ATOMiC

ATOMiC
User Guide

May 1995

Product Version:	1.0
Operating System and Version:	OSF/1 3.2

This guide describes how to use the ATOMiC performance workbench. ATOMiC provides an integrated framework as a front-end to the ATOM performance-measurement tools.

Digital Equipment Corporation
Maynard, Massachusetts

Contents

B Adding a New Tool

Glossary

Figures

About This Guide

The ATOMiC tool provides a graphical user interface as a front-end to the ATOM performance-measurement tools. With ATOMiC, you can use ATOM tools to monitor the behavior of your applications and keep track of the different sets of results for different runs of the application. ATOM is a system for building analysis tools to monitor application performance. A number of pre-defined tools are supplied with ATOM which check various aspects of your application. For example, the **iprof** tool measures the number of instructions in each function, and the number of times each function is called. The **gprof** tool produces a call-graph of the functions in the application.

Audience

This guide is for programmers who use ATOM to improve the performance of Alpha-based OSF/1 applications.

Organization

This guide is divided into three chapters, two appendixes, and a glossary, as follows:

- Chapter 1 provides an overview of ATOMiC, and a quick tutorial introduction to using it.
- Chapter 2 describes each of the tools registered by default with ATOMiC, and explains the output generated by them.
- Chapter 3 describes all the windows and menus in the system.
- Appendix A describes how to install the ATOMiC tool.
- Appendix B describes how to add a new tool to ATOMiC.
- The Glossary contains a definition of terms.

Associated Documents

- *ATOM User Manual*
  This manual is available online in Postscript form in /usr/lib/atom/doc/user.ps.
- *ATOM Reference Manual*
  This manual is available online in Postscript form in /usr/lib/atom/doc/ref.ps.
- *Third Degree User Manual*
  This manual is available online in Postscript form in /usr/lib/atom/doc/3rd.ps.
- The *atom(1)* and *atomtools(5)* man pages.

Contact Address

Comments and bug reports should be sent to the following address:

Technical Computing Group,
Digital Equipment Corporation,
Galway,
IRELAND.
Email: atomic@ilo.dec.com

Conventions

The left, middle, and right mouse buttons are called MB1, MB2, and MB3 respectively.

Italics are used to denote function names or UNIX manual page references, for example: the *malloc(2)* system call.

Bold type is used to denote the first occurrence of words which appear in the Glossary.

Bold type in bullet lists indicate menu options.

User input is indented and in a Courier font. Bold-faced text indicates characters to be typed by the user.

Example output and source code is in a Courier font.

Menu paths are denoted using an arrow (→) symbol, for example: File→Exit.

Prerequisites

The ATOMiC tool requires the following:

- DEC OSF/1 Version 2.0 or higher
- ATOM Program Analysis Tools, Version 1.0 or higher. Note that ATOM Version 2.0 or higher is not supported in this version of ATOMiC.
- ATOM Add-on toolkit, Version 1.0
- The PATH environment variable must be set to include the directories where the *xterm* and *sh* programs are located.

Restrictions

ATOMiC requires that your application contain **relocation information** and use **non-shared libraries**.

ATOMiC is configured only with ATOM-based tools by default, although you can add other tools.

The default viewer provided with ATOMiC supports only plain-text output files. If you add new tools that output non-text files, you must also add a **Tk/Tcl** utility program to display the output.

Some ATOM tools produce multiple output files if the applications uses the *fork(2)* system call and the ATOMUNIQUE environment variable. The default tool viewers provided with ATOMiC do not support the viewing of multiple output files.

Introduction 1

This chapter provides an overview of the ATOMiC performance workbench, and includes a short tutorial on how to use it.

1.1 Overview

The goal of ATOMiC is to speed up the performance-improvement phase of software development or porting. This phase involves the following tasks:

- Measurement of performance according to various yardsticks
- Identification of critical code
- Modification of that code to test alternative implementations
- Further measurement and modification, until performance reaches the required level

ATOMiC provides an integrated framework to support these activities in three main areas:

- **File Management**
  ATOMiC stores all versions of the source code and the results of all measurement runs, so that changes are stored automatically. The developer can continually refine code, or try out different approaches, without having to repeatedly save files to other locations.
- **Compile/Link/Run Test Cycle**
  ATOMiC, through its integrated graphical framework, provides a fast, convenient method of selecting a tool, instrumenting an application, running the instrumented executable, and viewing the tool output.
- **Critical Code Identification**
  ATOMiC analyzes the results of the ATOM tool, sorts them by the important criteria such as processing time or number of instructions, and displays the output in a graphical and color-coded form. This enhances comprehensibility and speeds up the process of identifying the critical code.

In addition to these main benefits, ATOMiC also has a number of other features to assist the developer:

- **Multi-window Synchronized Display**
  ATOMiC uses multiple windows to simultaneously display source code, assembly code and results from different tools. ATOMiC also links the windows together so that clicking on a function name or source line in one window updates all windows to display that code. This allows easy inspection of sections of critical code from different aspects.
- **Customization**
  ATOMiC can be modified to suit your particular needs. If you create your own

performance-measurement tools, ATOM-based or otherwise, you can add them to the list of ATOMiC tools.

ATOMiC provides a default viewer to display text-based tool output files. If your tools generate non-text output, or you want to display text output graphically, you can add your own viewers to do so.

The source code for some of the ATOM tools is also available. You can create a new ATOM-based tool by copying and modifying the code for a tool that is similar to the new tool.

- **Automatic Logging**
  ATOMiC maintains a log file to record all versions created, tools run, and changes made. The log file is maintained across different ATOMiC sessions, which assists online review of previous work.
- **Other Useful Features**
  ATOMiC supports the use of RCS for version control, has an internal environment variable mechanism and uses a directory search path to search for files in multiple directories.

1.2 Tutorial

The sample program supplied with the *atom(1)* utility, /usr/lib/atom/confidence/conftest.c, is used as an example in this tutorial. The tutorial includes the following steps:

1. Compiling an application
2. Starting ATOMiC
3. Instrumenting and running an application
4. Viewing the output and source

1.2.1 Compiling an Application

Follow these steps to compile the application.

1. Create a working directory and copy the sample program /usr/lib/atom/confidence/conftest.c into it:

```
% mkdir work
% cd work
% cp /usr/lib/atom/confidence/conftest.c .
```

2. Compile the application into a relocatable non-shared image:

```
% cc -Wl,-r -non_shared conftest.c -o conftest
```

The **-Wl,-r** flag indicates to the linker to generate relocatable code, and the **-non_shared** flag indicates that libraries should be statically linked into the executable.

Note that the resulting image is no longer an executable binary, therefore it will not run directly. It is used as the input file for the ATOM tools.

3. Start ATOMiC by typing the following command:

```
% atomic conftest &
```

1.2.2 Starting ATOMiC

When ATOMiC starts, it prompts you to choose a location in which to place its working files. Click on the ‘‘Create’’ button to use the current directory.

ATOMiC also displays a window asking whether you wish to specify an RCS directory. Click on the ‘‘Don’t use RCS’’ button to ignore this for the moment.

At this point, the ATOMiC Session window appears. This window, shown in Figure 1-1, contains a menu bar and the following **panes**:

- Application - user-specified information on the current application
- Tool Status - current status of ATOMiC
- Log - a scrolling log of user actions

Figure 1-1: Session Window

1.2.3 Instrumenting and Running the Application

In the Session window, the Tools menu contains a complete list of the ATOM tools available to analyze your application. Choose Tools→gprof... to build a graphical **call-graph** of your application. The dialog box shown in Figure 1-2 is displayed.

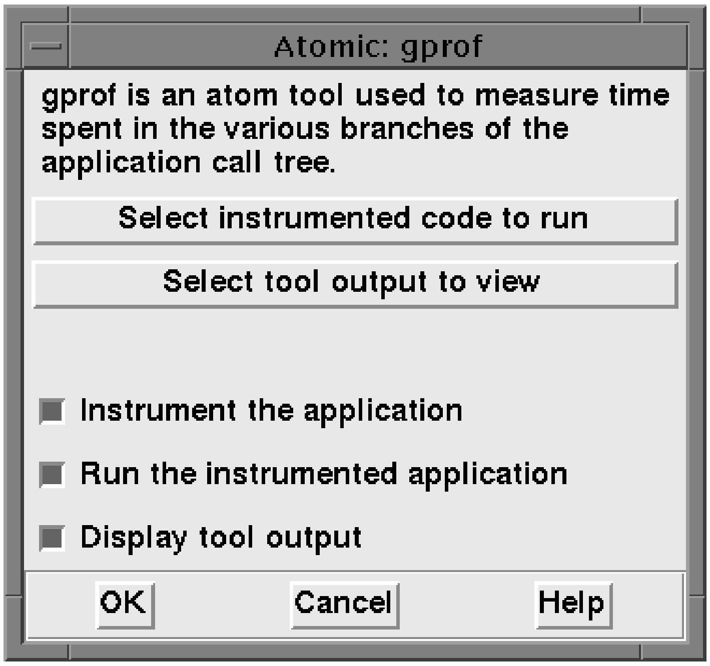

Figure 1-2: Tool Dialog Box

This dialog box displays two sets of buttons. The push buttons allow you to select an existing instrumented application or tool output file. The toggle buttons control the instrumentation, running, and display of output from an application. By default, the first time any tool is called, all three toggles are set. This should be the case in the current example.

