



# **TimeTrac Event Analyzer User Guide**

**600-200141-000, Rev. 2.0**

# Copyright © 2007 SKY Computers, Inc.

27 Industrial Avenue • Chelmsford, Massachusetts 01824  
978-250-2420 • www.skycomputers.com

Including this documentation, and any software and its file formats and visual displays described herein, all rights reserved, may only be used pursuant to the applicable software license agreement; contains confidential and proprietary information of SKY and/or third parties which is protected by copyright, trade secret, and trademark law and may not be provided or otherwise made available without prior written authorization.

## Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clauses at DFARS 252.227-7013 (October 1988) and FAR 52.227-19 (c) (June 1987).

**All trademarks mentioned herein are the property of their respective companies.  
The following are trademarks of:**

*SKY Computers, Inc.*

TimeTrac™

*Georgia Tech Research Corporation*

VS IPL™

**The following are registered trademarks of:**

*Adobe Systems Incorporated*

Adobe Acrobat®

*Analog Devices, Incorporated*

SHARC®

*Intel Corporation*

Intel®, Pentium®

*International Business Machines Corporation*

PowerPC®

*MetaWare Incorporated*

MetaWare®, High C/C++®

*Microsoft Corporation*

Windows NT®, Windows 2000®

*Motorola, Inc.*

AltiVec®

*Sun Microsystems*

Java®, Solaris®

*Wind River Systems, Inc.*

VxWorks®, Tornado®

# Table of Contents

---

<b>Chapter 1</b>	
<b>About This Manual.....</b>	<b>7</b>
Overview.....	8
Chapter Organization.....	8
Conventions.....	8
<b>Chapter 2</b>	
<b>Introduction.....</b>	<b>9</b>
Events.....	10
Registering Events.....	10
Selecting an Event Name.....	10
Selecting a Color.....	11
Specifying a Group ID.....	11
Recording Events.....	12
Guidelines for Recording Events.....	12
Contexts.....	13
Maximum Context Depth.....	16
Group IDs.....	18
Compiling and Linking.....	18
Running TimeTrac From a Remote System.....	18
<b>Chapter 3</b>	
<b>TimeTrac Function Descriptions.....</b>	<b>19</b>
time_trac_close.....	20
time_trac_continue.....	21
time_trac_open.....	22
time_trac_pause.....	25
time_trac_perror.....	26
time_trac_pop_context.....	27
time_trac_push_context.....	28
time_trac_record.....	29
time_trac_reg_context.....	30

time_trac_reg_range_event.....	31
time_trac_reg_single_event .....	33
time_trac_save.....	35

## Chapter 4

### TimeTrac Viewer ..... 37

Invoking the Viewer .....	37
Name List.....	40
Menu Bar .....	41
File Menu.....	41
Load Trace File(s) .....	41
Reload Trace File(s) .....	42
Save Modified Trace File(s).....	42
Start Up Properties.....	42
Print .....	45
Exit.....	46
View Menu .....	46
CPU Usage/Event Statistics.....	46
Trace File Name at Tail of Tree/Trace File Name at Head of Tree .....	47
Hide Context Names/View Context Names .....	48
8 Point Font/10 Point Font/11 Point Font/12 Point Font.....	48
Sort Menu .....	49
Sort by Increasing Time	
Sort by Decreasing Time .....	49
Sort by Increasing CPU Usage	
Sort by Decreasing CPU Usage.....	50
Sort by Increasing Name	
Sort by Decreasing Name .....	52
Collapse Menu .....	53
All.....	53
Level 2 and Greater	
Level 3 and Greater	
Level 4 and Greater .....	54
Specify .....	54
Expand Menu.....	55
All.....	55
Level 1	
Level 1,2	
Level 1,2,3 .....	57
Specify .....	57
Events Window.....	58
Mouse Controls.....	59
Statistics Area .....	61
Search/Filters/Controls Area.....	62
Search .....	63
Controls.....	63
Filters .....	65
Information Bar.....	66

**Chapter 5**  
**Using the TimeTrac Viewer..... 67**

    Analyzing Algorithms.....67  
    Verifying the Order of Program Operations .....73



# 1

## About This Manual

---

Use this manual as a reference when using TimeTrac, SKY Computers' event recording and analysis tool. This manual documents TimeTrac concepts, the TimeTrac API, and the TimeTrac graphical user interface.

This manual is written for software developers interested in using TimeTrac to analyze, debug, and tune the performance of single and multiprocessor programs. This introductory chapter describes:

- [Overview](#)
- [Chapter Organization](#)
- [Conventions](#)

---

# Overview

TimeTrac is an event recording and analysis tool used to enhance the performance of single and multiprocessor programs.

Refer to Chapter 3, *TimeTrac Function Descriptions* for a description of the functions that comprise the TimeTrac API. Refer to Chapter 4, *TimeTrac Viewer*, for a detailed description of the TimeTrac graphical user interface.

---

## Chapter Organization

The chapters in this manual are organized in the following manner:

- Chapter 1, *About This Manual* — This introductory chapter.
- Chapter 2, *Introduction* — Provides a general overview of TimeTrac.
- Chapter 3, *TimeTrac Function Descriptions* — Provides function descriptions for each of the functions that make up the TimeTrac API.
- Chapter 4, *TimeTrac Viewer* — Provides a detailed description of the TimeTrac graphical user interface.
- Chapter 5, *Using the TimeTrac Viewer* — Provides several examples showing how to perform tasks using TimeTrac.

---

## Conventions

The following conventions are used throughout this manual:

numbers

Unless specifically noted, numbers are listed as decimal values. Hexadecimal values are preceded by 0x.

Courier regular

Within an interactive example, Courier regular type indicates information displayed on the terminal. This type is also used for all non-interactive examples, and for the names of routines, commands, and system calls in text.

*Courier italic*

Information to be supplied by the user, such as variable file names and arguments in system calls and command syntax.

Square brackets [..]

Delimit optional parameters.



# 2

## Introduction

---

TimeTrac is an event recording and analysis tool used to analyze, debug, and tune the performance of single and multiprocessor programs, including programs using threads.

If your application includes an algorithm that you want to accurately analyze for time constraints, or if you want to discover if it has any time-related problems, you can use TimeTrac to track and analyze events that occur within the algorithm. Unlike debuggers, which do not integrate timing information as a part of tracing variables and functions, and function profilers, which only provide the average execution time of a function with poor accuracy, TimeTrac can record when events occur and accurately display the execution time for a section of an algorithm. You can record an event occurrence or the elapsed time of a function any number of times in order to track its behavior. You can then analyze the recorded events using the TimeTrac viewer.

TimeTrac works in a multicore, multiprocessor environment. Events for a particular session are written to trace files using a format that is independent of the architecture on which the events were recorded. In this way, you can analyze an algorithm that is split between a single host and multiple embedded processors to determine how the processors are reacting and/or synchronizing with each other.

There are two components to TimeTrac:

- Recording functions  
These functions are added to application code and generate trace files that contain performance data from the application.
- TimeTrac viewer  
The viewer displays and analyzes the data in trace files.

---

# Events

Events are actions, routines, behaviors, and value changes in your application that you want to record in order to determine exactly what occurs when your application is running. Events are recorded using the TimeTrac recording functions, which store data for each occurrence of an event. These event records are stored in a buffer until they are written to a trace file, which can be displayed by the TimeTrac viewer. All events recorded during a particular trace session are included in the trace file.

There are two types of events:

- **Single event**  
A single event is an event that is recorded with a single time stamp indicating when the event occurred.
- **Range event**  
A range event is two single events that provide a start and stop time stamp. The data recorded for these events tell you when the event occurred, how long it took to complete, and when the event completed.

## Registering Events

Before an event can be recorded, a description of the event must be registered using one of the TimeTrac registration functions. To register an event, you must provide:

- An event name
- A color
- A group ID

### Selecting an Event Name

The event name is the name by which the event will be identified when displayed on the TimeTrac viewer. Event names should be selected to represent the type of event being recorded. Carefully selecting event names can provide the following benefits:

- You can sort event names into alphabetical order in the TimeTrac viewer.  
Registering events with meaningful event names can help you to organize the events for viewing.

- You can link two events using their event names.

The two events will be causally related if:

- The first event is named “send<\*>”. For example, “send\_p1”.
- The second event is name “recv<\*>”. For example, “recv\_p1”.

Note that “recv\_p1” is related to “send\_p1” but not to “sendp1” or “send\_p2”.

- The firing index of the second event is the same as that of the first.

The firing index is the number of times that a particular event is recorded. For example, if you record 1000 “add” events, then the tenth “add” event recorded has a firing index of 10, the twenty-fifth “add” event has a firing index of 25, and so on.

The send and receive events do not have to originate from the same trace file. Although event names must be unique within a trace file, different trace files can have events with the same name.

## Selecting a Color

When registering an event, specify the color in which the event will be displayed by the TimeTrac viewer. You may find it useful to display a particular set of registered events in a specific color, for example, all functions that send data.

## Specifying a Group ID

The group ID is a method of identifying a group of events. This identification can be used to filter events when you are recording events or displaying them on the TimeTrac viewer:

- Recording events

You can set a particular group ID so that the event is filtered out during event recording. In other words, when events with the specified group ID occur, they are not written to the event buffer.

- Viewing events

When you are displaying events on the TimeTrac viewer, you can specify that a particular group ID not be displayed. In this way, you can display only the events that you want to analyze rather than all of the recorded events.

## Recording Events

Once an event has been registered, it can be recorded using the TimeTrac recording functions. For example, the following uses a range event to time a sort routine:

```
/* register the range event */
time_trac_reg_range_event(trace_handle, "Quick Sort", "red",
                          0x1, &qsort_start, &qsort_end);

/* Record the start event */
time_trac_record(trace_handle, qsort_start, 0.0);

new_list = qsort(old_list);

/* Record the stop event */
time_trac_record(trace_handle, qsort_end, 0.0);
```

The TimeTrac recording functions record data into a buffer for each event that occurs. When the buffer is full, the data for the next recorded event replaces the data for the oldest event in the buffer.

You can manually save the buffer at any time. You can also specify that the buffer be saved automatically whenever it is full of unsaved event data. The buffer size is determined by a parameter set in `time_trac_open()`. Each event in the buffer requires 20 bytes.

When you save recorded events, you can either overwrite the existing trace file or append event data to the existing trace file. If the trace file does not already exist, it will be created.

For more information about TimeTrac functions, refer to [Chapter 3](#).

## Guidelines for Recording Events

One important consideration when using TimeTrac is the amount of data to record. If you record too many events, you may not be able to easily analyze them using the TimeTrac viewer. On the other hand, recording too few events may not produce an accurate analysis.

The following three guidelines may be useful in helping you determine how to organize and manage the events that you record:

- Use `time_trac_pause()` and `time_trac_continue()`

Use these two TimeTrac functions when you need to record events for a particular section of code but not the entire algorithm.

- Begin a new session

You can group events into separate trace sessions. The events occurring for each trace session are written to a separate trace file. This may be useful if you have some event occurrences that should always be recorded and others, such as those used during debugging, that are sometimes recorded. Multiple trace files can be used to organize events so that it is easier to analyze complex applications.

- Use group ids

When registering events, you can assign them to particular groups. Then, using the trace session mask, set in `time_trac_open`, you can record only the events that are assigned to specific groups. Refer to [Group IDs on page –18](#) for more information about assigning group ids.

---

## Contexts

A context is a name used to group events into a common set. Contexts are very useful in organizing a large list of recorded events when they are displayed in the TimeTrac viewer. A context is registered and recorded in the same manner as an event.

You can use `time_trac_push_context()` to make a particular context the current one for recording events and `time_trac_pop_context()` to revert to the previous context.



### Note

Although you only need to register a context one time, you can push and pop the context as many times as necessary.

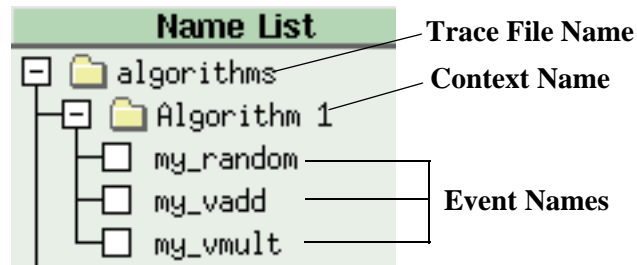
For example, the following uses a context to organize events:

```
/* register the context "Algorithm 1" */
time_trac_reg_context(handle, "Algorithm 1", &alg1_context);

/* make "Algorithm 1" the current context */
time_trac_push_context(handle, alg1_context);
.
{ record events }
.
.

/* revert to the previous context */
time_trac_pop_context(handle);
```

When the trace file produced by the previous example is displayed by the TimeTrac viewer, the recorded events will be organized under the specified context as shown below:



*Figure 1 Events Organized Using a Context*

You can use contexts to group common parts of algorithms as shown in the following example:

```

/* register the contexts "Modification 1" and */
/* Modification 2 */
time_trac_reg_context(handle, "Modification 1", &mod_1);
time_trac_reg_context(handle, "Modification 2", &mod_2);

/* make "Modification 1" the current context */
time_trac_push_context(handle, mod_1);
.
{ record events }
.
.

/* revert to the previous context */
time_trac_pop_context(handle);

/* make "Modification 2" the current context */
time_trac_push_context(handle, mod_2);
.
{ record events }
.
.

/* revert to the previous context */
time_trac_pop_context(handle);

```

When the trace file produced by the previous example is displayed by the TimeTrac viewer, the recorded events will be organized under the specified contexts as shown below:

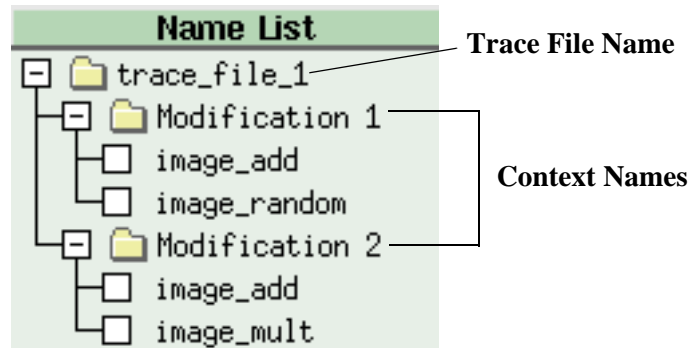


Figure 2 Events Organized Using Multiple Contexts

You can use nested contexts to further organize events as shown in the following example:

```
/* register contexts "Modify Image", Modification 1 */
/* and Modification 2 */
time_trac_reg_context(handle, "Modify Image", &mod_im);
time_trac_reg_context(handle, "Modification 1", &mod_1);
time_trac_reg_context(handle, "Modification 2", &mod_2);
```

```
/* push "Modify Image" into the stack */
time_trac_push_context(handle, mod_im);
```

```
/* make "Modification 1" the current context */
time_trac_push_context(handle, mod_1);
```

```
.
{ record events }
.
```

```
/* revert to the previous context */
time_trac_pop_context(handle);
```

```
/* make "Modification 2" the current context */
time_trac_push_context(handle, mod_2);
```

```
.
{ record events }
.
```

```
/* revert to the previous context */
time_trac_pop_context(handle);
time_trac_pop_context(handle);
```

When the trace file produced by the previous example is displayed by the TimeTrac viewer, the recorded events will be organized under the specified contexts as shown below:

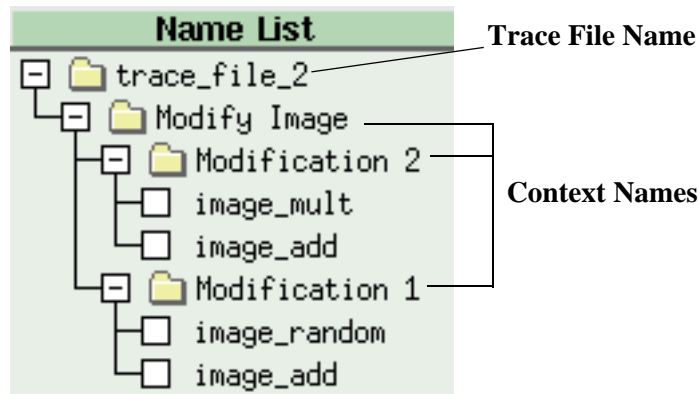


Figure 3 Events Organized Using Nested Contexts (Sorted by Decreasing Name)

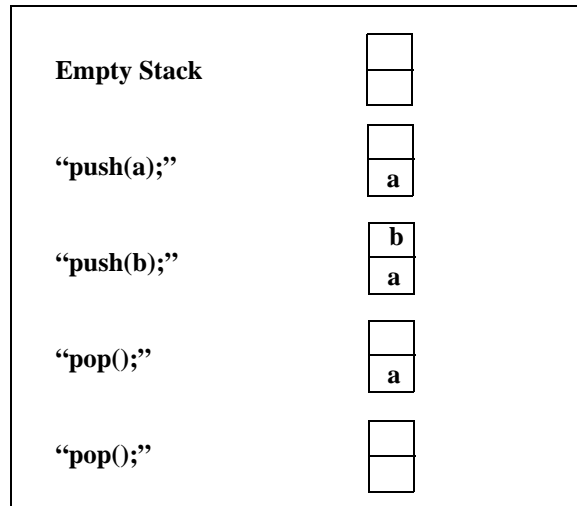
## Maximum Context Depth

The function `time_trac_open()` includes the parameter `maximum_context_depth`. The value you specify for this parameter defines the size of the “stack” of context names. For example, if you specify “5” for this parameter, the context name stack can contain five context names.

