewer

For brevity, we will select the simplest output method, the List Viewer window.

To launch the List Viewer

1. Right-click the Outputs icon in the Session Overview Area of the NightProbe main window and select the List Window Output menu option.

   A List Viewer window will appear and its textual contents will be empty.

Probing the RCIM

1. Press the Connect button in the Sampler Control Area of the NightProbe main window.

NOTE

If NightProbe pops up a diagnostic window that says you do not have sufficient permission to **mmap** the memory region, you need to either run as the root user or have your system administrator give you the CAP_SYS_RAW_IO capability as described in Capabilities in "Capabilities" on page G-1.

Save the current NightProbe session by selecting Save Session from the NightProbe menu in the NightProbe main window and then exit NightProbe. You can return to this spot in the tutorial as the root user or after you have been granted the required capability. Invoke **nprobe** with the name of the session file you just saved as its argument.

At this time we have connected to the target system and mapped the memory region of the RCIM into the sampler process's address space. No sampling or probing has yet occurred.

NOTE

For PCI device probing, it is usually best to use On Demand sampling since many devices may be sensitive to reads.

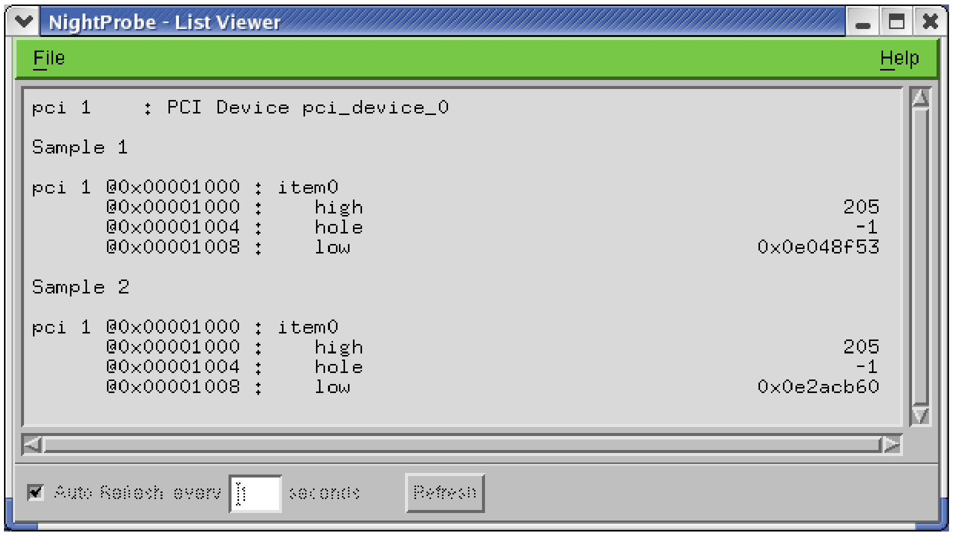
In the List Viewer window, a header has appeared:

```
pci 1   : PCI Device rcim
```

2. Press the Sample button in the Sampler Control Area of the NightProbe main window.
3. Press the Sample button again.

   The List Viewer window will automatically display the contents of the artificial variable we created in **rcim.c**.

The RCIM `sync_clock` timer has three portions: a high-order 32-bit value and a low-order 32-bit value separated by a 32-bit "hole". The low-order word ticks once every 400 nanoseconds. The high-order word ticks once the low-order word reaches 2**32 ticks.



```
pci 1    : PCI Device pci_device_0

Sample 1

pci 1 @0x00001000 : item0
      @0x00001000 :    high                                   205
      @0x00001004 :    hole                                    -1
      @0x00001008 :    low                             0x0e048f53

Sample 2

pci 1 @0x00001000 : item0
      @0x00001000 :    high                                   205
      @0x00001004 :    hole                                    -1
      @0x00001008 :    low                             0x0e2acb60
```

4. Exit NightProbe using the Exit menu option from the NightProbe menu of the NightProbe main window

C Source -- rcim.c

```
struct rcim_counter {
   int high;
   int hole;
   unsigned low;
};

struct rcim_counter counter;

main(){}
```

Conclusion

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

F
GUI Customization

You may customize the appearance of NightProbe using:

- X Window System resources (see "X Window System Resources" on page F-1)
- command-line options (see "Command-Line Options" on page F-2)
- application resources (see "Application Resources" on page F-2)

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, accelerators, and text messages.

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 F-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. One way to specify resources is to copy the default resource file to your home directory and change your version of NightProbe's resource file. That is the method used in this appendix. 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(7x)`** and **`xrdb(1)`** man pages.

The following files in the **`/usr/lib/X11/app-defaults`** directory contain NightProbe's default resources:

`Nprobe`	Application default resources file

You can look in this file for examples of ways to customize NightProbe's appearance and behavior. To see all the NightProbe resources, use the **`editres(1)`** tool.