Click on the ‘‘OK’’ button to instrument the relocatable image, run the instrumented executable, and display the gprof tool output.

Since this is the first run, ATOMiC displays a dialog box which asks you to provide a description for the log file. Click on ‘‘OK’’ to accept the current description.

In the Session window, the Status field shows the current stage of processing, and the Log scrolling window shows the details of which files have been created and where. By default, all the working files that ATOMiC creates are placed in a subdirectory of the current directory called atomic.

ATOMiC opens a temporary window in which to hold the standard input and output of the application. When the application finishes, ATOMiC saves the output to a file and closes the window.

1.2.4 Viewing the Tool Output

When the application has finished running, ATOMiC displays a tool output window, as shown in Figure 1-3.

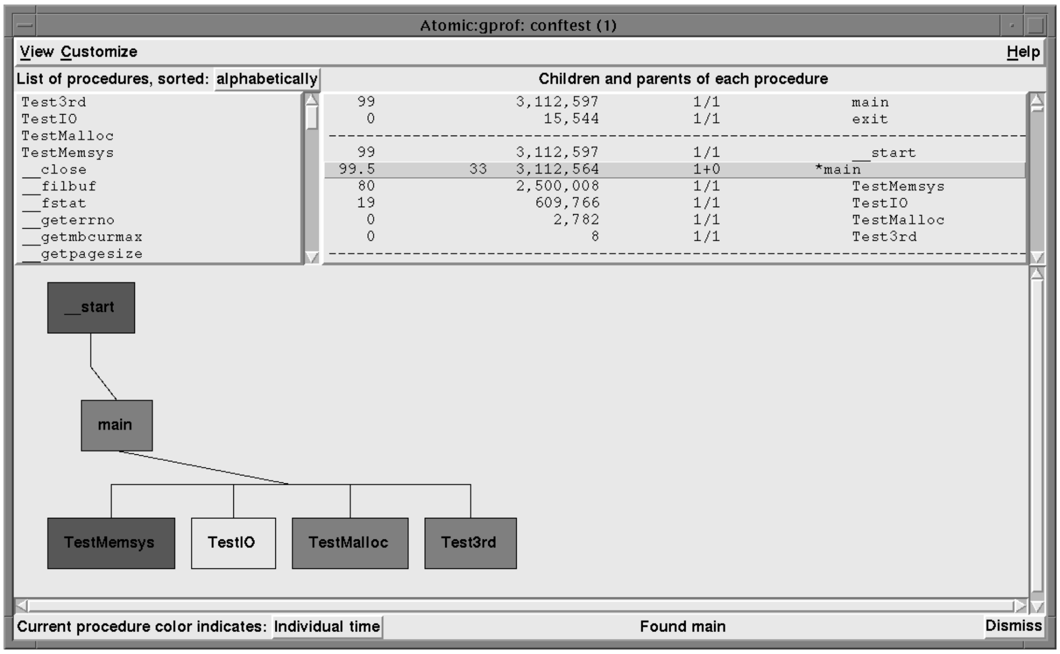

Figure 1-3: Tool Output Window for gprof

This window contains a menu bar and three display panes. The bottom pane of the window shows a function call-graph for the application. The root of the graph always begins at *__start* rather than *main()*. This is because in C programs, environment variables, command-line arguments, and statically initialized data are set up before *main()* is called. The call-graph is color-coded so that it is easy to see where the application is spending most time. Green indicates a small percentage of the total time, red a large percentage. Performance improvements can be made by reducing the amount of time spent in functions marked in colors close to red.

Click on the *main()* icon to display the functions called from within the *main()* function. The *main()* icon changes color to green, and the functions called from within *main()* appear as icons. In this case, the *TestMemsys()* function is the only red-colored icon, which indicates that most time is spent inside this function.

The thickness of the lines joining the icons in the graph is used to indicate both the calling hierarchy and the number of calls made. The thicker the line, the more calls to that function were made.

To display a particular function, double-click on its name in the alphabetical list of functions in the top left pane of the window.

The top right pane shows the text output file from which the gprof tool builds the call-graph. You can also select functions from this list to view in the call-graph.

By default, ATOMiC displays a partial tree containing three lines of icons. The middle line contains a single function, the top line contains any functions which call that function, and the bottom line contains any functions called by that function. To display a complete tree showing all functions in the application, choose the 'View→Full tree' menu item.

1.2.5 Viewing the Source Code

To view the source code of the application, choose the Output window View→Source menu item. This displays the Application window, shown in Figure 1-4, which initially contains a button bar across the top and a Source pane displaying code from the currently selected function.

Each time you click on a function icon in the call-graph or a function name from the list of procedures, that function is automatically displayed in the Application window.

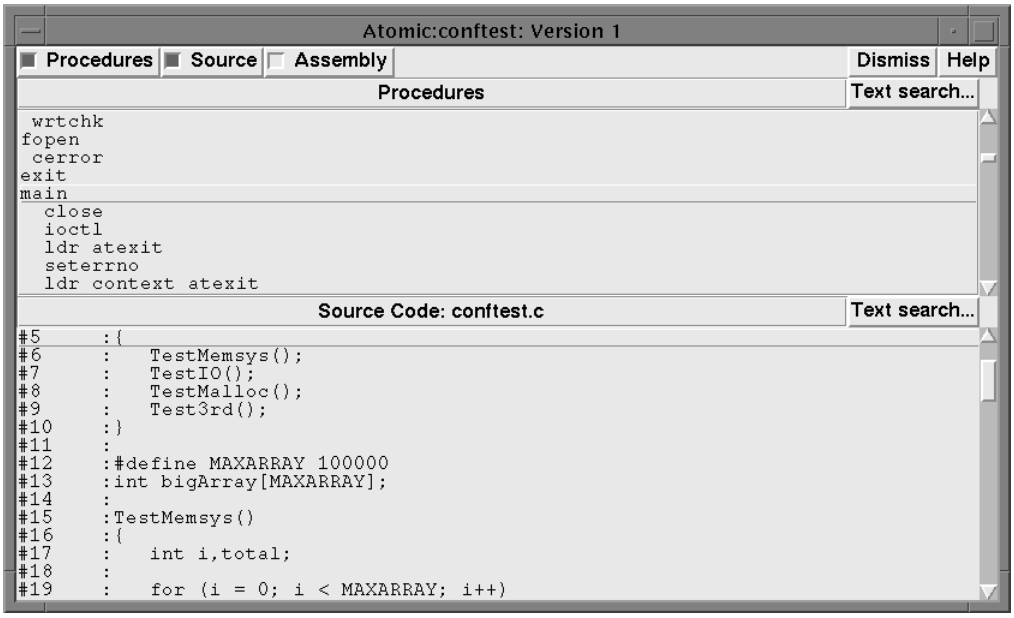

Figure 1-4: Application Window

The toggle buttons control which panes of the window are visible. The ‘‘Procedures’’ button displays a Procedures pane, which lists all the functions compiled into the application. The ‘‘Assembly’’ button displays an Assembly pane, which lists the assembly language code generated for each of the source code instructions.

Overview of Atom Tools 2

This chapter describes the ATOM tools registered by default with ATOMiC. The tools are not part of ATOMiC, and must be retrieved and installed separately. They are divided into the following types:

- Profile tools
- Cache and buffer simulation tools
- Input/Output tools
- Memory checking tools
- Instruction analysis tools

ATOMiC has no inherent knowledge of the tools, and it is possible to add your own ATOM-based or non-ATOM-based tools to ATOMiC.

2.1 Profile Tools

2.1.1 gprof — Function Call-Graph Profile

The gprof tool computes execution time information for each function in your application. The function call-graph is displayed in a color-coded graphical tree format which clearly shows where most time is spent.

The top left pane of the window, shown in Figure 2-1, contains a scrollable list of the functions in your application. You can control the sorting order of the list, which is initially sorted alphabetically. Click the ‘‘alphabetically’’ button to select another sort order. The following options can be selected.

- **alphabetically**
  The list is sorted in ascending ASCII order. This is the default.
- **by individual time**
  Each function is timed without including calls to other functions.
- **by cumulative time**
  The total time spent in each function including calls to other functions.