As with other stacks, you can push or pop context names in or out of the stack. Call `time_trac_push_context()` to push the specified context name to the top of the stack, that is, make the specified context name the current context name. Call `time_trac_pop_context()` to pop the current context name from the top of the stack, that is, revert to the previous current context name.



For example, if you set `maximum_context_depth = 2`, [Figure 4](#) illustrates the operation of the stack. Context “a” and context “b” are pushed into the stack and then are popped out of the stack.



*Figure 4 Example of Stack Operation*

Use contexts carefully with a recursive function or a deep set of functions because the maximum context depth can be quickly exceeded.

If a call to `time_trac_push_context()` exceeds the maximum context depth, then the specified context name will not be moved to the top of the stack and a status value will be returned. Use `time_trac_perror()` to view the status message:

```
Tried to push onto a full context list, time_trac_pop_context() required to clean up.
```

The TimeTrac software keeps track of the number of pushes that exceed the context depth. You will have to make the corresponding number of pops in order to return to the current context. The status message for a `time_trac_pop_context()` that corresponds to a `time_trac_push_context()` that exceeds the maximum context depth can be displayed using `time_trac_perror()`:

```
Context list was previously exceeded with a time_trac_push_context. Ignoring this pop.
```

In other words, push and pop function calls must always come in pairs.

This design allows you to avoid checking to determine if a push call failed. You must always call `time_trac_pop_context()` if you want to revert to a previous context, whether a call to `time_trac_push_context()` was successful or not. For this reason, use context functions carefully in recursive calls.

---

## Group IDs

One method of managing event recording is using group IDs. There are 32 groups to which you can assign an event. You make this assignment by setting one of the bits in the `group` parameter of the TimeTrac registration functions.

You can then control whether or not a particular group of events is recorded by using the `group_mask` parameter in the `time_trac_open()` function. By setting specific bits in the `group_mask` parameter, you will record only the events assigned to those specific groups during the trace session. All other events that are not assigned to those specific groups will not be recorded during the session.

---

## Compiling and Linking

After inserting TimeTrac function calls into your source code, compile and link the source code using the following compile and link time option:

```
cc -DTIME_TRAC hello.c -o hello
```



### Note

If you do not compile with the `-DTIME_TRAC` option, the pre-processor will remove all of the TimeTrac function calls.

---

## Running TimeTrac From a Remote System

There may be times when you want to run TimeTrac from a remote Linux system, that is, from a system other than one on which TimeTrac software has been loaded. To do this, perform the following:

1. Copy the TimeTrac executable to the remote system from:

```
/usr/bin/TimeTrac
```

2. Verify that the appropriate TimeTrac trace files reside on the remote system.

# 3

## TimeTrac Function Descriptions

---

This chapter provides descriptions of the functions that comprise the TimeTrac API. The functions include:

Function	Description
Trace Session	
<code>time_trac_open()</code>	Initializes a trace session.
<code>time_trac_close()</code>	Terminates the trace session.
Events	
<code>time_trac_reg_single_event()</code>	Registers a single event occurrence.
<code>time_trac_reg_range_event()</code>	Registers an event range (start event and stop event).
<code>time_trac_record()</code>	Records registered events.
<code>time_trac_pause()</code>	Halts event recording temporarily.
<code>time_trac_continue()</code>	Resumes event recording after it has been paused.
<code>time_trac_save()</code>	Writes the event buffer to a trace file.
Contexts	
<code>time_trac_reg_context()</code>	Registers a context name.
<code>time_trac_push_context()</code>	Establishes the specified context as current.
<code>time_trac_pop_context()</code>	Removes the current context.
Errors	
<code>time_trac_perror()</code>	Prints information about a valid status value to stdout.

**Table 1** TimeTrac Functions

## time\_trac\_close

Use `time_trac_close()` to stop the event recording session associated with the specified handle and free any resources used by that handle.

### Format

```
int time_trac_close (TTHandle handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

### Returns

TTE\_SUCCESS — The function completed successfully without errors.

TTE\_HANDLE\_INVALID — The specified handle is either NULL or not compatible with TimeTrac.

TTE\_COULD\_NOT\_OBTAIN\_MUTEX — The mutex required to maintain thread safeness could not be obtained.

### Notes

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

## time\_trac\_continue

Use `time_trac_continue()` to turn on event recording after it has been paused.

### Format

```
int time_trac_continue(TTHandle handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

### Returns

TTE\_SUCCESS — The function completed successfully without errors.

TTE\_HANDLE\_INVALID — The specified handle is either NULL or not compatible with TimeTrac.

TTE\_COULD\_NOT\_OBTAIN\_MUTEX — The mutex required to maintain thread safeness could not be obtained.

### Notes

If event recording is not paused, calling this function will have no effect.

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

`time_trac_pause()`

---

## time\_trac\_open

Use `time_trac_open()` to initialize an event recording session. Call this function in every process that requires event recording; it can be called more than once for each processor or thread. The function returns a handle that you use when recording and saving events. The recorded events are saved to a trace file, which can be used by TimeTrac.

### Format

```
int time_trac_open(const char *filename, int thread_safe_required, int num_events, unsigned group_mask,
  TTSaveEventsEnum save_when, TTFileOpenModeEnum file_open_mode, int maximum_context_depth,
  TTHandle *handle_ret);
```

### Arguments

#### *filename*

The name of the trace file into which recorded events will be saved. A ".trc" extension will automatically be appended to the filename, for example, `filename.trc` and the file will be saved to the current directory unless a pathname is specified.

When naming trace files, use a unique, descriptive names so that you know where a trace file originated and so that you do not accidentally overwrite data that you want to keep.

#### *thread\_safe\_required*

Use this argument to specify if the handle returned by this function will be used in more than one thread. Your choices are:

0 — Used in only one thread.

1 — Used in multiple threads.

#### *num\_events*

The maximum number of events that can be stored in the event buffer. The function allocates the memory for the event buffer based on this argument. When the buffer is full, a new event overwrites the oldest recorded event.

#### *group\_mask*

Use this argument to specify the groups to be recorded during the trace session. Setting a bit in `group_mask` indicates that all events assigned to that group will be recorded during a session. All other events will not be recorded during the trace session.

#### *save\_when*

Use this argument to indicate when the event buffer should be saved. Your choices are:

`TT_NO_AUTO_SAVE` — only save the event buffer manually, that is, only when you explicitly call `time_trac_save`.

`TT_SAVE_WHEN_BUFFER_FULL` — save the event buffer whenever the number of new events equals the value specified in `num_events`. If you select this option, you can also call `time_trac_save` at any time during the session.

*file\_open\_mode*

Use this argument to specify what to do when the trace file you are attempting to open already exists. Your choices are:

`TT_CREATE_FILE` — If the trace file already exists, truncate it to zero bytes.

`TT_CREATE_FILE_EXCLUSIVE` — If the trace file already exists, return an error but do not return a valid session handle.

*maximum\_context\_depth*

Use this argument to specify the number of nested contexts to use during the trace session. Refer to [Chapter 2](#) for more information about context depth.

*handle\_ret*

A pointer to a TimeTrac handle structure that is associated with a particular trace file. This handle is used in other function calls for recording events into this particular trace file.

**Returns**

`TTE_SUCCESS` — The function completed successfully without errors.

`TTE_INVALID_CARDINAL` — The value for `num_events` is invalid; it must be an integer greater than 0.

`TTE_CONTEXT_DEPTH_INVALID` — The value for `maximum_context_depth` is invalid; it must be an integer greater than 0.

`TTE_INVALID_SAVE_VALUES` — An invalid value was specified for `save_when`.

`TTE_INVALID_SAVE_MODE` — An invalid value was specified for `save_mode`.

`TTE_INVALID_FILE_OPEN_MODE` — An invalid value was specified for `file_open_mode`.

`TTE_HANDLE_INVALID` — The value specified for `handle_ret` is `NULL`.

`TTE_CANT_OPEN_FILE` — The specified file could not be open for write operations.

`TTE_BAD_GROUP_MASK` — An invalid value was specified for `group_mask`.

`TTE_NO_MEMORY` — There is insufficient memory to perform the current operation.

`TTE_INVALID_FILENAME` — The value specified for `filename` is `NULL`.

`TTE_MUTEX_NOT_CREATED` — The mutex for file saving and threads could not be created.

`TTE_COULD_NOT_OBTAIN_MUTEX` — The mutex required to maintain thread safeness could not be obtained.

**Notes**

When you call one of the registration functions to register an event, you can assign the event to a particular group by setting a bit in the `group` parameter.

**Include File**

```
#include <time_trac.h>
```

## See Also

`time_trac_close()`  
`time_trac_save()`  
`time_trac_reg_range_event()`  
`time_trac_reg_single_event()`  
`time_trac_record()`  
`time_trac_reg_context()`  
`time_trac_push_context()`  
`time_trac_pop_context()`  
`time_trac_pause()`  
`time_trac_continue()`  
`time_trac_perror()`



## time\_trac\_pause

Use `time_trac_pause()` to temporarily stop event recording for the specified trace session. After you call this function, `time_trac_record()` will have no effect until `time_trac_continue()` is called. By default, event recording is turned on.

### Format

```
int time_trac_pause(TTHandle handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

### Returns

TTE\_SUCCESS — The function completed successfully without errors.

TTE\_HANDLE\_INVALID — The specified handle is either NULL or not compatible with TimeTrac.

TTE\_COULD\_NOT\_OBTAIN\_MUTEX — The mutex required to maintain thread safeness could not be obtained.

### Notes

If event recording is already paused, call this function will have no effect.

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

`time_trac_continue()`

## time\_trac\_perror

Use `time_trac_perror()` to print to `stdout` an explanation of a valid status value.

### Format

```
void time_trac_perror(int status);
```

### Arguments

*status*

The status value returned from a previously called TimeTrac function.

### Returns

Does not return a value.

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

`time_trac_save()`

`time_trac_reg_range_event()`

`time_trac_reg_single_event()`

`time_trac_reg_context()`

## time\_trac\_pop\_context

Use `time_trac_pop_context()` to remove the current context name from the top of the list of context names, that is, remove it from being the current context. In addition, the function adds a “pop” event into the event buffer.

### Format

```
int time_trac_pop_context(TTHandle handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

### Returns

`TTE_SUCCESS` — The function completed successfully without errors.

`TTE_HANDLE_INVALID` — The specified handle is either `NULL` or not compatible with TimeTrac.

`TTE_POPPING_EMPTY_CONTEXT_LIST` — The pop operation failed because the context list is empty.

`TTE_POPPING_EXCEEDED_CONTEXT_LIST` — The pop operation is ignored because the context list was previously exceeded by a call to `time_trac_push_context`.

`TTE_COULD_NOT_OBTAIN_MUTEX` — The mutex required to maintain thread safeness could not be obtained.

### Notes

A `time_trac_pop_context()` that fails because the corresponding call to `time_trac_push_context()` exceeded the maximum context depth is not written to the event buffer. Only successful pop operations are written to the event buffer.

### Include Files

```
#include <time_trac.h>
```

### See Also

```
time_trac_open()  
time_trac_reg_context()  
time_trac_push_context()
```

---

## time\_trac\_push\_context

Use `time_trac_push_context()` to make the specified context name the current context name. In addition, the function adds a “push” event to the event buffer. From this point on, any recorded event is considered to be within the current context. To revert to the previous context name, call `time_trac_pop_context()`.

### Format

```
int time_trac_push_context(TTHandle handle, TTContextHandle context_handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

*context\_handle*

The handle returned by `time_trac_reg_context`.

### Returns

`TTE_SUCCESS` — The function completed successfully without errors.

`TTE_HANDLE_INVALID` — The specified handle is either `NULL` or not compatible with TimeTrac.

`TTE_CONTEXT_HANDLE_INVALID` — The TimeTrac context handle is not valid.

`TTE_CONTEXT_DEPTH_EXCEEDED` — The push operation failed to move the specified context into the stack because the maximum context depth has been exceeded. However, you must call `time_trac_pop_context` to pop this level.

`TTE_COULD_NOT_OBTAIN_MUTEX` — The mutex required to maintain thread safeness could not be obtained.

### Notes

A `time_trac_push_context()` that fails because it exceeds the maximum context depth is not written to the event buffer. Only successful push operations are written to the event buffer.

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

`time_trac_reg_context()`

`time_trac_pop_context()`

## time\_trac\_record

Use `time_trac_record()` to record an event and a value associated with the event. The value can be used to track floating point values, buffer sizes, bus usage, temperature, and so on.

### Format

```
int time_trac_record(TTHandle handle, TTEventHandle event_handle, float value);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

*event\_handle*

The handle returned by one of the registering functions. Alternatively, you can specify `TT_EVENT_IDLE_START` or `TT_EVENT_IDLE_END`, which are pre-registered events used to indicate that the process is waiting for some resource. These idle events can be recorded within a range event set.

*value*

A value associated with the recorded event. For example, this argument can be used to track floating point values such as variable value, buffer size, and so on. If you do not want a value associated with the event, use 0.0 as the value.

### Returns

`TTE_SUCCESS` — The function completed successfully without errors.

`TTE_HANDLE_INVALID` — The specified handle is either `NULL` or not compatible with TimeTrac.

`TTE_EVENT_HANDLE_INVALID` — The TimeTrac event handle is not valid.

`TTE_COULD_NOT_OBTAIN_MUTEX` — The mutex required to maintain thread safeness could not be obtained.

### Notes

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

`time_trac_reg_range_event()`

`time_trac_reg_single_event()`

## time\_trac\_reg\_context

Use `time_trac_reg_context()` to register a context name. Refer to [Chapter 2](#) for more information about contexts.

### Format

```
int time_trac_reg_context(TTHandle handle, const char *context_name, TTContextHandle *context_handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

*context\_name*

Use this argument to specify a unique name for the context. The TimeTrac viewer provides sorting by name; therefore, select a naming scheme that will make it easy to analyze the data.

*context\_handle*

A pointer to a TimeTrac handle structure for the specified context. The handle is used by `time_trac_push_context` to set the current context.

### Returns

TTE\_SUCCESS — The function completed successfully without errors.

TTE\_NULL\_NAME — The event name must not be NULL.

TTE\_NAME\_IN\_USE — The event/context name is already in use.

TTE\_NO\_MEMORY — There is insufficient memory to perform the current operation.

TTE\_HANDLE\_INVALID — The specified handle or context\_handle is either NULL or not compatible with TimeTrac.

TTE\_COULD\_NOT\_OBTAIN\_MUTEX — The mutex required to maintain thread safeness could not be obtained.

### Notes

```
#include <time_trac.h>
```

### Include Files

```
#include <time_trac.h>
```

### See Also

```
time_trac_open ( )
time_trac_push_context ( )
time_trac_pop_context ( )
time_trac_perror ( )
```

## time\_trac\_reg\_range\_event

Use `time_trac_reg_range_event()` to register a new “start” and “end” event. The function returns start and end handles that you can use with `time_trac_record` to record when a particular event begins and ends. This can be useful when you need to time a section of code.

### Format

```
int time_trac_reg_range_event(TTHandle handle, const char *event_name, const char *color, unsigned group,
TTEventHandle *start_handle, TTEventHandle *end_handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

*event\_name*

A unique name that identifies the event being recorded. The name is displayed on the TimeTrac viewer. Refer to [Chapter 2](#) for a discussion of naming conventions.