Command-Line Options

NightProbe has its own set of command-line options. (See "Invoking NightProbe" on page 2-1.) 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`** *resourcestring* to set selected resources. For example:

```
nprobe -xrm "*drecWindow*geometry:-0+0" &
```

would put the `Main` window in the upper right corner. For a complete list of these options, refer to the **`X(7x)`** 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 resource file. See "Color Resources" on page F-4. The next time you run NightProbe, the color of all the input fields has the new setting.

Changing the `inputBackground` line in your **`Nprobe`** 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.

`lowCautionColor`

Control the color of the `Low Caution` field of the `Spreadsheet Viewer` window. The default value is orange. (See "Cell Color Legend" on page 14-13.)

`lowDangerColor`

Control the color of the `Low Danger` field of the `Spreadsheet Viewer` window. The default value is magenta. (See "Cell Color Legend" on page 14-13.)

`highCautionColor`

Control the color of the `High Caution` field of the `Spreadsheet Viewer` window. The default value is yellow. (See "Cell Color Legend" on page 14-13.)

`highDangerColor`

Control the color of the `High Danger` field of the `Spreadsheet Viewer` window. The default value is red. (See "Cell Color Legend" on page 14-13.)

`text.maxArrayExpansion`

Control the maximum number of array components which can are shown when expanding an array variable or type in the `Item Browser` and `Item Definition` windows. The default value is 1000 components. (See "Interactive Variable Browser" on page 11-3 and "Interactive Type Browser" on page 12-4.)

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.

Color Resources

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 in colors areas that <u>require</u> 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.

G
Target System Requirements

This chapter provides an overview of the user and system configuration requirements that need to be taken into account prior to running NightProbe on a target system.

Capabilities

The following capabilites may be required when using NightProbe:

- `CAP_SYS_RAWIO`

  If you wish to probe PCI devices or other target resources to which you do not have appropriate file access (e.g. **`/dev/mem`** or a shared memory segment or process owned by a different user), you must have this capability.

- `CAP_SYS_NICE`

  In order to set the CPU bias of the NightProbe target or of the program specified using the Program Output method (see “Program Output” on page 6-14), you must have this capability.

Linux provides a means to grant otherwise unprivileged users the authority to perform certain privileged operations. The Pluggable Authentication Module (see **`pam_capability(8)`**) is used to manage sets of capabilities, called *roles*, required for various activities.

Linux systems should be configured with an `nprobeuser` role which provides the `CAP_SYS_RAWIO` and `CAP_SYS_NICE` capabilities.

Edit **`/etc/security/capability.conf`** and define the `nprobeuser` role (if it is not already defined) in the “ROLES” section:

```
role nprobeuser CAP_SYS_RAWIO CAP_SYS_NICE
```

Additionally, for each NightProbe user on the target system, add the following line at the end of the file:

`user` *username* `nprobeuser`

where *username* is the login name of the user.

If the user requires capabilities not defined in the `nprobeuser` role, add a new role which contains `nprobeuser` and the additional capabilities needed, and substitute the new role name for `nprobeuser` in the text above.

In addition to registering your login name in **`/etc/security/capability.conf`**, certain files under the **`/etc/pam.d`** directory must also be configured to allow capabilities to be activated.

To activate capabilities, add the following line to the end of selected files in **`/etc/pam.d`** if it is not already present:

```
session required pam_capability.so
```

The list of files to modify is dependent on the list of methods that will be used to access the system. The following table presents a recommended configuration that will grant capabilities to users of the services most commonly employed in accessing a system.

Table 6-1. Recommended /etc/pam.d Configuration

/etc/pam.d File	Affected Services	Comment
`remote`	telnet rlogin rsh (when used w/o a command)	Depending on your system, the **`remote`** file may not exist. Do not create the **`remote`** file, but edit it if it is present.
`login`	local login (e.g. console) telnet* rlogin* rsh* (when used w/o a command)	*On some versions of Linux, the presence of the **`remote`** file limits the scope of the **`login`** file to local logins. In such cases, the other services listed here with **`login`** are then affected solely by the **`remote`** configuration file.
`rsh`	rsh (when used with a command)	e.g. **`rsh system_name a.out`**
`sshd`	ssh	You must also edit **`/etc/ssh/sshd_config`** and ensure that the following line is present: **`UsePrivilegeSeparation no`**
`gdm`	gnome sessions	
`kde`	kde sessions	

If you modify **`/etc/pam.d/sshd`** or **`/etc/ssh/sshd_config`**, you must resetart the **`sshd`** service for the changes to take effect:

```
service sshd restart
```

In order for the above changes to take effect, the user must log off and log back onto the target system.

NOTE

To verify that you have been granted capabilities, issue the following command:

```
/usr/sbin/getpcaps $$
```

The output from that command will list the roles currently assigned to you.

Index