The top right pane of the window displays the text output of the gprof tool. This is the data from which the call-graph is built.

The bottom pane of the window displays a call-graph. The icon colors range from green through orange to red, indicating increasing percentages of total time spent in a function. In addition, the number of calls made by one function to another is indicated by the thickness of the line between them. Thicker lines represent more calls.

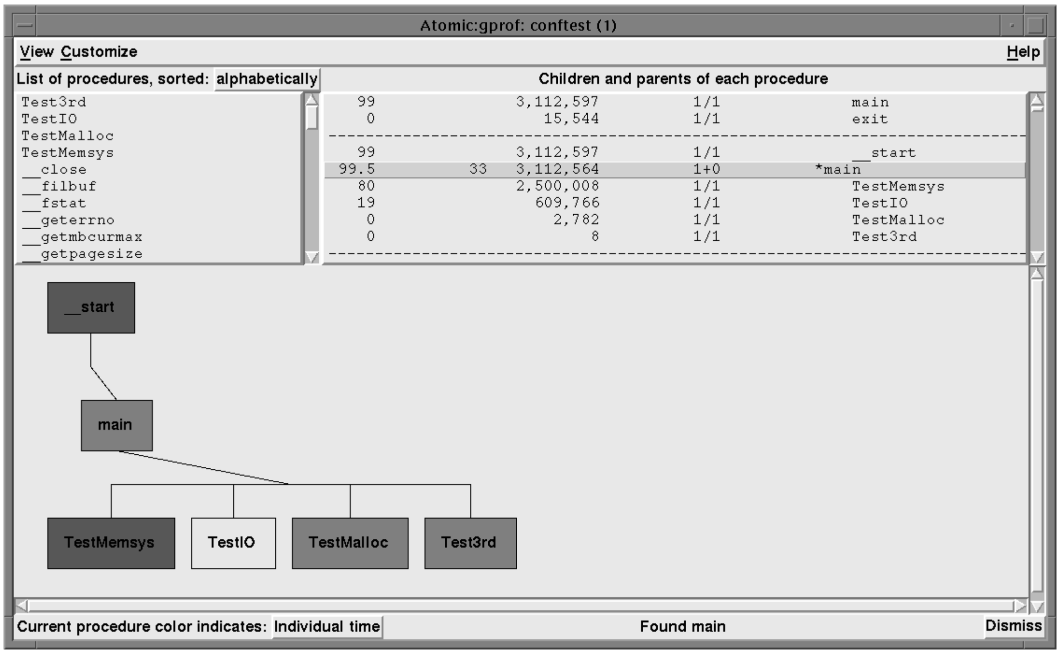

Figure 2-1: Graphical Function Call-graph

The three panes in the window are synchronized, so double-clicking on a function in either of the top panes displays the function in the bottom pane.

The View menu has the following items.

- **Partial tree**
  Display a partial tree showing only the currently selected function and its parents and children.
- **Full tree**
  Display a full tree showing all the functions in the application.
- **Source**
  Open the Application window.
- **Redraw full tree**
  Refresh the tree.
- **Zoom in**
  When a full tree is displayed, you can zoom in to see a smaller part of the tree in a larger size.
- **Zoom out**
  When a full tree is displayed, you can zoom out to see more of the tree.

The Customize menu has two items.

- **Set palette**
  Display the Palette Selector dialog box. This allows you to choose the colors used in the window. See Section 3.5.2 for further information.

- **Set bins**
  Display the Set Bins for Statistics dialog box. This allows you to choose the percentage range represented by each color. See Section 3.5.1 for further information.

2.1.2 inline — Function Call Hierarchy

The **inline** tool counts the number of function calls made by your application, and the functions from which they were made. The calling function is the parent. The output also displays the Program Counter address where the call is made.

The Output window displays a table with the following headings:

- PC – Program Counter
- Parent – Parent function name
- Called – Child function name
- Number – Number of times the parent called the child

In the following example, the *TestIO()* function calls *fgets()* a total of 20 times at two locations:

PC	Parent	Called	Number
0x01200056ec	__start	main	1
0x0120005728	__start	exit	1
0x0120005774	main	TestMemsys	1
0x0120005784	main	TestIO	1
0x0120005794	main	TestMalloc	1
0x01200057a4	main	Test3rd	1
0x0120005864	TestIO	fopen	1
0x01200058bc	TestIO	fprintf	19
0x01200058e4	TestIO	fclose	1
0x0120005904	TestIO	fopen	1
0x0120005950	TestIO	fgets	1
0x0120005970	TestIO	fgets	19
0x0120005988	TestIO	fclose	1
0x01200059a0	TestIO	__unlink	1
0x01200059e0	TestMalloc	malloc	10
0x0120005bd8	exit	__ldr_atexit	1

In the source code *fgets()* is called only once, on line 48 (see the source listing which follows). This indicates that the compiler has translated that one function call into two separate calls in the assembly language.

```
31  TestIO()
32  {
33     int j;
34     char line[STRLEN];
35     FILE *fileOut,*fileIn;
36
37     fileOut = fopen("fileOut","w");
38     assert(fileOut != NULL);
39     for (j = 1; j < 20; j++)
40       {
41         fprintf(fileOut,"%d: Helloworld\n",j);
42       }
43     fclose(fileOut);
44
45     fileIn = fopen("fileOut","r");
46     assert(fileIn != NULL);
47
```

```
    while (fgets(line, STRLEN, fileIn) != NULL)
      {
      }
    fclose(fileIn);

    unlink("fileOut");
}
```

2.1.3 liprof — Basic Block Instruction Profile

The **liprof** tool computes instruction profile information for each basic block in your application. A basic block is a section of code where instructions are guaranteed to execute sequentially and which can only be entered at the beginning and exited at the end.

The Output window displays a table with the following headings:

- Line – Source code line number
- PC – Program Counter
- Instructions – Number of instructions
- Count – Count of execution iterations
- Total – Total instruction execution count

The following example shows the output for the function *TestMalloc()* in conftest.c. In the function, the third basic block starts at line 63, has 8 instructions and was executed 10 times for a total of 80 instructions.

```
----------------------------------------------------------------------
TestMalloc: conftest.c
----------------------------------------------------------------------
       Line          PC Instructions        Count         Total
         57 0x120005af0            6            1             6
         63 0x120005b08            3           10            30
         63 0x120005b14            8           10            80
         65 0x120005b34            3            1             3
```

The corresponding source code for the function is as follows:

```
56  TestMalloc()
57  {
58      int i;
59      char *ptr;
60
61      for (i = 1; i < 1000; i += 100)
62      {
63          ptr =  (char *) malloc(i);
64      }
65  }
```

Note that this tool identifies basic blocks at the assembly level, not the source code level, so even a single source code instruction might be split into a number of basic blocks.

To open the Application window, click on the ‘‘View source’’ button in the Output window. If the Application window Source or Assembly panes are open, you can double-click on a PC address or a function name in the Output window to scroll the source and assembly code to the appropriate location.

The Procedures pane of the Application window contains two extra columns when **liprof** is run. These columns display the third and fifth columns of the Output display. When you press MB3 in either of these columns, ATOMiC brings up a menu providing a number of further commands. These menu items are described in detail in Section 3.2.

2.1.4 pixie — Count Cycles Per Function

The **pixie** tool counts the number of cycles executed in each function of your application. The Output window displays a table with the following headings:

- cycles – Cycles
- %cycles – Percentage of total cycles
- cum % – Cumulative percentage of total cycles
- cycles/call – Cycles per call
- bytes/line – Bytes per line
- procedure – Function name
- (file) – File name

```
2327475 cycles

    cycles %cycles  cum %     cycles  bytes procedure (file)
                               /call  /line

   2300009   98.82  98.82    2300009     22 TestMemsys (conftest.c)
      8027    0.34  99.16        423     44 _doprnt
      3702    0.16  99.32        195      9 __memccpy
      2287    0.10  99.42         41      5 memcpy
      1900    0.08  99.50        100      7 strchr
      1874    0.08  99.58         99     29 fwrite
      1423    0.06  99.65         72     19 fgets
      1174    0.05  99.70         98     12 malloc
      1031    0.04  99.74        129     23 cartesian_alloc
       893    0.04  99.78         47     28 fprintf
```

The pixie tool supports several options, which you control using the environment variable PIXIE_ARGS. You can specify environment variables using the File→Environment menu item in the Session window. The tool reads the environment variable when you run the instrumented application. The pixie tool supports the following options, which must be separated by white space:

- **-pids**
  When this option is specified, the tool appends the process ID to the output file name. This is useful if your application calls the *fork(2)* function. A separate output file is created for each child process of your application.