*color*

The color in which the event occurrences will be displayed on the TimeTrac viewer. The color can be either of the following formats:

“<*predefined color*>” — The name of any color (BLACK, WHITE, RED, etc) predefined by the windowing system (for X Window System, see `/usr/lib/X11/rgb.txt`).

“#rrggbb” — The 24-bit representation of the color.

rr: 256 values for red

gg: 256 values for green

bb: 256 values for blue

For example, pure white is `#ffffff`, pure black is `#000000`, and pure red is `#ff0000`.

*group*

Use this argument to specify the group to which the event is assigned. You specify a group by setting one of the 32 bits. For more information about group ids, refer to [Chapter 2](#).

*start\_handle*

A pointer to a TimeTrac handle structure for the start event. Use this handle in `time_trac_record`.

*end\_handle*

A pointer to a TimeTrac handle structure for the end event. Use this handle in `time_trac_record`.

## Returns

TTE\_SUCCESS — The function completed successfully without errors.

TTE\_NULL\_NAME — The event name must not be NULL.

TTE\_NAME\_IN\_USE — The event/context name is already in use.

TTE\_BAD\_COLOR — The color is not correctly specified.

TTE\_BAD\_GROUP — The value for `group` is invalid. Only one bit of the lower 32 bits can be set.

TTE\_NO\_MEMORY — There is insufficient memory to perform the current operation.

TTE\_HANDLE\_INVALID — The specified `handle`, `start_handle`, or `end_handle` is either NULL or not compatible with TimeTrac.

TTE\_COULD\_NOT\_OBTAIN\_MUTEX — The mutex required to maintain thread safeness could not be obtained.

## Notes

## Include Files

```
#include <time_trac.h>
```

## See Also

`time_trac_open ( )`

`time_trac_record ( )`

`time_trac_reg_single_event ( )`

`time_trac_perror ( )`



## time\_trac\_reg\_single\_event

Use `time_trac_reg_single_event()` to register a single event occurrence. The function returns a handle that you can use with `time_trac_record()` to record when a particular event occurs.

### Format

```
int time_trac_reg_single_event(TTHandle handle, const char *event_name, const char *color, unsigned group,
TTEventHandle *event_handle);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

*event\_name*

A unique name that identifies the event being recorded. The name is displayed on the TimeTrac viewer. Refer to [Chapter 2](#) for a discussion of naming conventions.

*color*

The color in which the event occurrences will be displayed on the TimeTrac viewer. The color can be either of the following formats:

“<predefined color>” — The name of any color (BLACK, WHITE, RED, etc) predefined by the windowing system (for X Window System, see `/usr/lib/X11/rgb.txt`).

“#rrggbb” — The 24-bit representation of the color.

rr: 256 values for red

gg: 256 values for green

bb: 256 values for blue

For example, pure white is `#ffffff`, pure black is `#000000`, and pure red is `#ff0000`.

*group*

Use this argument to specify the group to which the event is assigned. You specify a group by setting one of the 32 bits. For more information about group ids, refer to [Chapter 2](#).

*event\_handle*

A pointer to a TimeTrac handle structure for the event. Use this handle in `time_trac_record`.

## Returns

TTE\_SUCCESS — The function completed successfully without errors.

TTE\_NULL\_NAME — The event name must not be NULL.

TTE\_NAME\_IN\_USE — The event/context name is already in use.

TTE\_BAD\_COLOR — The color is not correctly specified.

TTE\_BAD\_GROUP — The value for `group` is invalid. Only one bit of the lower 32 bits can be set.

TTE\_NO\_MEMORY — There is insufficient memory to perform the current operation.

TTE\_HANDLE\_INVALID — The specified handle or event handle is either NULL or not compatible with TimeTrac.

TTE\_COULD\_NOT\_OBTAIN\_MUTEX — The mutex required to maintain thread safeness could not be obtained.

## Notes

## Include Files

```
#include <time_trac.h>
```

## See Also

`time_trac_open ( )`

`time_trac_record ( )`

`time_trac_reg_range_event ( )`

`time_trac_perror ( )`

## time\_trac\_save

Use `time_trac_save()` to force all valid events in the event buffer to be saved to the trace file associated with the specified handle. The saved events will remain in the event buffer until the buffer is full. At that time, each new event will overwrite the oldest event in the buffer.

### Format

```
int time_trac_save (TTHandle handle, TTSaveModeEnum file_save_mode);
```

### Arguments

*handle*

The value returned by a call to `time_trac_open`.

*file\_save\_mode*

Use this argument to specify the mode to use when saving events. Your choices are:

`TT_OVERWRITE_FILE` — Truncate the trace file to 0 bytes before writing new events to it.

`TT_APPEND_TO_FILE` — Write new events at the end of the trace file without affecting events already stored in the file.

### Returns

`TTE_SUCCESS` — The function completed successfully without errors.

`TTE_HANDLE_INVALID` — The specified handle is either `NULL` or not compatible with TimeTrac.

`TTE_SAVE_REENTERED` — The function `time_trac_save` has been re-entered, possibly via an interrupt handler.

### Notes

Since it is time consuming to save events to a file on the host disk, avoid calling this function at places in your code where high performance is most important.

### Include Files

```
#include <time_trac.h>
```

### See Also

`time_trac_open()`

`time_trac_close()`

`time_trac_perror()`



# 4

## TimeTrac Viewer

---

The TimeTrac viewer is the tool used to display the trace files that you have previously recorded using the TimeTrac recording functions. You can view a single trace file or multiple trace files to show how they react together. The viewer is very useful in displaying:

- Where speed improvements could be made in algorithms
- Which processors are overloaded
- Causal relationships
- Synchronizing communications or semaphores
- Operating system context switches
- Memory contention

---

### Invoking the Viewer

To invoke the TimeTrac viewer, enter the following:

```
% TimeTrac <options>
```

Any trace files (\* .`trc`) located in the current directory will be loaded and displayed in the viewer. You can specify any of the following options at the TimeTrac command line:

Option	Description
-path <directory   file>	Loads the trace file(s) located in the specified directory, which can be a full pathname or a relative pathname. This option overrides the default action which is to load the trace file in the current directory.
-colormap	Forces the viewer to allocate its own color map. This is useful on systems having an 8-bit color table instead of 24-bit color.
-m	Displays the mouse control functions.
-v	Returns the version information, then exits.
-h	Displays the available startup options, then exits.
<all standard X server options>	Accepts standard X server options, such as <code>-display</code> . This is only valid if running on a UNIX platform.

**Table 2** *Command Line Options for the TimeTrac Viewer*

The major components of the TimeTrac viewer, shown in [Figure 5](#), are:

Component	Description
Name List	Hierarchical list of trace files, contexts, and event names.
Menu Bar	Menu selections for the TimeTrac viewer.
Event Window	Window in which event occurrences are displayed.
Statistics Area	Area in which event statistics and CPU usage are displayed.
Search/Controls/Filters Area	Area in which you can search the Name List and change the Events Window display.
Information Bar	Information display for the currently loaded trace file(s).

**Table 3** *Major Components of the TimeTrac Viewer*

Each of the components is described in the following sections.

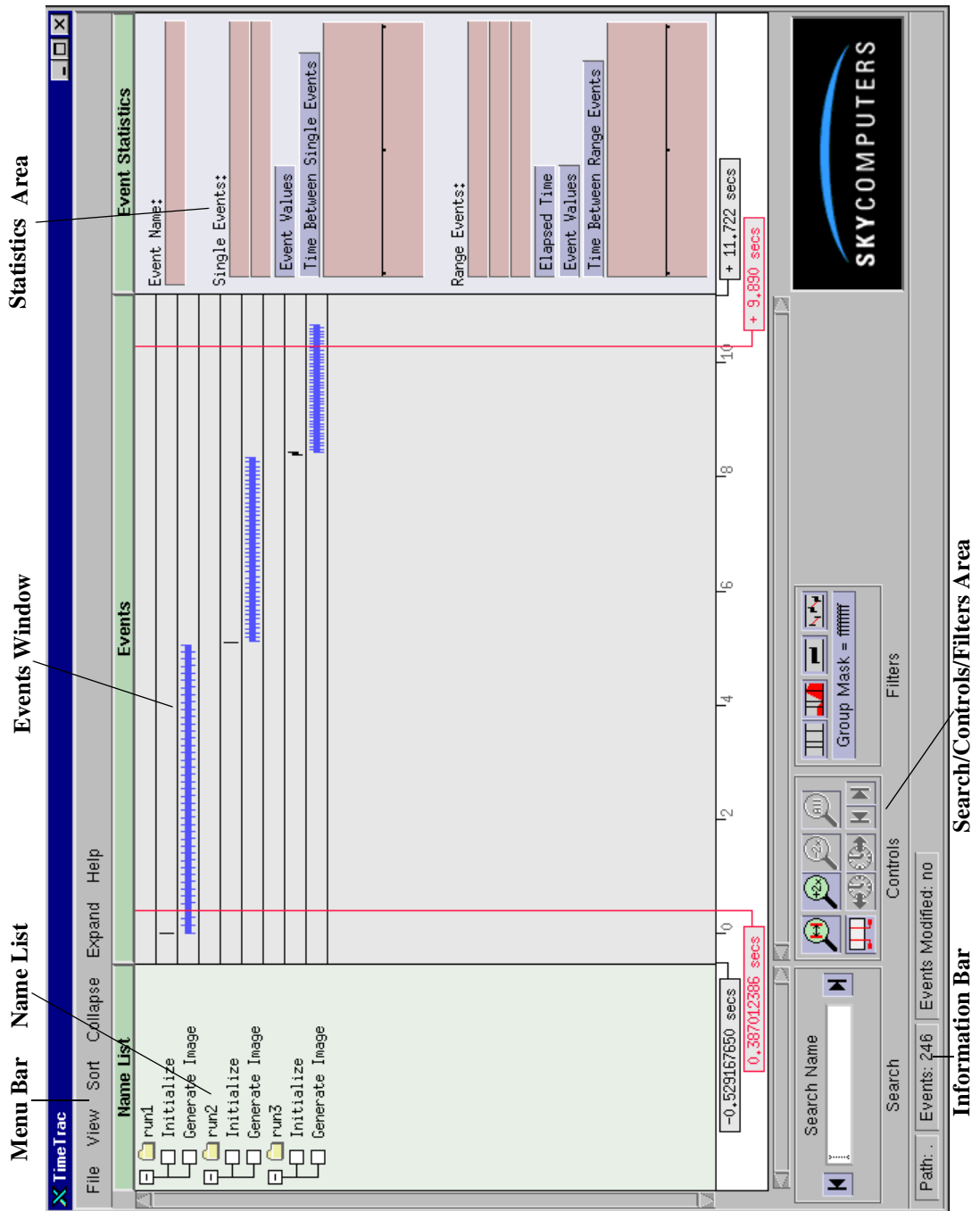


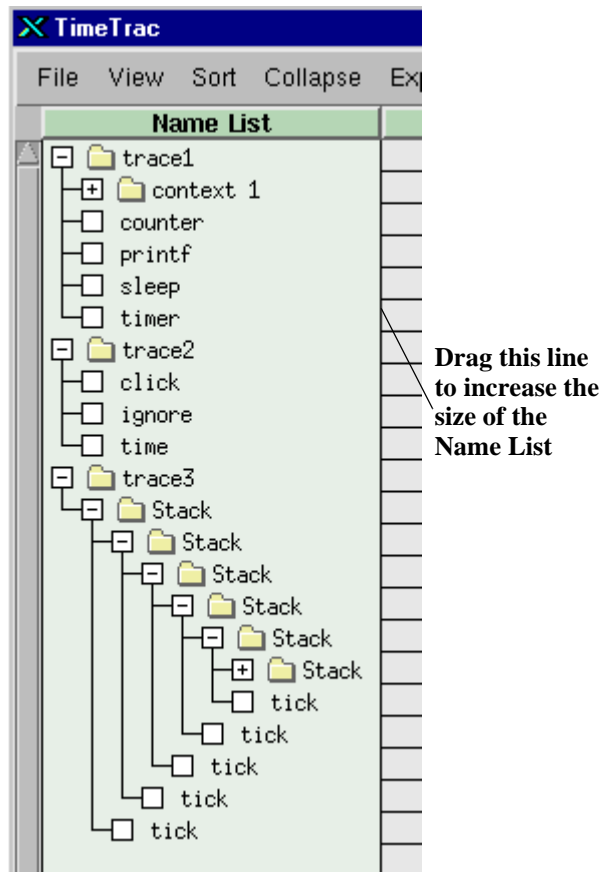
Figure 5 TimeTrac Viewer

## Name List

The Name List, shown in [Figure 6](#), is located beneath the menu bar on the left-hand side of the TimeTrac viewer. It displays the hierarchy of the names contained in the currently loaded trace files.

As shown in [Figure 6](#), the Name List consists of the names of the trace files (`trace1`), context names (`context 1`, when used), and names of events (`counter`). The levels of the hierarchy are denoted by folder icons. To the left of each folder icon is a white box containing a “+” or “-” character, indicating that the level is hidden (“+”) or expanded (“-”). You can expand or hide a level by moving the mouse pointer over this box and clicking the left mouse button.

Alternatively, you can expand or hide levels using the Collapse and Expand pull-down menus.



*Figure 6 Name List Section of the TimeTrac Viewer*

Use the scroll bar to the left of the Name List to navigate a long list of names. To increase the size of the Name List, move the mouse pointer over the line that separates the Name List and the Events Window, shown in [Figure 6](#). The mouse pointer changes to a two-ended arrow. You can then drag the line to increase the size of the Name List.



---

## Menu Bar

The TimeTrac viewer includes the menu bar shown in [Figure 7](#).

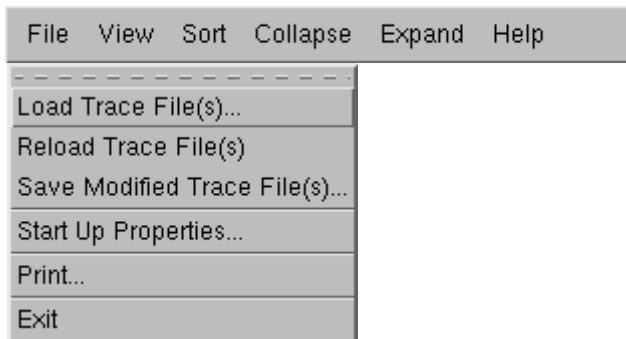


*Figure 7 TimeTrac Viewer Menu Bar*

Each of the pull-down menus has a number of selections, which are described in the following sections.

## File Menu

The File menu includes the selections shown in [Figure 8](#).



*Figure 8 File Menu Selections*

## Load Trace File(s)

The **Load Trace File(s)** menu selection enables you to select which trace files (\*.trc) will be loaded and displayed in the TimeTrac viewer.

Clicking **Load Trace File(s)** displays the Open Event Files dialog box. You can then load individual trace files by selecting a particular trace file. If you select a directory, all of the trace files located in that directory will be loaded. Click OK to open the trace files.



### Note

If you already have trace files loaded into the viewer and you have made modifications, you are prompted about saving the current trace files before loading new ones.

## Reload Trace File(s)

The **Reload Trace File(s)** selection enables you to reload the trace files that are currently displayed by the TimeTrac viewer.



### Note

If you made modifications to the current trace files, you are prompted to save them before reloading.

## Save Modified Trace File(s)

The **Save Modified Trace File(s)** menu selection enables you to save any changes that you made to the currently displayed trace files.

Clicking **Save Modified Trace File(s)** displays an input box into which you enter the directory location where you want the trace files to be saved.

## Start Up Properties

The **Start Up Properties** menu selection enables you to specify how you want the TimeTrac viewer to be set up when it is invoked.

Clicking **Start Up Properties** displays the Start Up Options dialog box, shown in [Figure 9](#), from which you make your selections.

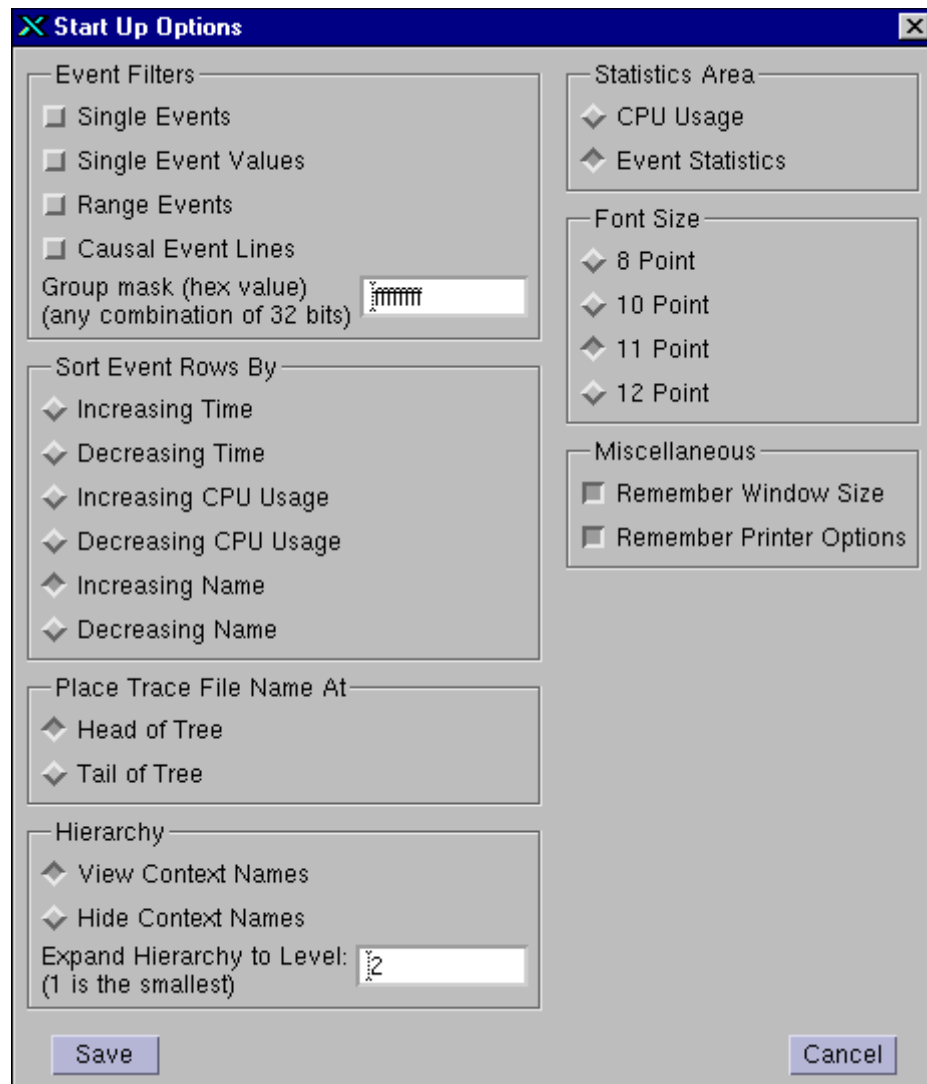


Figure 9 Start Up Options Dialog Box

Select the options that you want, then click **Save** to store the options for the next time that the TimeTrac viewer is invoked. The viewer options are stored in a file (.time\_trac\_prefs) located in your home directory.

The dialog box includes the following options:

Option	Description
<b>Event Filters</b>	
Single Events	Displays/Hides single events in the Events Window.
Single Event Values	Displays/Hides values of single events in the Events Window.
Range Events	Displays/Hides range events in the Events Window.
Causal Event Lines	Displays/Hides causal event lines in the Events Window.
Group mask	Set bits in the group mask to control the events that are displayed in the Events Window.
<b>Sort Event Rows By:</b>	
Increasing Time	Sorts events by time in the Events Window, from the first recorded time scale time to the last.
Decreasing Time	Sorts events by time in the Events Window, from the last recorded time scale time to the first.
Increasing CPU Usage	Sorts events by CPU usage in the Events Window, from least CPU used per event to most CPU used.
Decreasing CPU Usage	Sorts events by CPU usage in the Events Window, from most CPU used per event to least CPU used.
Increasing Name	For each level of the Name hierarchy, sorts names in alphanumerical order.
Decreasing Name	For each level of the Name hierarchy, sort names in inverse alphanumerical order.

**Table 4 Start Up Options**

Option	Description
Place Trace File Name At:	
Head of Tree	Displays the hierarchy of the Name List with events organized beneath their associated trace files.
Tail of Tree	Displays the hierarchy of the Name List with events organized with their associated trace files listed beneath them.
Hierarchy	
View Context Names	Displays events in the Name List with their associated context names.
Hide Context Names	Displays events in the Name List with their associated context names hidden.
Expand Hierarchy to Level: (1 is the smallest)	Specifies the level of the hierarchy to display in the Name List.
Statistics Area	
CPU Usage	Displays the percentage of CPU usage per event row in the Statistics Area of the viewer.
Event Statistics	Displays information about selected events in the Statistics Area of the viewer.
Font Size	
8 Point	Uses 8 point font size for viewer display.
10 Point	Uses 10 point font size for viewer display.
11 Point	Uses 11 point font size for viewer display.
12 Point	Uses 12 point font size for viewer display.
Miscellaneous	
Remember Window Size	When you exit the viewer, saves the current size of the window for the next time the viewer is invoked.
Remember Printer Options	When the viewer is exited, saves the current printer options for the next time that the viewer is invoked.

**Table 4** Start Up Options

## Print

The **Print** menu selection enables you to print the TimeTrac viewer display. Clicking **Print** from the File menu displays the Printer Options dialog box from which you can select the options to use when printing the viewer display to a file or to a printer. Click Print from the dialog box to execute the print operation.

## Exit

The **Exit** menu selection closes the TimeTrac viewer.

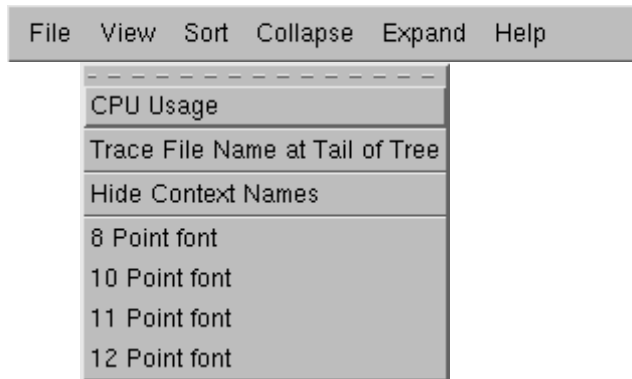


### Note

If you made modifications to the current trace files, you are prompted to save them before exiting.

## View Menu

The View menu includes the selections shown in [Figure 10](#).



*Figure 10 View Menu Selections*

## CPU Usage/Event Statistics

This menu selection toggles between the **CPU Usage** and **Event Statistics** options. Selecting one of these options displays its corresponding information in the Statistics Area of the viewer.

Clicking **CPU Usage** displays the percent of CPU usage for each row of event occurrences in the Events Window.



### Note

For each row of event occurrences, the percentage of CPU usage reflects only the event occurrences that are visible in the Events Window. For CPU usage percentages of the entire row of event occurrences, zoom out so that the entire row appears in the display.

Clicking **Event Statistics** displays statistics for the currently selected row of event occurrences. The event statistics include the number of occurrences, frequency, CPU usage (range events only), elapsed time (range events only), event values, and time between events.



### Note

For the selected row, the statistics reflect only the event occurrences that are visible in the Events Window. For the event statistics of the entire row of event occurrences, zoom out so that the entire row appears in the display.

## Trace File Name at Tail of Tree/Trace File Name at Head of Tree

This menu selection toggles between the **Trace File Name at Tail of Tree** and **Trace File Name at Head of Tree** options. The options control the display of events in the Name List of the viewer. Clicking **Trace File Name at Head of Tree** displays the hierarchy of the Name List with events organized under their associated trace files. As shown in [Figure 11](#) (left), events are organized under their trace files.

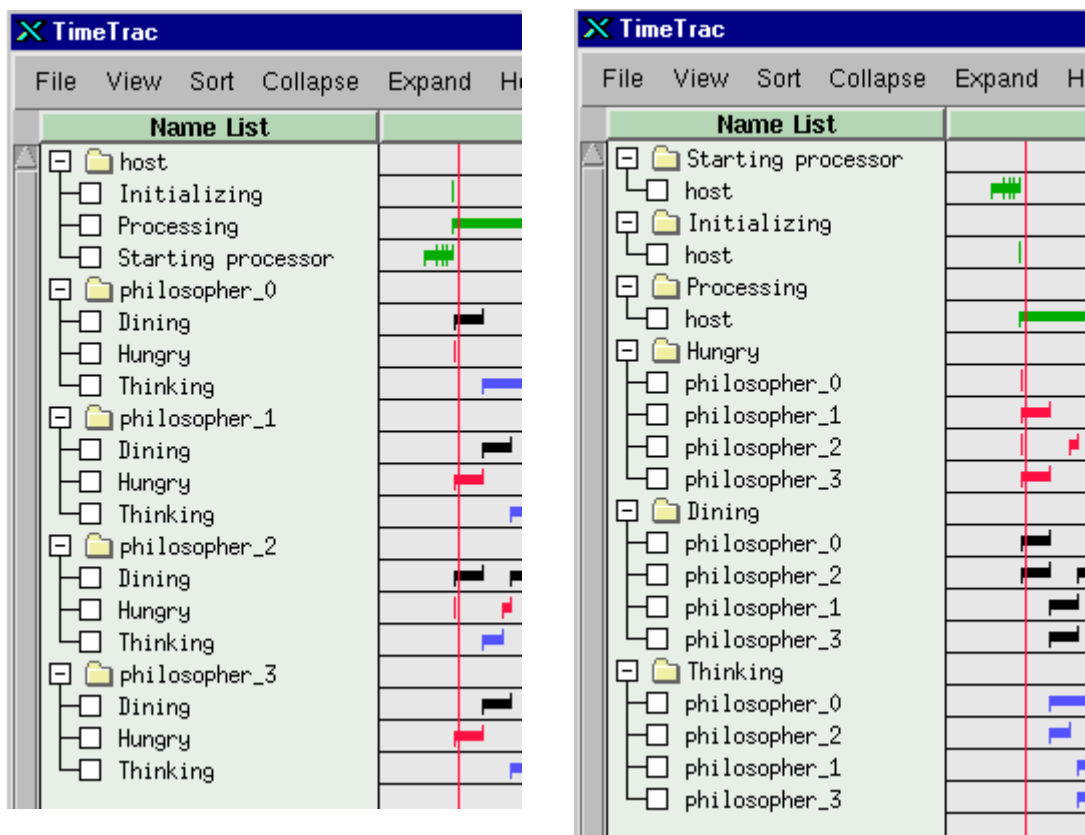
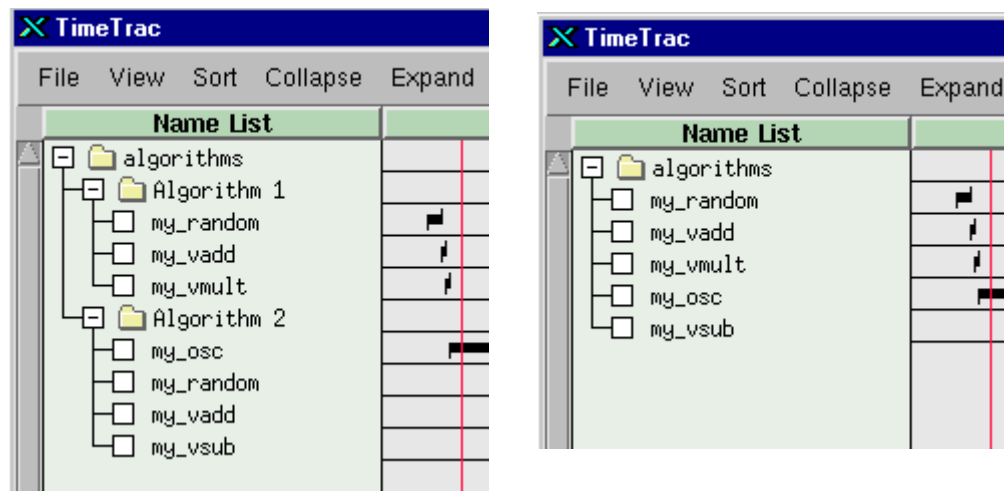


Figure 11 Name List Displaying Head of Tree Selection

The same display is shown in [Figure 11](#) (right) with **Trace File Name at Tail of Tree** selected. Notice that the events are listed first with their associated trace file listed beneath them. This is very useful when an event appears in more than one trace file.

## Hide Context Names/View Context Names

This menu selection toggles between the **Hide Context Names** and **View Context Names** options. The options control whether context names are displayed in the Name List of the viewer. When the **View Context Names** option is selected, the hierarchy of the Name List shows events organized under their associated context names. As shown in [Figure 12](#) (left), the events are organized under their context names (Algorithm 1 and Algorithm 2).



*Figure 12 Name List Displaying Context Names*

The same display is shown in [Figure 12](#) (right) with the **Hide Context Names** option selected. Notice that the events are listed without their associated context name.

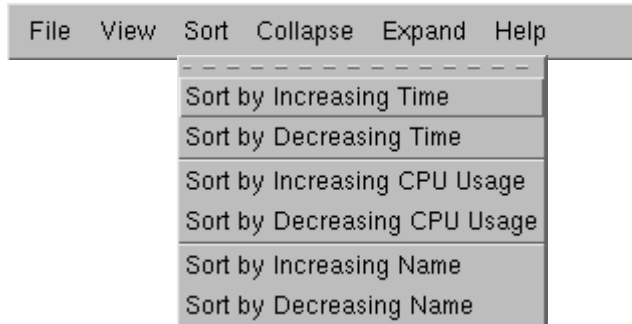
## 8 Point Font/10 Point Font/11 Point Font/12 Point Font

These menu selections control the point size of the viewer display. You can set the font size of the viewer display to 8, 10, 11 or 12 point.



## Sort Menu

The Sort menu includes the selections shown in [Figure 13](#).

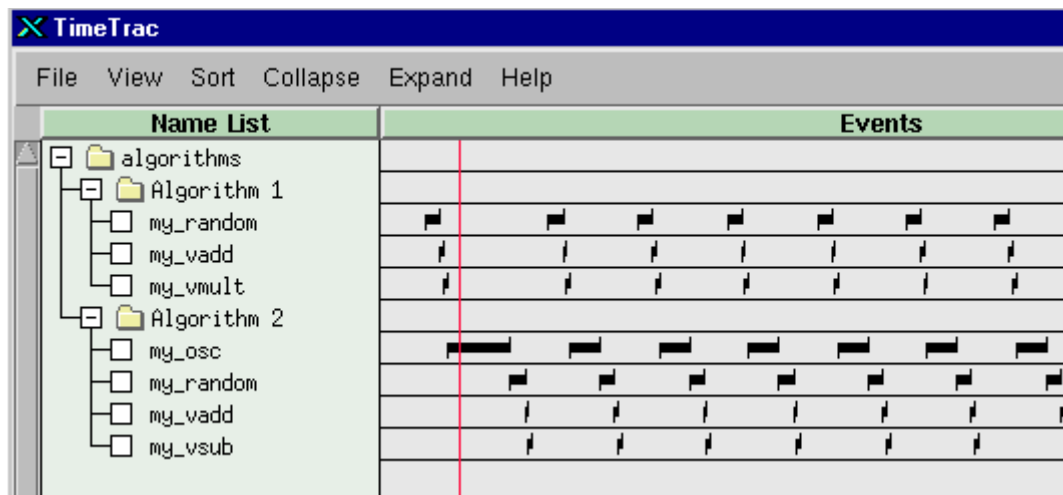


*Figure 13 Sort Menu Selections*

### Sort by Increasing Time Sort by Decreasing Time

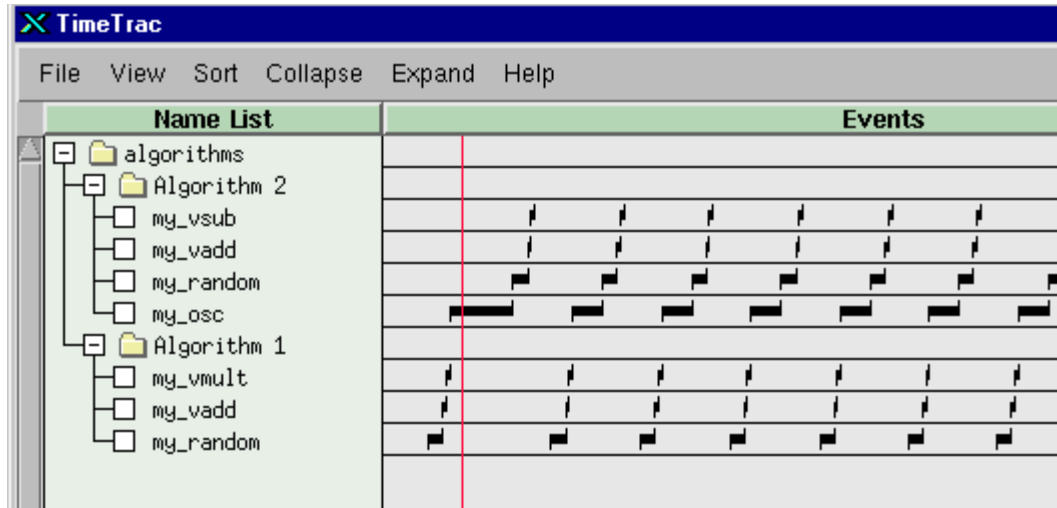
These menu selections sort the events corresponding to each hierarchical level of the Name List according to time. For each level, the sorting is performed on the first event in each row of event occurrences, then, the levels are sorted in the same manner.

For example, clicking **Sort by Increasing Time** sorts events from the earliest occurring event on the time scale to the latest, as shown in [Figure 14](#). Notice that for the levels **Algorithm 1** and **Algorithm 2**, the first event in each row of event occurrences for the level is sorted from the earliest occurring event on the time scale to the latest, then, both levels are sorted in the same manner.



*Figure 14 Events Sorted by Increasing Time*

Clicking **Sort by Decreasing Time** sorts events from the latest occurring event on the time scale to the earliest, as shown in [Figure 15](#). In this example, for the levels `Algorithm 1` and `Algorithm 2`, the first event in each row of event occurrences for the level is sorted from the latest occurring event on the time scale to the earliest, then, both levels are sorted in the same manner.



*Figure 15 Events Sorted by Decreasing Time*

## Sort by Increasing CPU Usage

## Sort by Decreasing CPU Usage

These menu selections sort the events corresponding to each hierarchical level of the Name List according to CPU Usage. The sorting is performed on the first event in each row of event occurrences for a level. Each level is then sorted in the same manner.

For example, clicking **Sort by Increasing CPU Usage** sorts events from the most CPU usage to the least, as shown in [Figure 16](#). Notice that for the levels `Algorithm 1` and `Algorithm 2`, the first event in each row of event occurrences for the level is sorted from the most CPU usage to the least. Both levels are then sorted in the same manner.

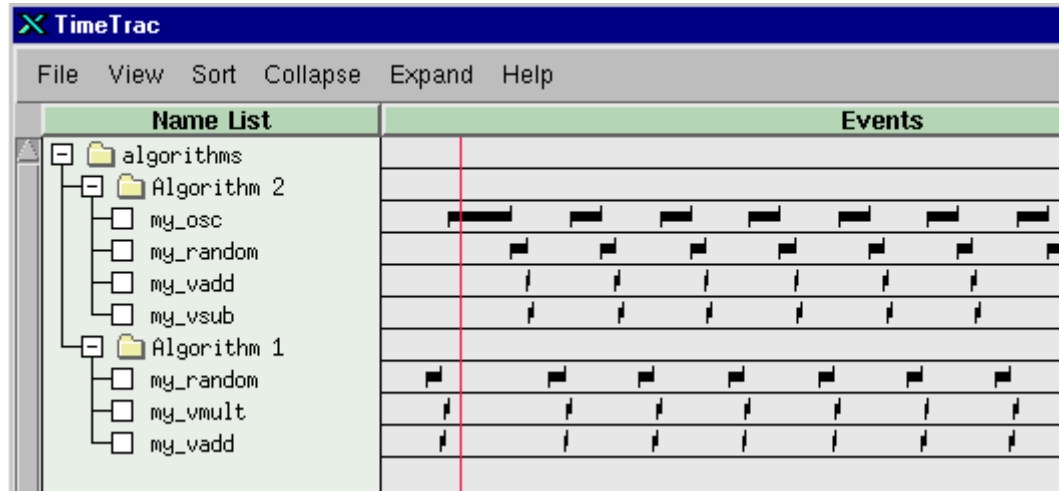


Figure 16 Events Sorted by Increasing CPU Usage

Alternatively, clicking **Sort by Decreasing CPU Usage** sorts events from the least CPU usage to the most, as shown in Figure 17. Notice that for the levels Algorithm 1 and Algorithm 2, the first event in each row of event occurrences for the level is sorted from the least CPU usage to the most. Both levels are then sorted in the same manner.

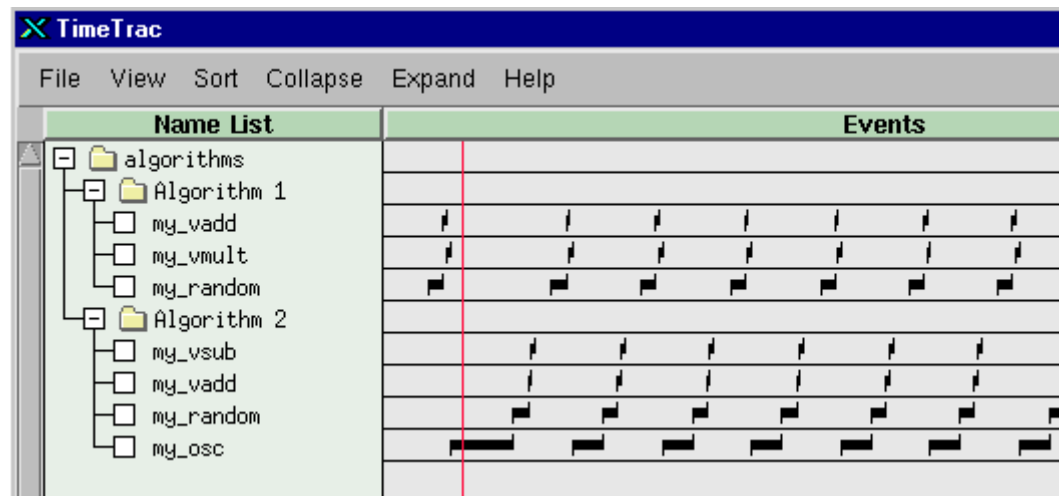


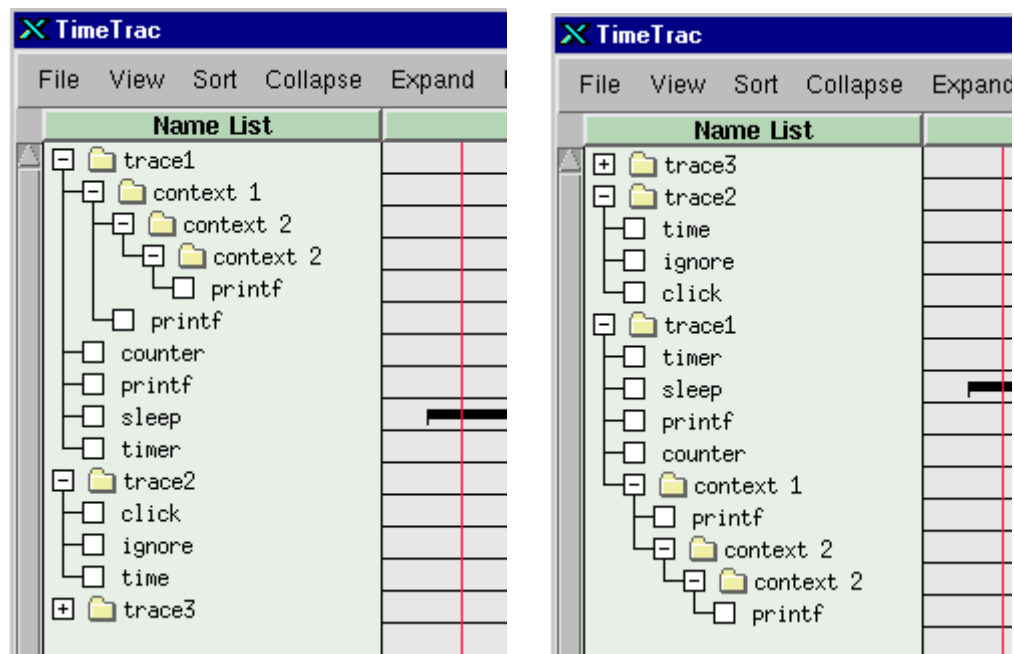
Figure 17 Events Sorted by Decreasing CPU Usage

## Sort by Increasing Name

## Sort by Decreasing Name

These menu selections alphanumerically sort the names in each hierarchical level of the Name List. The sorting is performed on each name of a level. The name of each level is then sorted in the same manner.

For example, clicking **Sort by Increasing Name** organizes the names in the Name List in alphanumeric order, as shown in [Figure 18](#) (left). Notice that for the levels `trace 1` and `trace 2`, the names in each level are alphanumerically ordered. The levels are then sorted in the same manner.



*Figure 18 Name List Sorted in Alphanumeric Order*

Alternatively, clicking **Sort by Decreasing Name** organizes the names in the Name List in inverse alphanumeric order, as shown in [Figure 18](#) (right). Notice that for the levels `trace 1` and `trace 2`, the names in each level are sorted in inverse alphanumeric order. The levels are then sorted in the same manner.

## Collapse Menu

The Collapse menu includes the selections shown in [Figure 19](#).

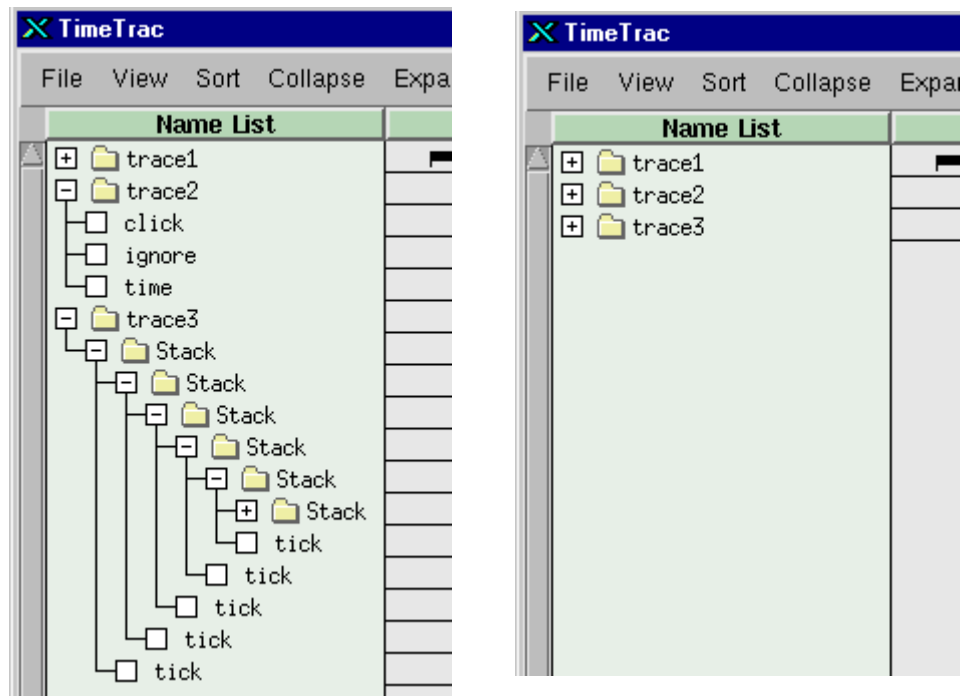


*Figure 19 Collapse Menu Selections*

### All

The **All** menu selection reduces the hierarchy in the Name List so that only the first levels display. All other levels are collapsed and hidden.

Clicking **All** when the Name List appears as in [Figure 20](#) (left) causes the viewer to hide all of the levels in the hierarchy except the first level, as shown in [Figure 20](#) (right).



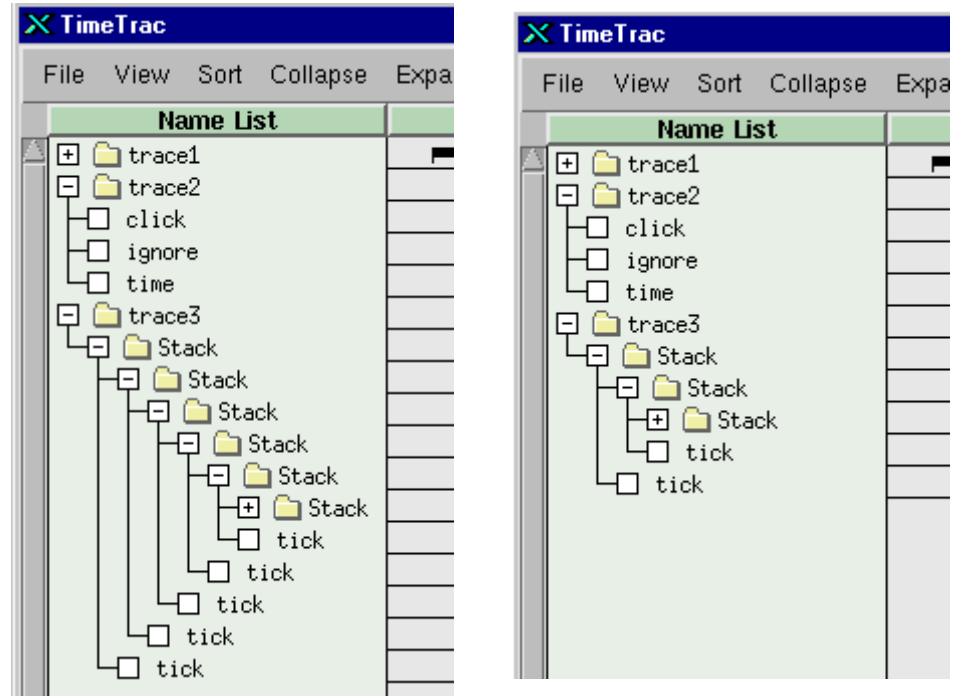
*Figure 20 Name List - Collapse All Option*

## Level 2 and Greater

## Level 3 and Greater

## Level 4 and Greater

These menu selections control the number of levels of the hierarchy that display in the Name List. For example, clicking **Level 4 and Greater** hides all of the levels greater than or equal to four, as shown in [Figure 21](#) (right).



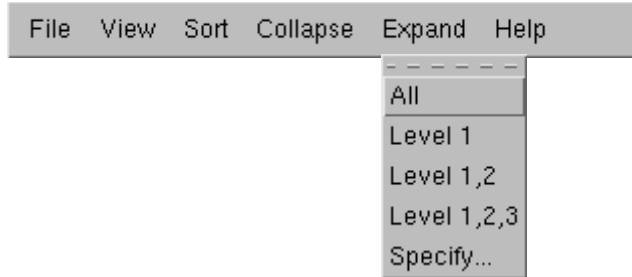
*Figure 21 Name List - Collapse Level 4 or Greater Option*

## Specify

The **Specify** menu selection enables you to select the level of the hierarchy that displays in the Name List. Clicking **Specify** displays an input box into which you enter a number such that all levels greater or equal to the specified number are hidden in the Name List. For example, if a trace session contains 10 levels and you enter 4 into the Input box, then all levels 4 or greater are hidden.

## Expand Menu

The Expand menu includes the selections shown in [Figure 22](#).

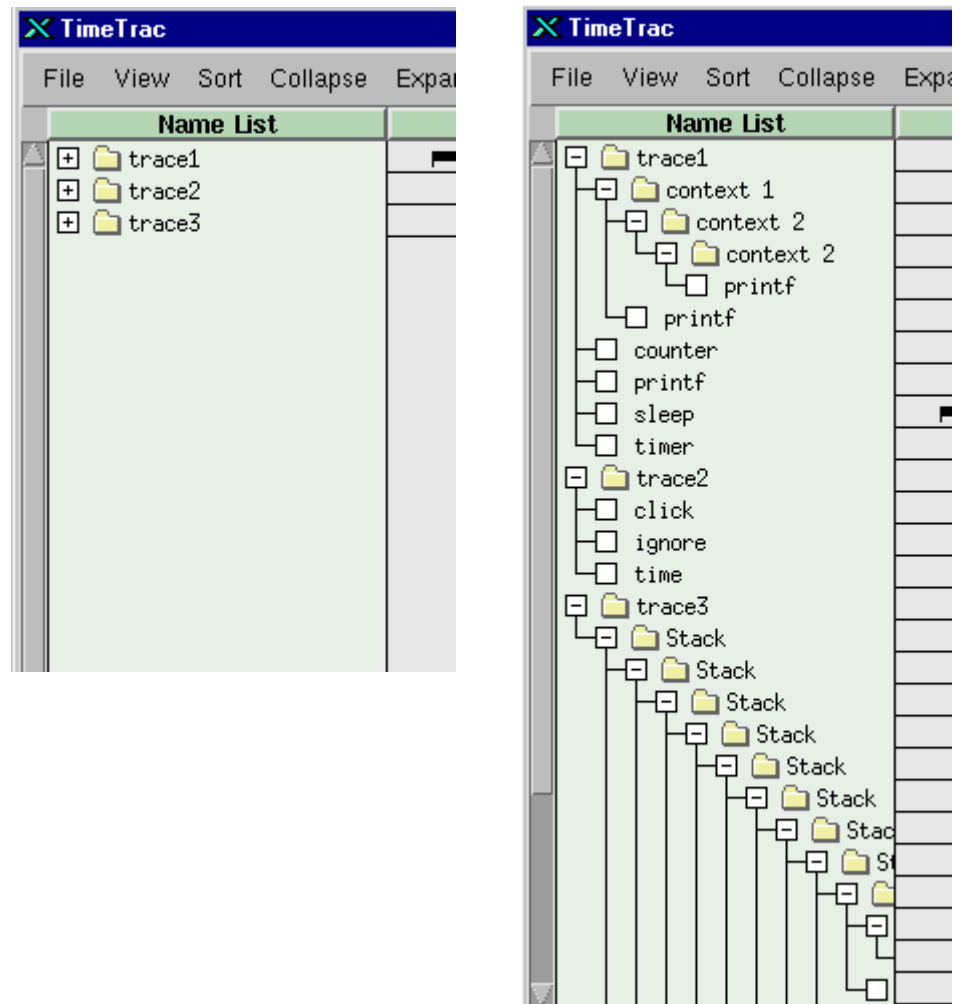


*Figure 22 Expand Menu Selections*

### All

The **All** menu selection enables you to expand all of the levels of the hierarchy in the Name List.

Clicking **All** when the Name List appears as in [Figure 23](#) (left) causes the viewer to expand all of the levels in the hierarchy, as shown in [Figure 23](#) (right).



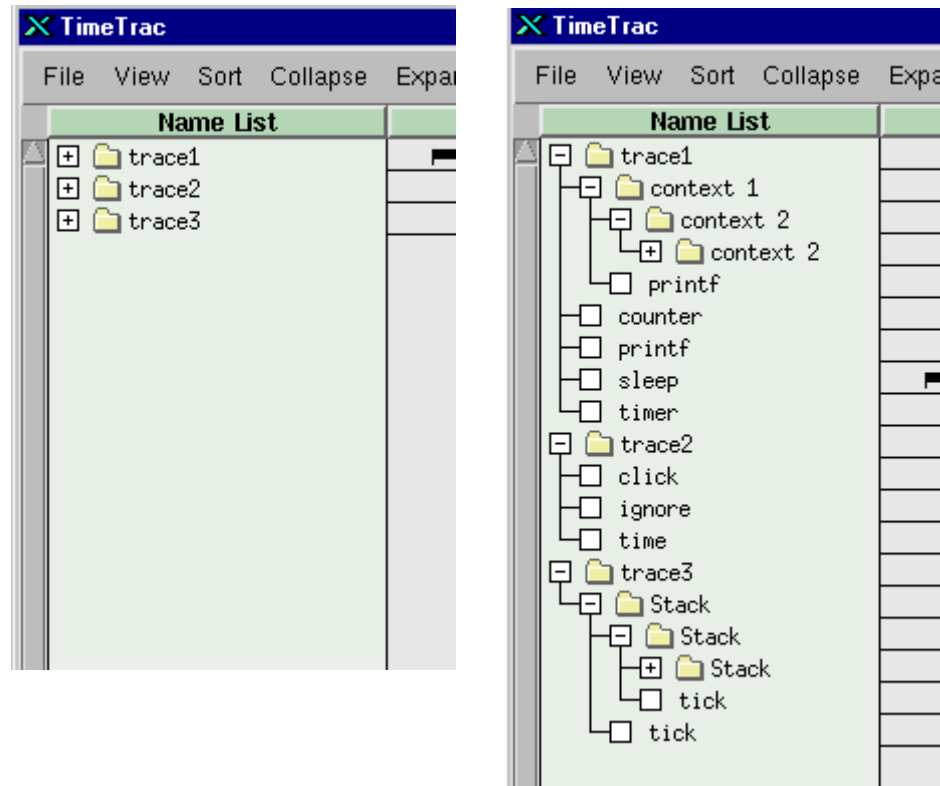
*Figure 23 Name List - Expand All Option*

The Name List includes a scroll bar along its left-hand side to scroll up and down the list. There is also a scroll bar beneath the Name List to scroll to the left or right.



**Level 1**  
**Level 1,2**  
**Level 1,2,3**

These menu selections control the number of levels of the hierarchy that display in the Name List. For example, clicking **Level 1,2,3** displays levels 1, 2, and 3, as shown in [Figure 24](#) (right).



*Figure 24 Name List - Expand 1, 2, 3 Option*

## Specify

The **Specify** menu selection enables you to select the level of the hierarchy that displays in the Name List. Clicking **Specify** displays an input box into which you enter a number such that the specified number of levels are displayed in the Name List. For example, if a trace file contains 10 levels and you enter 5 into the input box, the first 5 levels are displayed.

# Events Window

The Events Window, located in the center of the TimeTrac viewer, displays rows of event occurrences, as shown in Figure 25. A time scale runs horizontally across the bottom of the window. The left end of the time scale has a box containing the time at the beginning of the time scale. The other end of the time scale has a box containing the amount of time from the beginning to the end of the visible time scale.

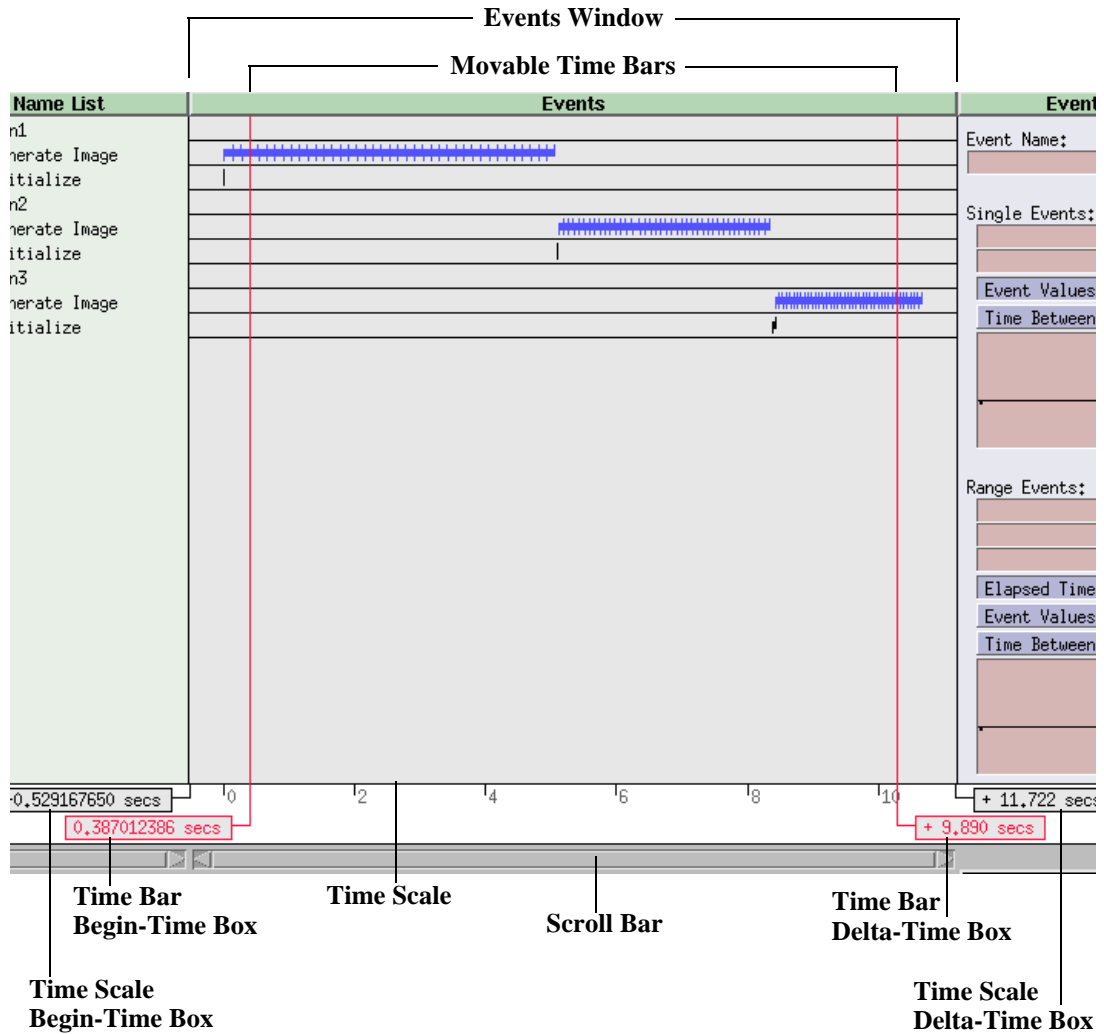


Figure 25 TimeTrac Viewer Events Window

If you are zoomed in on the event occurrences in the Events Window, you can use the scroll bar, located below the time scale, to move along the time scale. As you move the scroll bar, the time scale's begin-time box changes to reflect the movement of the time scale. However, note that the time scale's delta-time box does not change since the amount of time from the beginning to the end of the visible time scale does not change.

Depending on the number of times you zoom in on the event occurrences in the Events Window, the units in the delta-time box will change from seconds (secs) to milliseconds (ms) to microseconds ( $\mu$ s) and finally to nanoseconds (ns).

The Events Window also includes a pair of movable time bars. Each time bar has an associated box similar to the time scale. The box for the left-side time bar contains the value on the time scale where the time bar is located. The box for the right-side time bar contains the amount of time between the time bars.

## Mouse Controls

You can relocate the time bars in the Events Window by moving the mouse pointer over a time bar (the pointer changes to a "+" symbol), holding down the left mouse button, and dragging the time bar to the desired location.

A more convenient way to relocate the time bars is to move the mouse pointer to the desired location and click the middle mouse button to move the left-side time bar or the right mouse button to move the right-side time bar.