- **-sigdump**
  When this option is specified, you must supply the name of a signal, such as SIGUSR1. This option is useful for applications that never terminate. The application terminates when the specified signal is sent, and it writes the output file for the tool. Your application must not reset the signal handler for this signal. Specify the signal by its name, not its number, for example: `-sigdump SIGUSR1`. You can send a signal to a process by using *ps(1)* to identify the

process by its process ID, and using *kill(1)* to send the signal.

```
% ps |grep pixie
12298 ttyp0    S        0:00.11 conftest.pixie
% kill -USR1 12298
```

- **-sbrk**
  When this option is specified, the tool allocates the dynamic memory it requires using the *sbrk(2)* system call. Normally the tool uses *mmap(2)*. The memory is required to store the execution count information.

2.2 Cache and Buffer Simulation Tools

The tools in this section simulate the behavior of processor caches and translation buffers during the running of your application. The models of Alpha-based systems have varying sizes of instruction and data caches and translation buffers. The average cost of servicing cache misses and writes also varies. The characteristics of many Alpha models are kept in the file /usr/lib/atom/tools/platform.h. The ATOM tools in this section use these characteristics to compute results for the simulation tools. If the machine type is not found, the value is computed based on the cycle time of the DEC3000/500 system.

2.2.1 cache — Simulate Data Cache

The **cache** tool simulates the use of the memory systems' data cache for your application, and counts the number of cache references and misses. The lower the miss ratio, the better the performance.

In the following example, the miss ratio is 0.122, or 12.22%:

```
References: 105234
Misses:     12863
Miss Rate:  0.122232
```

2.2.2 dtb — Simulate Data Translation Buffer

The **dtb** tool simulates the use of the memory systems' data address translation buffer for your application, and counts the number of hits and misses. The lower the miss ratio, the better the performance.

In the following example, only 114 of the 208101 DTB references missed, so the miss ratio is 0.0005, or 0.05%:

```
DTB hits:              207987
DTB Misses:               114
DTB references:        208101
Miss Rate:             0.0005
```

2.2.3 lmemsys — Simulate Memory System By Instruction

The **lmemsys** tool simulates the use of the data and instruction caches of the Alpha processor during the running of your application. It examines each boundary, load and store instruction in the instruction cache. For each instruction, it estimates the time spent handling cache events.

The Output window displays a table with the following headings:

- Line – Source code line number
- PC – Program Counter
- imisses – Instruction cache misses
- drmisses – Data cache read misses
- dwrites – Data cache writes
- bmisses – Board cache misses
- bvictims – Board cache victims (where a **dirty block** had to be written to main memory to make way for a new board cache line)
- total – Total time

In the following example, a store instruction was executed on line 21 and a load instruction on line 25. This can be seen by inspection of the Assembly code pane of the Application window. The store instruction required 1.0 ms of processing time for writing data, and 2.5 ms for board misses. The load instruction required 1.0 ms to process the data cache read miss.

```
-------------------------------------------------------------------------
TestMemsys: conftest.c
-------------------------------------------------------------------------
   Line           PC  imisses drmisses  dwrites  bmisses bvictims    total
                         (ms)     (ms)     (ms)     (ms)     (ms)     (ms)
     21 0x1200057e4       0.0      0.0      1.0      2.5      0.0      3.5
     25 0x120005820       0.0      1.0      0.0      0.0      0.0      1.0

  Other                   0.1      0.0      0.1      0.2      0.0      0.4

Summary                   0.1      1.1      1.1      2.7      0.0      5.0
```

If the environment variable MEMSYSCOUNT exists, the output shows absolute counts rather than times. You can declare environment variables using the File→Environment menu item in the Session window.

2.2.4 memsys — Simulate Memory System By Function

The **memsys** tool simulates the use of the data and instruction caches of the Alpha processor during the running of your application. For each function, it estimates the time spent handling cache events.

The Output window displays a table with the following headings:

- Function name
- imisses – Instruction cache misses
- drmisses – Data cache read misses
- dwrites – Data cache writes
- bmisses – Board cache misses
- bvictims – Board cache victims (where a dirty block had to be written to main memory to make way for a new board cache line)

• total – Total time

```
              imisses drmisses  dwrites  bmisses bvictims    total
                 (ms)     (ms)     (ms)     (ms)     (ms)     (ms)
  TestMemsys      0.0      1.0      1.0      2.5      0.0      4.6
     _doprnt      0.0      0.0      0.0      0.0      0.0      0.1

       Other      0.1      0.0      0.1      0.1      0.0      0.3

     Summary      0.1      1.1      1.1      2.7      0.0      5.0
```

The tool does not display details of functions which generate less than 100 microseconds of memory system activity.

2.2.5 tb — Simulate Translation Buffer

The **tb** tool simulates the use of the memory systems' data and instruction translation buffers for your application, counts the number of hits and misses, and estimates the time spent handling the misses. The following output is produced:

```
Procedure    ITB Misses  ITB time    DTB Misses  DTB Time  Total Time
                             (ms)                    (ms)        (ms)

    Other             8       0.0           114       0.0         0.0

    Total             8       0.0           114       0.0         0.0
```

2.3 Input/Output Tools

The tools in this section examine the input and output characteristics of your application. Some of the results give values for total **wall-clock** time or bytes per second data rates. The tools compute these values based on the parameter file /usr/lib/atomtools/platform.h. This file contains the processor cycle time and other information for a range of Alpha systems. If the file does not contain information on your platform, the value is computed based on the cycle time of the DEC3000/500 system.

2.3.1 io — Summary by I/O Operation

The **io** tool summarizes the total input and output of your application. The tool gathers summary statistics on the file accesses made by the application through functions such as *fopen()*, *fputs()* and *fgets()*. The output is divided into the following sections:

- Input/Output Summary
- Read Summary
- Read Address Alignment
- Write Summary
- Write Address Alignment

In the following example, the Input/Output summary shows that file descriptor 4 was opened twice and two reads, one write, and no seeks were done. A total of 16660 bytes were read or written. The following Read and Write summaries show that two reads of 8192 bytes and one write of 276 bytes were performed:

```
-----------------------------------------------------------------------
Input/Output Summary
-----------------------------------------------------------------------

 File Desc      Opens      Reads     Writes      Seeks         Bytes
         4          2          2          1          0         16660
-----------------------------------------------------------------------
Read Summary
-----------------------------------------------------------------------

         Size          Count          Bytes      Time(ms)          Rate
         8192              2          16384           0.5           0.0
-----------------------------------------------------------------------
Read Address Alignment
-----------------------------------------------------------------------

         Size           Quad           Long          Word          Byte
         8192              2              0             0             0
-----------------------------------------------------------------------
Write Summary
-----------------------------------------------------------------------

         Size          Count          Bytes      Time(ms)          Rate
          276              1            276           0.8           0.0
-----------------------------------------------------------------------
Write Address Alignment
-----------------------------------------------------------------------

         Size           Quad           Long          Word          Byte
          276              1              0             0             0
```

2.3.2 io2 — Summary by File Name

The io2 tool performs the same analysis as the io tool but displays its output in a different format. The statistics are summarized on a per-open-file basis so that they give a more detailed picture of the amount of data read from and written to individual files.

The following example shows the output of the io2 tool based on the same application used in the previous example. In the following example, the file fileOut was opened once for writing and twice for reading:

```
reads writes lseeks fcntls   read   write        read      write File
                           (bytes) (bytes)       (bps)      (bps)
    0      1      0      0      0     276        0.00 713894.94 fileOut
    2      0      0      0  16384       0 202743424.00      0.00 fileOut
    0      0      0      0      0       0        0.00      0.00 stdin
    0      0      0      0      0       0        0.00      0.00 stdout
    0      0      0      0      0       0        0.00      0.00 stderr
```

2.3.3 syscall — Summary of System Calls

The **syscall** tool counts the number of system calls made by your application and summarizes the results on a per-system-call basis. The Output window displays a table with the following headings:

- System Call – System Call name

- Calls – Number of calls made
- Time(ms) – Time spent on call
- context – Number of context switches made during call

In the following example, open was called twice and close five times. This is because the process was passed open file descriptors for standard input, output and error, but had to close them explicitly.

```
System Call      Calls          Time(ms)    context
       read          2              0.1          0
      write          1              0.4          0
      close          5              0.3          0
     unlink          1              0.7          0
     obreak          4              0.2          0
       open          2              0.9          0
      ioctl          2              0.0          0
getpagesize          1              0.0          0
      fstat          2              0.0          0

      Total                         2.7          1
```

2.4 Memory Checking Tools

The tools in this section examine the use of memory in your application.

2.4.1 3rd — Check Memory Leaks and Accesses

The **3rd** (Third Degree) tool performs runtime checking of the memory use of your application. The tool checks the use of *malloc(2)* and *free(2)* to ensure that there are no memory leaks. It also checks read accesses to memory to ensure that the memory is initialized.

In the following example, 3rd detects that uninitialized memory was read on line 72, and memory dynamically allocated on line 63 was never freed:

```
Reading uninitialized memory in stack frame of Test3rd at offset 20
  - Test3rd                                        conftest.c, line 72
  - main                                           conftest.c, line 9

_______________________________________________________________________
-----------------------------------------------------------------------
Searching for new leaks in heap before 1st call of _exit in parent process
We found 4510 bytes in 10 objects:

4510 bytes in 10 leaks (including 10 super leaks) created by malloc at
  - TestMalloc                                     conftest.c, line 63
  - main                                           conftest.c, line 8
```

The Output window Options button displays a dialog box to set many 3rd options. A detailed description of 3rd, including these options, is available in the file /usr/lib/atom/doc/3rd.ps.

2.4.2 malloc — Check Memory Allocation

The **malloc** tool measures the amount of memory dynamically requested by your application. The Output window displays a table with the following headings:

- Size – Number of bytes requested
- calls – Number of calls made
- allocated – Total number of bytes allocated

In the following example, the first ten calls to *malloc(2)* are made directly from the application code (conftest.c). The extra calls made on the last line do not appear explicitly in the source code. The output of the inline tool shows that the extra calls are made from the *_findbuf()* function call.

```
    Size          calls           allocated
       1              1                   1
     101              1                 101
     201              1                 201
     301              1                 301
     401              1                 401
     501              1                 501
     601              1                 601
     701              1                 701
     801              1                 801
     901              1                 901
    8208              2               16416

  Totals             12               20926

Maximum Memory Allocated: 62128 bytes
```

The corresponding source code which generates these results is as follows:

```
TestMalloc()
{
   int i;
   char *ptr;

   for (i = 1; i < 1000; i += 100)
   {
       ptr =  (char *) malloc(i);
   }
}
```

2.5 Instruction Analysis Tools

The tools in this section analyze assembly language instructions to try to identify where improvements can be made. Whereas profiling tools analyze the execution of your source code, and would probably generate similar results on different machine architectures, instruction analysis tools are tuned specifically to the Alpha instruction set and some features of its architecture.

2.5.1 branch — Branch Prediction Ratio

The **branch** tool counts the number of times your application executes each branch instruction. It computes the ratio of correctly predicted to total number of branches. The Output window displays a table with the following headings:

- PC – Program Counter
- References – Number of references
- Predicted – Ratio of correct predictions

In the following example, the branch instruction at hexadecimal address 0x0000000120005808 was executed 25,000 times, and correctly predicted 99.988% of the time:

```
                 PC           References            Predicted
0x0000000120005640                    23             0.869565
0x000000012000566c                     1             0.000000
0x0000000120005684                     1             0.000000
0x000000012000569c                     1             0.000000
0x00000001200056b0                     1             0.000000
0x0000000120005700                     1             1.000000
0x0000000120005710                     1             1.000000
0x0000000120005808                 25000             0.999880
0x0000000120005848                 25000             0.999880
```

2.5.2 classes — Analyze Instruction Classes

The **classes** tool performs a static analysis of your application by examining the types of instructions used. It does not need to run the application. The Output window displays a count of the total number of each type of instruction. The following example shows the results of the tool when run on the conftest.c program:

```
Static Instruction Counts

         Number Class
           3465 Integer load
             19 Floating point load
           1501 Integer store data
             18 Floating point store data
           1750 Integer branch
              9 Floating point branch
            761 Integer subroutine call
           3640 Integer arithmetic
              1 Integer longword multiply
             39 Integer quadword multiply
           1615 Logical functions
            920 Shift functions
            132 Conditional move
            438 Integer compare
             46 Other floating point operations
              0 Floating point single precision divide
              0 Floating point double precision divide
             24 call pal, hw_x etc

          14378 Total
```

2.5.3 lpipe — Identify Pipeline Stalls

The **lpipe** tool computes the actual and minimum number of cycles needed to execute each basic block in your application. It also counts the number of times each basic block is executed. With this information it can identify code which causes processor pipeline stalls. You can sometimes remove some of the potential pipeline stalls by reorganizing the source code or rewriting some code in assembly language.

The Output window displays a table with the following headings:

- Line – Source code line
- PC – Program Counter

- Inst – Number of instructions
- Pack – Theoretical minimum number of cycles
- Cycl – Current number of cycles
- CPI – Cycles per instruction
- Count – Call count
- Total Cycles – Total number of cycles executed
- Pipe Stalls – Pipeline stalls

```
-----------------------------------------------------------------
TestMalloc : conftest.c
-----------------------------------------------------------------
Line          PC Inst Pack Cycl    CPI    Count    Total    Pipe
                                                  Cycles  Stalls
  57 0x1200059f0    6    4    4   0.67        1        4       0
  63 0x120005a08    3    2    4   1.33       10       40      20
  63 0x120005a14    5    4    4   0.80       10       40       0
  65 0x120005a28    4    3    4   1.00        1        4       1
```

2.5.4 pipe — Identify Pipeline Stalls

The **pipe** tool computes the actual and minimum number of cycles needed to execute each basic block in your application. Its output is displayed in a graph showing the most time-consuming functions, where optimizations will have most effect. Figure 2-2 shows the Output window for this tool. The green bar indicates the number of instructions executed. The orange bar indicates the minimum number of cycles in which these instructions can potentially be executed. The orange and red bars combined show the actual number of cycles taken, so the red bar on its own shows the potential cycles that can be eliminated.

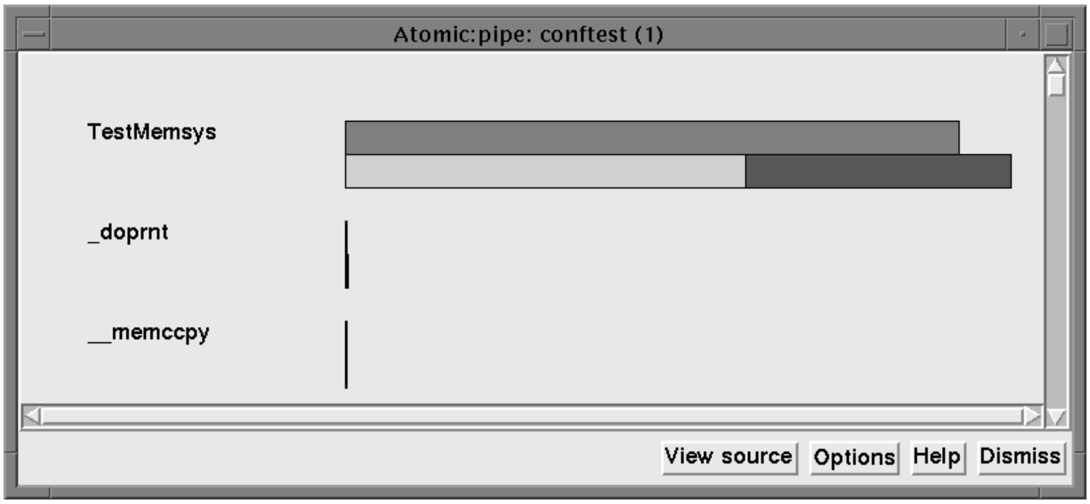

Figure 2-2: Per-function Processor Pipeline Stall Graph

The ''View source'' button displays the Application window. The ''Options'' button displays a submenu with the following options:

- **Set palette**
  Display the Palette Selector dialog box.

- **Dynamic output, graph**
  Display the dynamic pipe information in graph form (default on).
- **Dynamic output, text**
  Display the dynamic pipe information in text form (default off).
- **Static output, text**
  Display the static pipe information in text form (default off).

Press MB3 in any of the panes in the Output window to display a submenu from which you can display a short description, search for a string or location, set the size of each pane, or sort the output.

Double-click on a function name in the graph pane, or on a line in the text panes, to scroll all three panes to the selected function or address.

2.5.5 unalign — Count Unaligned Instructions

The **unalign** tool counts the number of instructions which do not begin on a quadword or longword boundary in your application. The Alpha architecture does not support byte or short load and store instructions, so it uses longword instructions instead. Once loaded, extra operations are required to extract and process only the required data from the longword. Fewer unaligned instructions reduce the number of instructions to be executed and allows the application to run faster.

The Output window displays a table with the following headings:

- Procedure – Procedure or function name
- UnalignInst – Number of unaligned instructions in the function
- Instructions – Total number of instructions in the function
- % Proc – Number of unaligned instructions in the function expressed as a percentage of the total number of instructions in the function.
- % Unalign – Number of unaligned instructions in the function expressed as a percentage of the total number of unaligned instructions in the application.
- % Inst – Number of unaligned instructions in the function expressed as a percentage of the total number of instructions in the application.