### Note

If you have reprogrammed your mouse buttons to work in reverse order, then the previous instructions for left and right mouse buttons will work in reverse order.

You can select a row of event occurrences by clicking in the row with the left mouse button. In addition, you can display information about a particular event by moving the mouse pointer over the event and pressing the Shift key and the left mouse button. The resulting event information is shown in [Figure 26](#).

Name: Initialize				
Time: 12.991759568 seconds				
Type	Firing	Time (seconds)	Value	Elapsed Time
start	1	12.971608600	0	101.481 ms
stop	1	13.073089800	0	

Figure 26 Event Information Window

The information indicates that the Initialize event is a range event whose start and stop events occurred at the times shown. The elapsed time of the event is also given.

The solid line between the start and stop statistics indicates where the mouse pointer is located in relation to the range event. In this case, the mouse pointer is between the start and stop event. The “Time:” value, located beneath “Name:”, is the location of the mouse pointer on the time scale.



### Note

Once the event information window displays, you can release the Shift key. The window will remain displayed.

If you move the mouse pointer to immediately before the start of the range event, the event information displays as shown in [Figure 27](#). Note that the solid line appears before the events.

Name: Initialize				
Time: 12.963842256 seconds				
Type	Firing	Time (seconds)	Value	Elapsed Time
start	1	12.971608600	0	101.481 ms
stop	1	13.073089800	0	

*Figure 27 Mouse Pointer Before Range Events*

If you move the mouse pointer to immediately after the end of the range event, the event information displays as shown in [Figure 28](#). Note that the solid line appears after the events.

Name: Initialize				
Time: 13.082490800 seconds				
Type	Firing	Time (seconds)	Value	Elapsed Time
start	1	12.971608600	0	101.481 ms
stop	1	13.073089800	0	

*Figure 28 Mouse Pointer After Range Event*

## Statistics Area

The Statistics Area is located on the right-hand side of the TimeTrac viewer. Depending on the setting, it displays the event statistics for the currently selected row of event occurrences in the Events Window or the percent of CPU usage for each row of event occurrences in the Events Window, as shown in [Figure 29](#).



### Note

Only the event occurrences that are visible in the Events Window are reflected in the CPU usage percentages or the event statistics displayed. For CPU usage percentages or event statistics of the entire row of event occurrences, zoom out so that the entire row appears in the display.

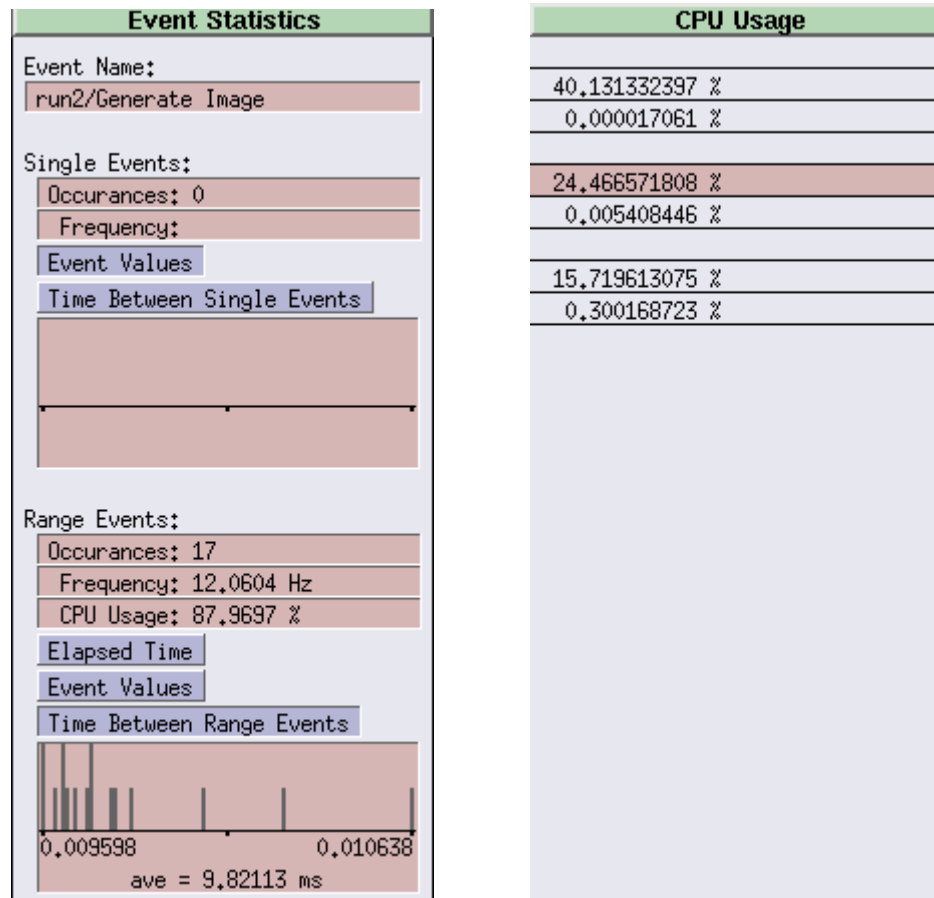


Figure 29 Statistics Area of the TimeTrac Viewer

When Event Statistics are displayed, the name of the selected event row (for example, Generate Image) is listed. Depending on the type of event, its statistics are listed under “Single Events” or “Range Events.”

The event statistics include the:

- Number of event occurrences visible in the Events Window
- Frequency of the event occurrences
- CPU usage of the event occurrences (range events only)
- Elapsed time of the event occurrences (range events only)
- Values of the event occurrences
- Time between event occurrences including an average time for range events

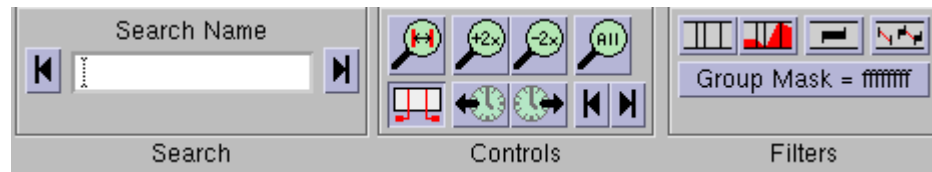
The statistics area for “Single Events:” includes a graphical display. By clicking “Event Values” or “Time Between Single Events,” you can display the associated information for the selected event occurrences.

The statistics area for “Range Events:” includes a graphical display. By clicking “Elapsed Time,” “Event Values,” or “Time Between Single Events,” you can display the associated information for the selected event occurrences. As shown in [Figure 29](#), the graphical display for “Range Events:” displays the time between range events for “run2/Generate Image.”

---

## Search/Filters/Controls Area

The Search/Controls/Filters area, which is located beneath the Name List and Events Window, is shown in [Figure 30](#). From this area, you can search for a particular event in the Name List or control the Events Window.



*Figure 30 Search/Controls/Filters Area of the TimeTrac Viewer*

## Search

The Search Name text box enables you to search for a specific name in the Name List. Enter a name in the text box and click one of the arrows on either side of the box to initiate the search.

Clicking the arrow to the left of the text box searches the Name List “upward” from the currently selected name to the top of the Name List. Clicking the arrow to the right of the text box searches the Name List “downward” from the currently selected name to the bottom of the Name List.

You can enter the full name or a partial string. For a partial string, the search will find the first occurrence of the string.

A beep will sound if the string was not found in the direction indicated.

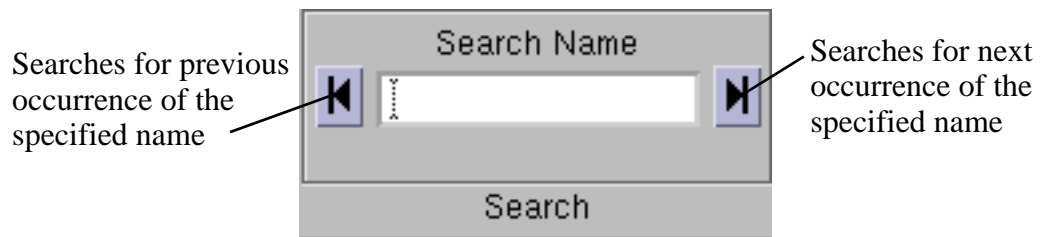


Figure 31 Search Area of the TimeTrac Viewer

## Controls

The Controls area, shown in [Figure 32](#), enables you to manage the events in the Events Window. The features of the Controls area are described in [Table 5](#).

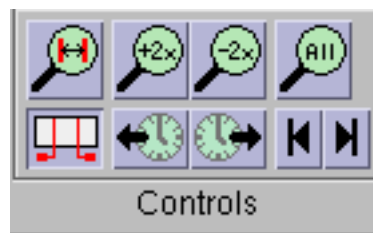




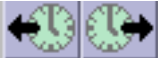



Figure 32 Controls Area of the TimeTrac Viewer

Option	Description
	<p>Clicking this icon expands the area within the time bars. This is a convenient way to zoom in on a specific event occurrence or multiple event occurrences.</p> <p>This icon is not available if the time bars are hidden.</p>
	<p>Clicking these icons zoom in (+2x) or zoom out (-2x) on the event occurrences currently displayed in the Events Window.</p> <p>These icons are not available if you have zoomed to the minimum or maximum level.</p>
	<p>Clicking this icon zooms out to display all of the event occurrences for each row currently visible in the Events Window.</p> <p>This icon is not available if you have zoomed out completely.</p>
	<p>Clicking this icon toggles the time bars in the Events Window so that they are either visible or hidden.</p>
	<p>Clicking these icons shifts the currently selected row (and any rows under this level) of event occurrences to the left or right by the amount of time between the time bars.</p> <p>These icons are not available if the time bars are hidden.</p>
	<p>Clicking these icons displays the next or previous event occurrence for the selected event row in the center of the Events Window.</p> <p>For the selected event row, clicking the left arrow centers the first event occurrence to the left in the Events Window. Clicking the right arrow performs the same operation for the first event occurrence to the right.</p> <p>This is a convenient way of displaying an event occurrence without having to zoom out and then zoom in.</p> <p>A beep will sound if you have scrolled to the minimum or maximum limit.</p> <p>These icons are not available if you have not selected a row of event occurrences.</p>

**Table 5** Features of the Controls Area








## Filters

The Filters area, shown in [Figure 33](#), enables you to control how events are displayed in the Events Window. The features of the Filters area are described in [Table 6](#).



*Figure 33 Filters Area of the TimeTrac Viewer*

Option	Description
	Clicking this icon toggles whether single events are displayed or hidden in the Events Window. Normally, single events are displayed.
	Clicking this icon toggles among the following values displayed in the Events Window: <ul style="list-style-type: none"> <li>- the interpolated values associated with single events are displayed</li> <li>- the actual values associated with single events are displayed</li> <li>- all values are hidden</li> </ul> This icon is not available if single events are hidden.
	Clicking this icon toggles whether range events are displayed or hidden in the Events Window. Normally, range events are displayed.
	Clicking this icon toggles whether any defined causal events are displayed or hidden in the Events Window. Normally, causal events are displayed. <p>This icon is not available if both single and range events are hidden.</p>
	Clicking this icon displays a TimeTrac input box that you can use to change the value of the Group Mask. Setting a bit to 0 in the group mask causes all events associated with that group ID to be hidden. To make all events visible, set the group mask to ffffffff.

**Table 6** *Features of the Filters Area*

---

## Information Bar

The Information Bar, located beneath the Search/Controls/Filters area, is shown in [Figure 34](#).



*Figure 34 TimeTrac Viewer Information Bar*

The Information Bar includes the following:

- Path  
This section displays the full pathname of the currently load trace file(s).
- Events  
This section displays the total number of events in the currently loaded trace file(s).
- Events Modified  
This section indicates whether you have modified the currently loaded trace file(s).

# 5

## Using the TimeTrac Viewer

---

The TimeTrac viewer can be very useful in helping you analyze your programs to:

- Determine the efficiency and performance of an algorithm
- Ensure that specific program operations occur in the correct order

This chapter provides screen captures showing the use of the TimeTrac viewer for these types of analysis. TimeTrac example files are included on the distribution CD.

Each example includes a readme file and makefile as well as the source file(s) and the resulting trace file(s), which are viewed using the TimeTrac viewer.

---

### Analyzing Algorithms

One simple way of evaluating the performance of algorithms in your program is by viewing their CPU usage in the TimeTrac viewer.

1. Copy the example files to a local directory:

```
% mkdir timetrac
% cd time_trac
% cp -r /usr/sky/examples/time_trac/grouping_algorithms .
```

2. Invoke the viewer by typing:

```
% TimeTrac
```

[Figure 35](#) shows the resulting TimeTrac display.

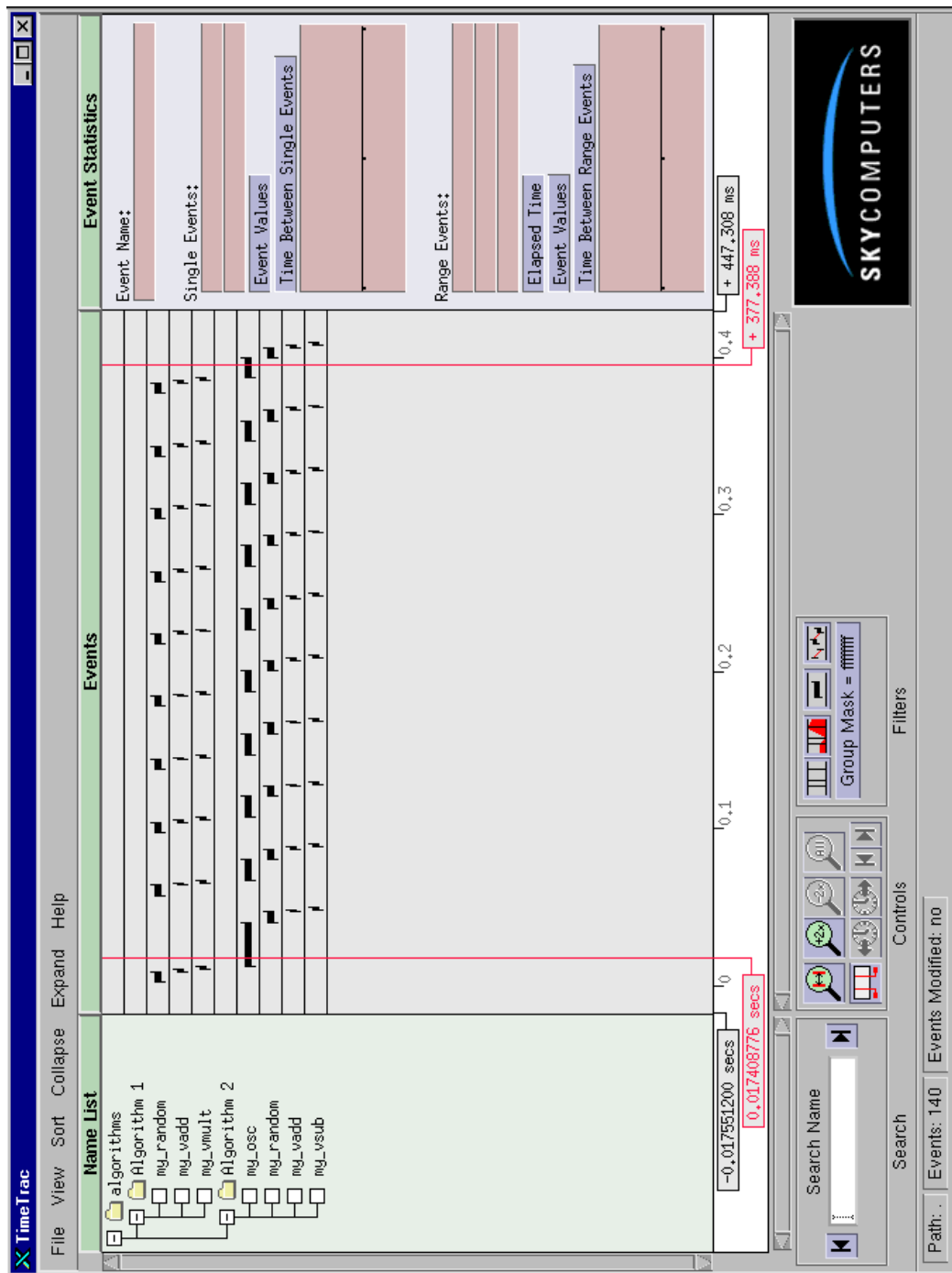


Figure 35 TimeTrac Viewer

- From the menu bar, click **Collapse > Level 2 and Greater**. Figure 36 shows the resulting Name List and Events Window.

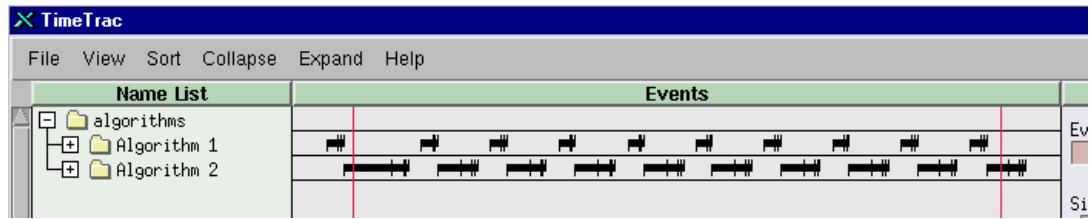


Figure 36 TimeTrac Viewer - Collapse > Level 2 or Greater

- From the menu bar, click **View > CPU Usage**. Figure 37 shows the resulting Name List and CPU Usage.

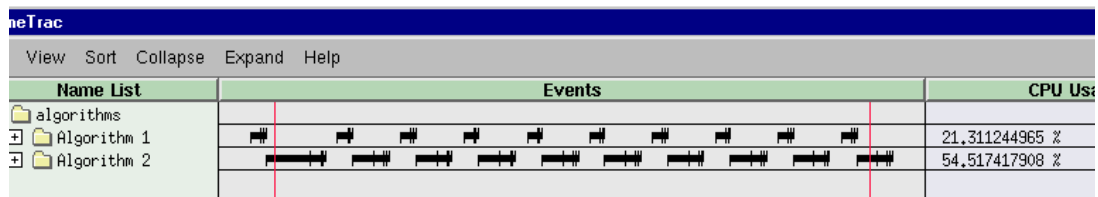


Figure 37 TimeTrac Viewer - View > CPU Usage

Figure 37 illustrates that Algorithm 2 has a significantly higher percentage of CPU usage than does Algorithm 1. Therefore, Algorithm 2 may be where you focus your attention to improve the performance of your program.

Another way of evaluating the performance of an algorithm is to run the test several times to see if successive changes to the algorithm have improved its performance. The following example shows the results of optimizing two components of an image processing algorithm. The test was run three times. Each time, the results were saved to a separate trace file so that the tests could be compared.

To compare test results:


1. Copy the example files to a local directory:

```
% mkdir timetrac
% cd time_trac
% cp -r /usr/sky/examples/time_trac/matrix .
```

2. Invoke the viewer by typing:

```
% TimeTrac
```

[Figure 38](#) shows the resulting TimeTrac display. The test was run three times (run1, run2, and run3) and was modified after the first and second run. Aligning the data will make it easier to compare.

3. First, align the data for run2. Move the mouse pointer to the beginning of the data for run1 and click the middle mouse button to move the left-side time bar into place.
4. Move the mouse pointer to the beginning of the data for run2 and click the right mouse button to move the right-side time bar into place.
5. In the Events Window, move the mouse pointer to the run2 row and click left mouse button. This action highlights the run2 row and selects the row and all of the rows associated with run2.
6. In the Controls area, beneath the Events Window, click  .
7. This option aligns the data for run1 and run2.
8. Next, align the data for run3. Move the mouse pointer to the beginning of the data for run3 and click the right mouse button to move the right-side time bar into place.
9. In the Events Window, move the mouse pointer to the run3 row and click left mouse button. This action highlights the run3 row and selects the row and all of the rows associated with run3.

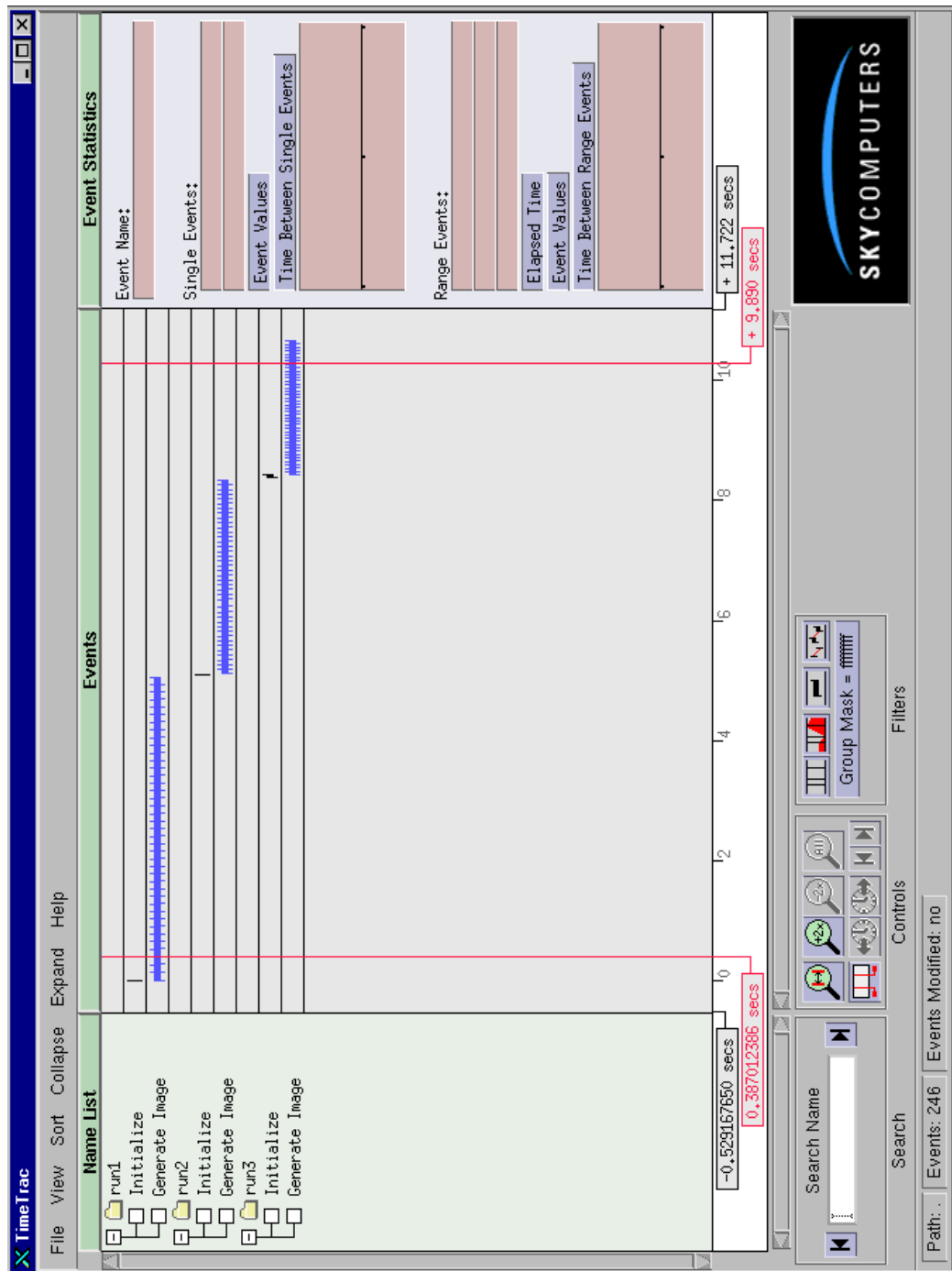



Figure 38 TimeTrac Viewer

10. In the Controls area, beneath the Events Window, click  .

This option aligns the data for run3 with the data for run1 and run2, as shown in [Figure 39](#).

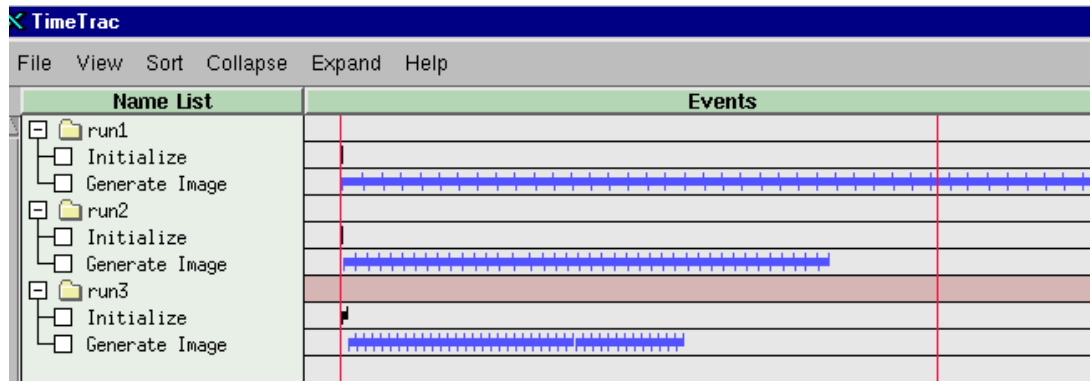


Figure 39 Aligned Data From Test Runs

After aligning the data, it is easy to see that the modifications made after the first and second test runs have significantly improved the performance of the algorithm.

11. If the Statistics Area is not displaying event statistics, click **View > Event Statistics** from the menu bar.
12. Click anywhere in the “Generate Image” row for run1 to select the data.  
The Statistics Area displays the event information for the Generate Image event occurrences.
13. In the Statistics Area, click “Elapsed Time” under Range Events.  
The average elapsed time for the Generate Image event occurrences displays in the graphical display box (approximately 117 ms).
14. Select the Generate Image row for run2 and display its average elapsed time. Do the same for the Generate Image row for run3.  
Note the decrease in the average elapsed time (approximately 71 ms and 46 ms respectively).
15. Move the mouse pointer over the run1 Initialize event. Press the Shift key and hold down the left mouse button to display the event information window for the event. (When the event information window displays, you can release the Shift key. The window remains displayed as long as you hold down the left mouse button.)  
Note that the elapsed time of the initialization event is 2  $\mu$ s.
16. Move the mouse pointer over the run2 Initialize event to display its elapsed time. Do the same for the run3 Initialize event. Note their elapsed times (634  $\mu$ s and 35 ms respectively).  
Notice that the modifications to the algorithm increased its initialization time; however, this is offset by the improvements to the algorithm’s performance.



---

# Verifying the Order of Program Operations

You can use the TimeTrac viewer to verify that operations within a program occur in the correct order. The following is an example of synchronizing multiprocessor operations.

As described in its readme file, the dining philosophers example involves four philosophers seated at a round table. They alternate between eating, thinking, and being hungry; however, no two adjacent philosophers may dine at the same time. Use the TimeTrac viewer to determine if the test results show two adjacent philosophers dining at the same time.

To view the test results:

1. Copy the example files to a local directory:

```
% mkdir time_trac
% cd time_trac
% cp -r /usr/sky/examples/time_trac/dining_philosophers .
```

2. Invoke the viewer by typing:

```
% TimeTrac
```

[Figure 40](#) shows the resulting TimeTrac display. This example is a split application that creates a trace file for the host program and for each of the philosopher's actions.

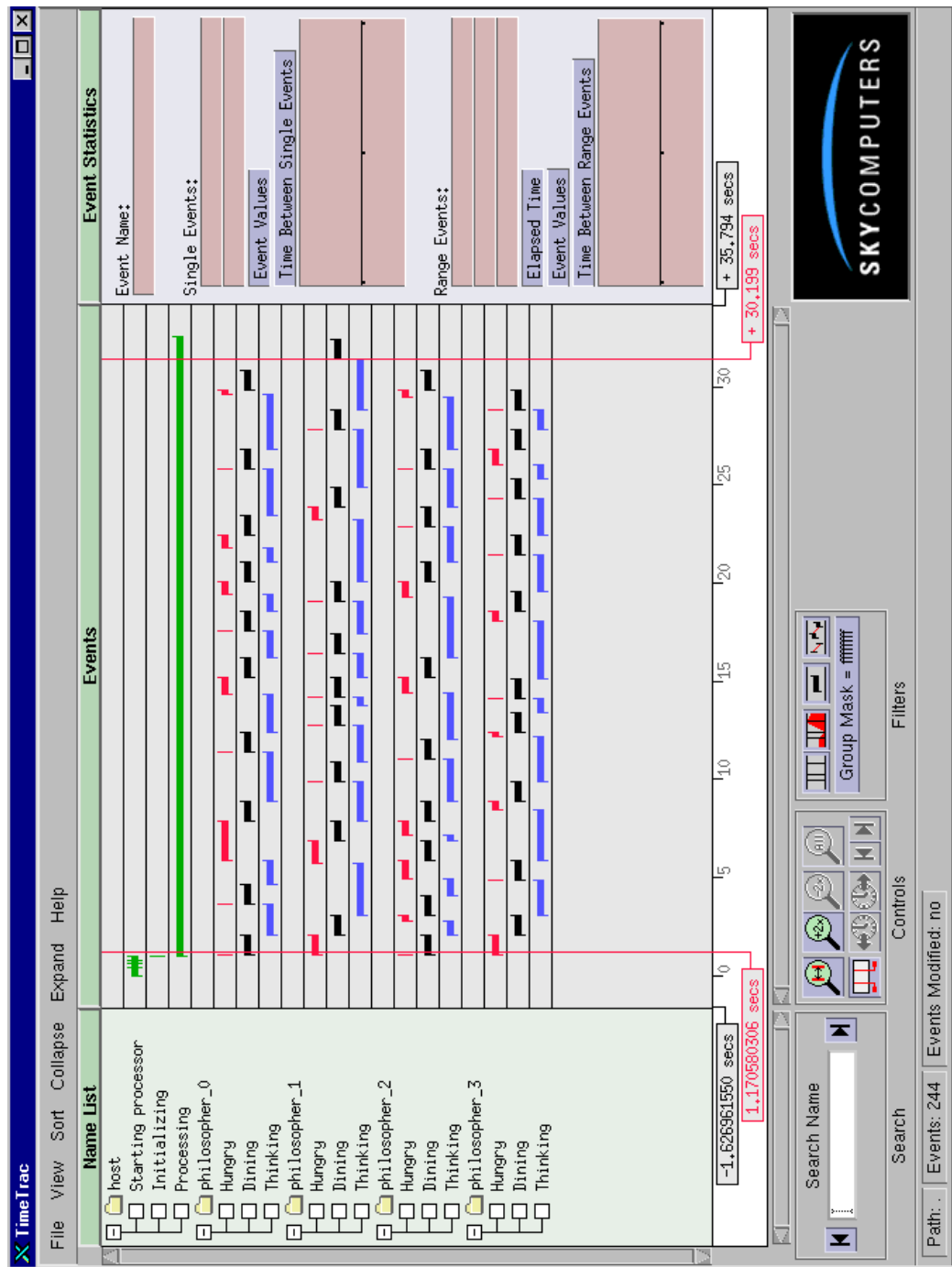


Figure 40 TimeTrac Viewer - Dining Philosophers Example

- From the menu bar, click **View > Trace File Name at Tail of Tree**. As shown in [Figure 41](#), this menu option organizes the dining events per philosopher.

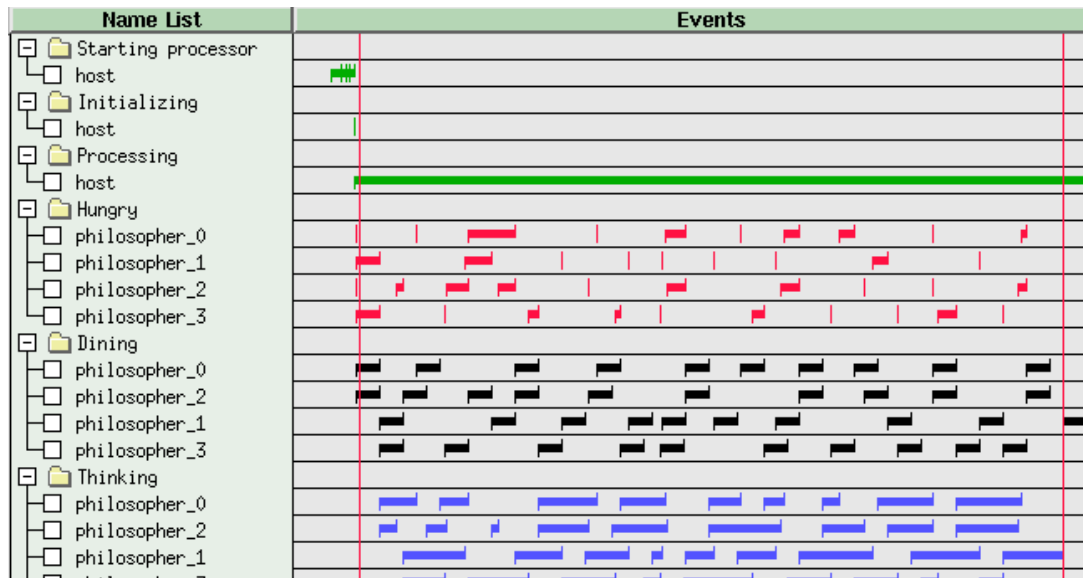


Figure 41 TimeTrac Viewer - View Trace File Name at Tail of Tree

- From the menu bar, click **Sort > Sort By Increasing Name**. As shown in Figure 42, this menu option places the philosophers in alphanumerical order so that you can check whether adjacent philosophers are dining at the same time.

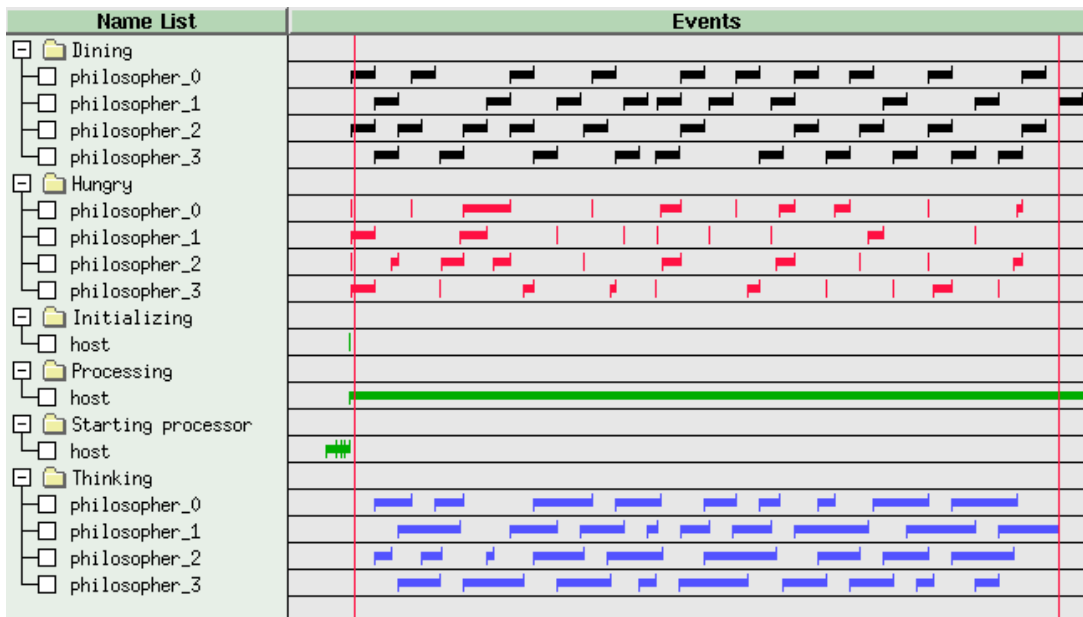



Figure 42 TimeTrac Viewer - Sort by Increasing Name

5. In the Controls area, beneath the Events Window, click  .

This option zooms into the Events Window.

6. Use the scroll bar located beneath the Events Window time scale to scroll through the dining events to verify that no two adjacent events occur at the same time.

# Index

---

## C

- Collapse Menu 53
  - collapsing by specified level 54
  - collapsing selected levels 54
- Contexts 13
  - nested 15

## E

- Events
  - contexts 13
    - nested 15
  - recording 12
    - group id 18
    - guidelines 12
  - registering 10
    - firing index 11
    - selecting a color 11
    - selecting event name 10
    - specifying a group id 11
- Events Window 58
  - mouse controls 59
    - displaying event information 59
- Expand Menu 55
  - expanding all levels 55
  - expanding by specified level 57
  - expanding selected levels 57

## F

- File Menu 41
  - exiting 46
  - loading events 41
  - printing 45
  - reloading event files 42
  - saving events 42
  - startup properties 42
- Firing Index 11

## G

- Group ID 11, 13, 18

## I

- Information Bar 66

## M

- Menu Bar 41
  - collapse menu 53
    - collapsing by specified level 54
    - collapsing selected levels 54
  - expand menu 55
    - expanding all levels 55
    - expanding by specified level 57
    - expanding selected levels 57

- file menu 41
  - exiting 46
  - loading events 41
  - printing 45
  - reloading event files 42
  - saving events 42
  - startup properties 42
- sort menu 49
  - sorting by CPU usage 50
  - sorting by name 52
  - sorting by time 49
- view menu 46
  - changing point size 48
  - displaying context names 48
  - displaying statistics 46

**R**

Range Event 10

**S**

- Search/Controls/Filters Area 62
  - control tools 63
  - filter tools 65
  - searching Name List 63
- Single Event 10
- Sort Menu 49
  - sorting by CPU usage 50
  - sorting by name 52
  - sorting by time 49
- Statistics Area 61

**T**

- TimeTrac
  - API ??-35
  - compiling and linking 18
  - components 9
  - events
    - contexts 13
    - range event 10
    - recording 12
    - registering 10
    - single event 10
  - overview 9
  - running on remote system 18

- viewer
  - analyzing algorithms 67
  - command line options 38
  - Events Window 58
  - Information Bar 66
  - menu bar 41
  - Name List 38
  - Search/Controls/Filters Area 62
  - Statistics Area 61
  - verifying program order 73

**V**

- View Menu 46
  - changing point size 48
  - displaying context names 48
  - displaying statistics 46