In the following example, the *fprintf()* function executed a total of 817 instructions, of which only 152 were unaligned. This represented 18.6% of all the instructions in the function, and 2.325% of all the unaligned instructions in the program. It represented only 0.026% of all the instructions in the program.

```
 Procedure   UnalignInst  Instructions   % Proc  % Unalign   % Inst
    __start            0           122    0.000      0.000    0.000
       main            0            24    0.000      0.000    0.000
TestMemsys             1        550016    0.000      0.015    0.000
     TestIO            1           403    0.248      0.015    0.000
TestMalloc             0            90    0.000      0.000    0.000
    Test3rd            0             1    0.000      0.000    0.000
       exit            0            25    0.000      0.000    0.000
      fopen            0            38    0.000      0.000    0.000
   _endopen           74           195   37.949      1.132    0.013
    fprintf          152           817   18.605      2.325    0.026
        ...          ...           ...      ...        ...      ...
      Total         6537        576257                          1.134
```

Overview of ATOMiC Windows 3

This chapter describes the windows you will encounter while using ATOMiC. Because ATOMiC is built with the Tk/Tcl toolkit, the look and feel of the windows is slightly different from Motif, the default user interface standard. One obvious difference is that buttons are highlighted in a darker color as soon as you move the mouse over them.

3.1 The Session Window

The Session window, shown in Figure 3-1, is the main display window for ATOMiC. It consists of a menu bar and a number of panes. The Application pane displays the following fields:

- **Executable**
  The original version of the application, from which instrumented executables are generated.
- **Description**
  A description of any changes made to the original version. If you modify the original version, ATOMiC automatically prompts for a new comment to be entered in this field.
- **Command line args**
  Any arguments to be passed to the application can be entered here. See the online help for further information.
- **Setup command**
  Specify a script to carry out any actions required before the application is run here. A script which calls *make* ensures that the executable is always up-to-date with respect to the source code files.

  Specify an explicit target in the Makefile to create a relocatable, non-shared executable image and have the script make that target.
- **Cleanup command**
  A script to carry out any actions required after the application is run can be specified here if necessary. For example, some output files might be deleted.

The Tool Status pane displays information about the current state of ATOMiC. When it is waiting for user input, the status is Idle. When it is instrumenting and running your application with a particular tool, it displays the tool name and the stage of the process.

Figure 3-1: Session Window

The Log pane displays a copy of the file which is a complete record of all your actions in all sessions of ATOMiC with this application. You can type your own additional comments in the scrolling text window. The following buttons are available:

- **Save**
  Save the log text to the atomic.log file. If you forget to save the log after editing it, ATOMiC will remind you when you exit the program.
- **Wrap**
  Enable or disable word wrapping of the log.
- **Text Search...**
  Search for text in the log. This button opens the Text Search dialog box.

3.1.1 The File Menu

The File menu contains the following items:

- **Environment...**
  This menu item displays the Environment Variables dialog box, shown in Figure 3-2, in which you can specify environment variables used by the ATOM tools. The description of each tool identifies which variables, if any, affect its behavior.

Many tools check for the ATOMUNIQUE environment variable, and append the process ID to their output files if it exists.

The variables used by the tools are at the top of the scrolling window containing the list of environment variables. ATOMiC also reads all environment variables which are active in its start-up window.

You can create new variables by entering the name and value in the appropriate fields and clicking ‘‘OK.’’

You can delete variables by clicking on the toggle button to the left of the name. The value field will be deleted.

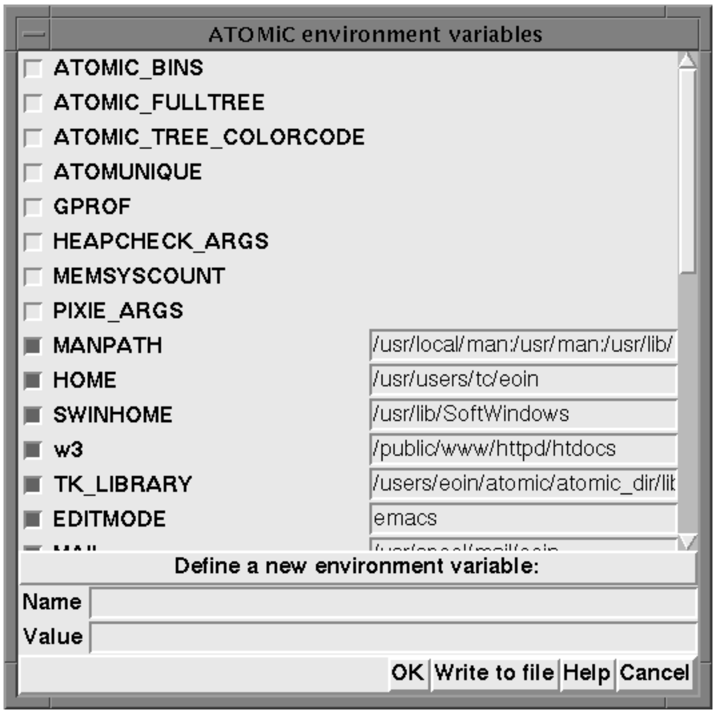

Figure 3-2: Environment Variables Dialog Box

- **Search path...**
  This menu item displays the Search Path dialog box. The search path is a list of directories. ATOMiC searches this list for source code files for your application. To set the search path, enter the directories (one per line), and click on ‘‘OK.’’

  When source code is displayed in an Application window, ATOMiC searches for the source files in the list of directories specified here.

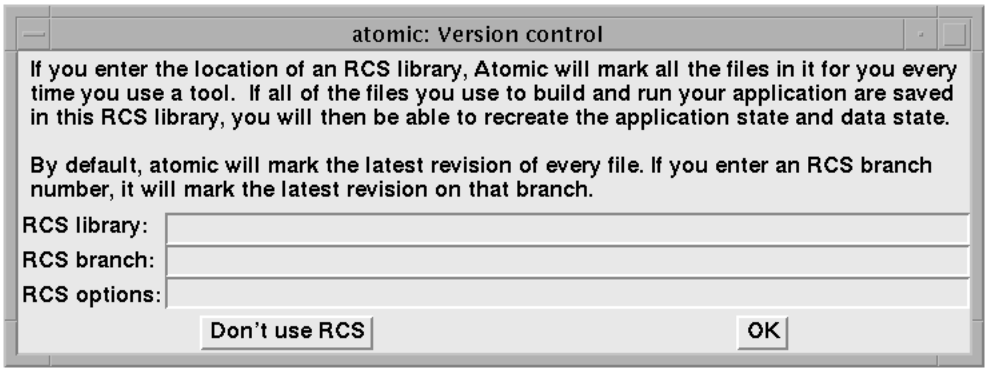

Figure 3-3: Version Control Dialog Box

- **Version control...**
  This menu item displays the Version Control dialog box shown in Figure 3-3. You can specify the following options for use with RCS-based version control:

  – An RCS directory

  – An RCS branch

  – Options to the *rcs* command.

  The RCS directory is where your source code is stored. ATOMiC marks all the files in the directory each time you use a tool.

  The RCS branch number indicates to ATOMiC the branch in each file where a mark is to be placed. The latest revision on that branch is used. By default, ATOMiC marks the latest revision of every file.

  The RCS options are used by ATOMiC when calling the *rcs* command to mark files. The -q option (for quiet mode) is automatically used. The -N option is used to mark files. The name ATOMIC_DATA with a numerical suffix, ATOMIC_DATA3 for example, is used.

  If all files used to build and run your application are maintained in one RCS directory, you can recreate any version of the application by using the appropriate markers. The log file displays the marker associated with each version.

  It is probably best to copy a snapshot of your application source code to the working directory and create a temporary RCS directory to mark changes. If you can identify the small number of files which are critical to performance, place only these files in the RCS directory, and use libraries for the others. This reduces the amount of marking ATOMiC needs to do.

- **Maintenance...**
  ATOMiC makes a subdirectory to store each instrumented executable version of your application, and the output files from each run of the instrumented tools. This menu item displays a dialog box, shown in Figure 3-4, listing all the subdirectories and the amount of space used in each.

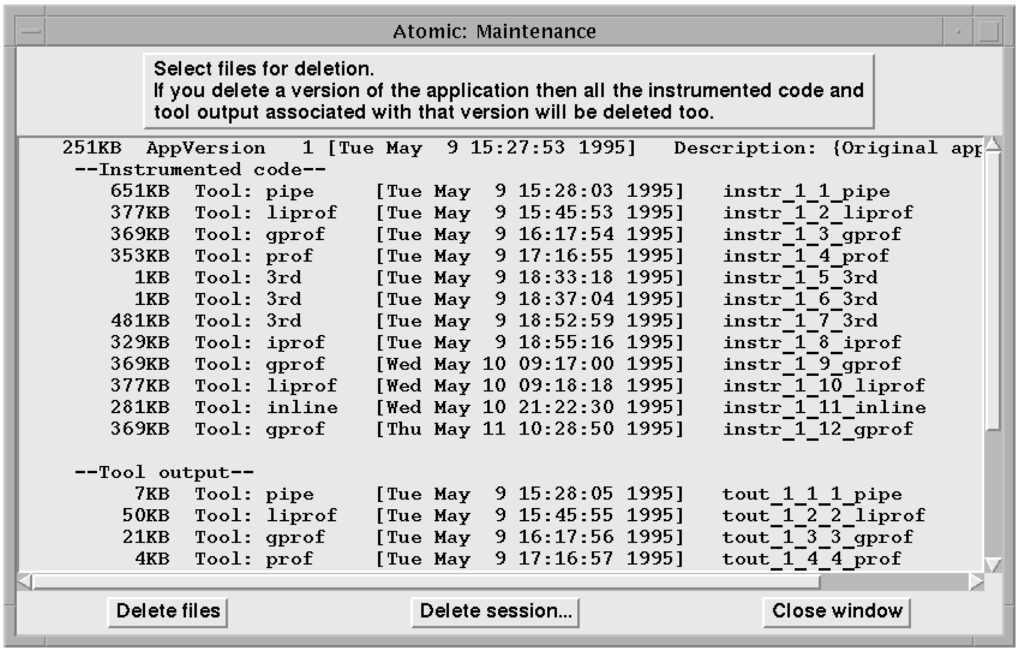

Figure 3-4: Maintenance Dialog Box

- Select items from the list by clicking on a line, or pressing MB1 and dragging the mouse up or down. Selected files are highlighted in blue. Delete selected files by clicking on the ‘‘Delete files’’ button.

  Click on the ‘‘Delete session...’’ button to delete all atomic subdirectories and exit ATOMiC.

  Click on a line containing ‘‘AppVersion’’ to select all instrumented executable and tool output subdirectories for a single version of the application.

  Click on a line containing ‘‘--Instrumented code--’’ to select all instrumented executable subdirectories for a single version of the application.

  Click on a line containing ‘‘--Tool output--’’ to select all tool output subdirectories for a single version of the application.

- **Exit**
  Close all windows and exit ATOMiC.

3.1.2 The Tools Menu

The Tools menu contains the names of the tools currently registered with ATOMiC. Click on any of the tool names to create an executable version of your application, instrumented with that tool.

If you add your own new tools, you can access them from this menu. See Appendix B for information on adding new tools. All tools are controlled from the Tool Chooser dialog box, shown in Figure 3-5.

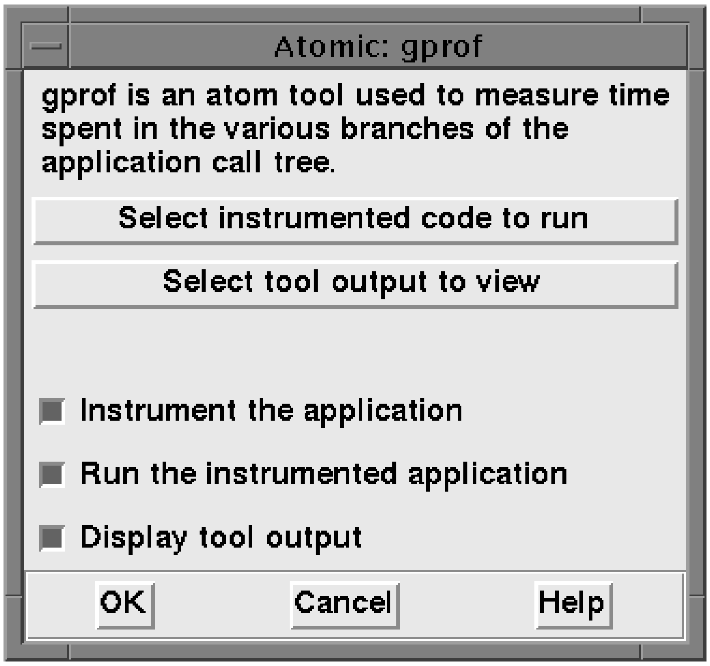

Figure 3-5: Tool Chooser Dialog Box

The top of the dialog box displays a short explanation of the tool you have chosen.

To re-run a previously created instrumented executable, click on the button marked ‘‘Select instrumented code to run’’. This displays a scrolling list of all versions of executables instrumented with that tool. Double click on a line containing ‘‘Tool:’’ to run a particular version.

To re-display an output file which was previously created, click on the button marked ‘‘Select tool output to view’’. This displays a scrolling list of all copies of tool output created with that tool. Double click on a line containing ‘‘Tool:’’ to display a particular output file. (This is equivalent to the View→Tool output menu item.)

3.1.3 The View Menu

This menu controls which windows are displayed, and contains the following items:

- **Application**
  This menu item displays a submenu which lists the available versions of the application. The initial version of the application before you make any changes is available as the ’Current’ menu item. Each time the application relocatable executable file you have specified changes, ATOMiC copies it to its own subdirectory, creates a new version number for it, and adds it to this menu.
- **Tool output...**
  This menu item displays a dialog box listing all available tool output files. The files are grouped according to the version number of the executable from which they were created. Double click on a line containing a tool name to display a particular output file. Double click on a line containing ‘‘AppVersion’’ to display the Application window of a particular version.

- **Program output...**
  This menu item displays a dialog box listing the available program output files. The files are grouped according to the version number of the executable from which they were created. Double click on a line containing a tool name to display a particular output file. Double click on a line containing ‘‘AppVersion’’ to display the Application window of a particular version.

  The output of your application is displayed in a temporary window while it is running. The temporary window also accepts input for the application. The file stores both input and output.

- **Viewers...**
  This menu item displays a dialog box listing the currently available ATOMiC windows. Double click on any line to bring the corresponding window to the foreground. The window is opened if it was minimized.

3.1.4 The Help Menu

This menu displays the following items:

- **Contents**
- **Overview**
- **Session window**
- **Product Information**
- **Help on Help**

Choose any of the items to open a help window on that topic. See Section 3.4 for a detailed description of the ATOMiC help window.

3.2 The Application Window

The Application window consists of three panes used to display a list of the functions in your application, the source code, and the assembly code. At the top of the window, shown in Figure 3-6, is a set of toggle buttons to control which panes are currently visible.

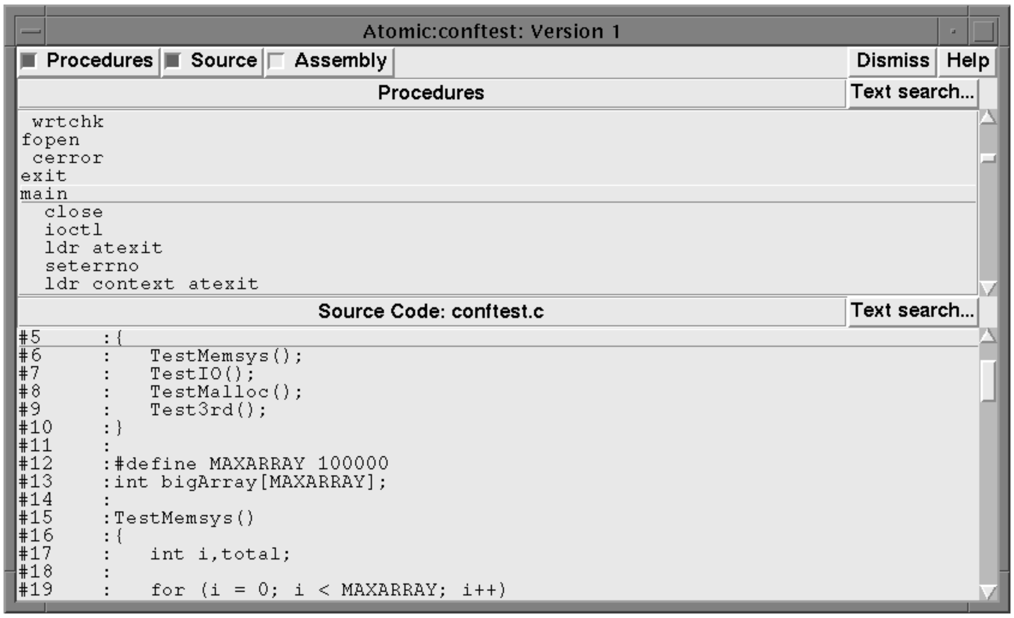

Figure 3-6: Application Window

The three panes are synchronized, so that clicking on a function name in the Procedures pane automatically scrolls the Source and Assembly panes to that function.

Each pane has its own Text search... button to the right of its title bar. The Text Search dialog box is described in Section 3.5.3.

Normally, you display the Application window for the current version of your application. But, since ATOMiC copies all previous versions of your applications' files, you can use the View→Application menu of the Session window to open an Application window for any version of your application,

3.2.1 Application Window Menus

Each of the panes has its own specific menu. To access these menus, place the mouse in the pane you want and press MB3. To select an item in the menu, press MB3 over the item. To make the menu disappear, press MB1 or MB3 anywhere outside the menu.

Each of the menus contains at least two of the following items:

- **Description...**
  Display a short explanation of the contents of the pane.
- **Find location...**
  Search for a source line, function name or PC address in the pane. See Section 3.5.4 for further information.
- **Text Search...**
  Search for text in the pane. See Section 3.5.3 for further information.

- **Set window height...**
  Change the number of visible lines in the pane. See Section 3.5.5 for further information.
- **Close View**
  Delete the pane from the Application window. This is equivalent to clicking on one of the three toggle buttons at the top of the window.

The Procedures pane menu has a Sort menu item, which displays a submenu and gives the following sort order options:

- Name order
- Address order
- Inverse name order
- Inverse address order

Some tools cause extra columns to be displayed in the Procedures pane, for example, the liprof tool. These columns, one of which is shown in Figure 3-7, have extra menu items in their menus.

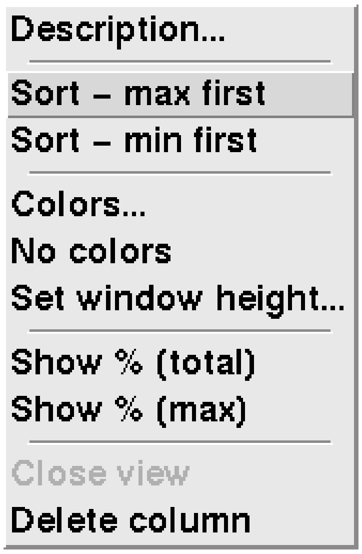

Figure 3-7: liprof Column Menu

The following extra menu items are included:

- **Sort - max first**
  Sort all columns in decreasing order of the values in this column. This gathers the busiest functions at the top of the list.
- **Sort - min first**
  Sort all columns in increasing order of the values in the column.
- **Colors...**
  Display the Set Bins for Statistics dialog box. See Section 3.5.1 for further information.
- **No colors...**
  Disable the color-coding of the column values. To enable the color-coding again, choose the Colors... menu item and click on OK in the dialog box.

- **Show % (total)**
  Display a new column showing the percentage of total time spent in each function.
- **Show % (max)**
  Display a new column showing the time spent in each function as a percentage of the maximum time.
- **Delete column**
  Delete this column.

3.3 The Output Window

Most of the ATOMiC tools display their output in an ordinary scrolling text window. These windows have a button bar across the bottom which contains the following items:

- **View source**
  Display the Application window.
- **Address**
  When a Program Counter address in the text window is selected, scroll the Application window Assembly pane to that address.
- **Proc Name**
  When a function name in the text window is selected, scroll the Application window Source and Assembly panes to that function.
- **Text search...**
  Display the Text Search dialog box.
- **Dismiss**
  Close the Output window.
- **Help**
  Display the Help window.

3.4 The Help Window

The Help window for ATOMiC is based on **BLT**, a simple Tk/Tcl hypertext browser. Click on a Help button in any window to display the Help window, which is shown in Figure 3-8.

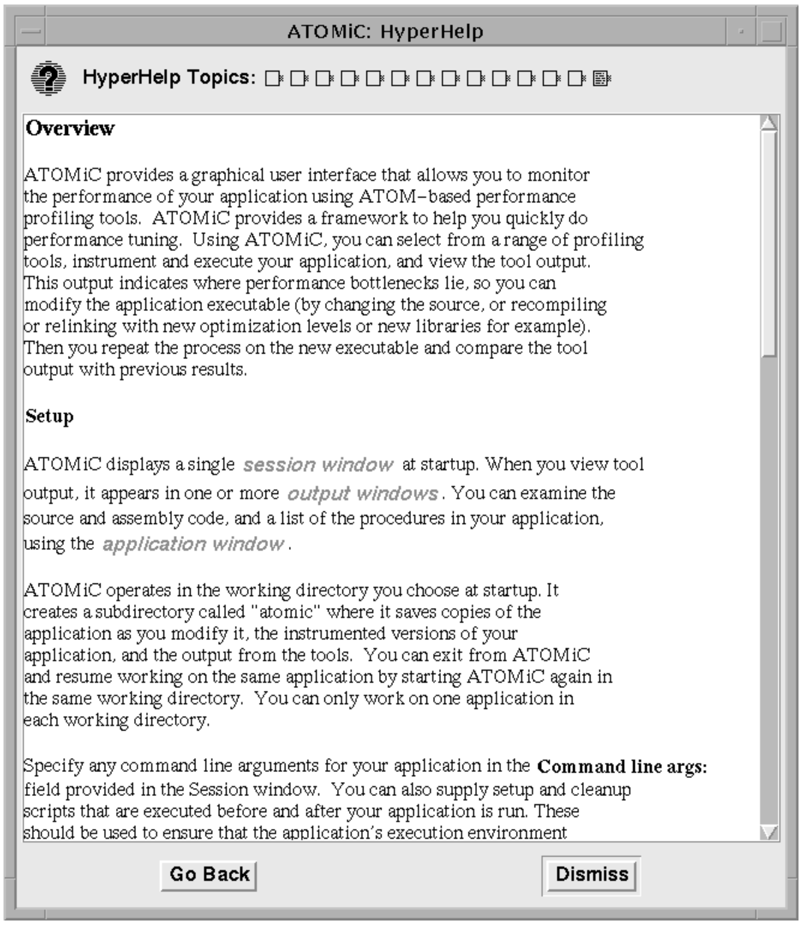

Figure 3-8: Help Window

Click on the scroll bar to read the help text sequentially. In addition, text in blue can be clicked on to go directly to that topic. Any item in the table of contents can be clicked on to go directly to it. Click on the ‘‘Go Back’’ button at the bottom of the window to get back to a previous page. Click on the ‘‘Dismiss’’ button to remove the Help window.

As each link is traversed or help option is selected, a new icon is appended to a chain of icons at the top of the window. You can revisit any previous page by clicking on the corresponding icon. The current page in the hyperlink chain is represented by a grayed-in icon. Other icons are white. You can scroll through the help text sequentially, or by clicking on hypertext links to go directly to other locations in the help text.

3.5 Dialog Boxes

This section describes the dialog boxes which can be accessed from different menus and window buttons in ATOMiC.

3.5.1 Set Bins for Statistics

ATOMiC uses color to highlight results in many of the windows it displays. Function names are displayed in ten colors ranging from green to red, depending on the percentage of processing time spent in that function. This dialog box, shown in Figure 3-9, allows you to define the range of percentage values to which each of the ten colors applies. By default, each bin represents a five percentage point range, except the last, which represents anything above 45%.

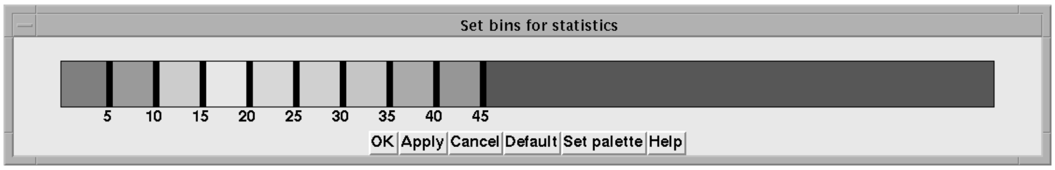

Figure 3-9: Set Bins for Statistics Dialog Box

To change the ranges, press MB1 over the black bar between colors, and move the mouse left or right. The numeric value under the bar increases or decreases as you move the bar. Click ‘‘OK’’ or ‘‘Apply’’ to apply the changed bin sizes.

Only the colors of the window or column from which this dialog box was selected are affected. This means that different bins can be used for the output of different tools.

The button bar at the bottom of the window has the following items:

- **OK**
  Apply the current bin sizes and close the dialog box.
- **Apply**
  Apply the current bin sizes without closing the dialog box.
- **Cancel**
  Close the dialog box without applying the current bin sizes.
- **Default**
  Reset the bin sizes to the default values.
- **Set palette**
  Open the Palette Selector dialog box.
- **Help**
  Open the Help window.

3.5.2 Palette Selector

This dialog box allows you to change the color used to represent each of the ranges of values specified in the Set Bins for Statistics dialog box. The dialog box is shown in Figure 3-10.

To change a color, click on the color you wish to modify then use the red, green or blue slider to change the amount of that color used. The color of the button changes as you move the slider.

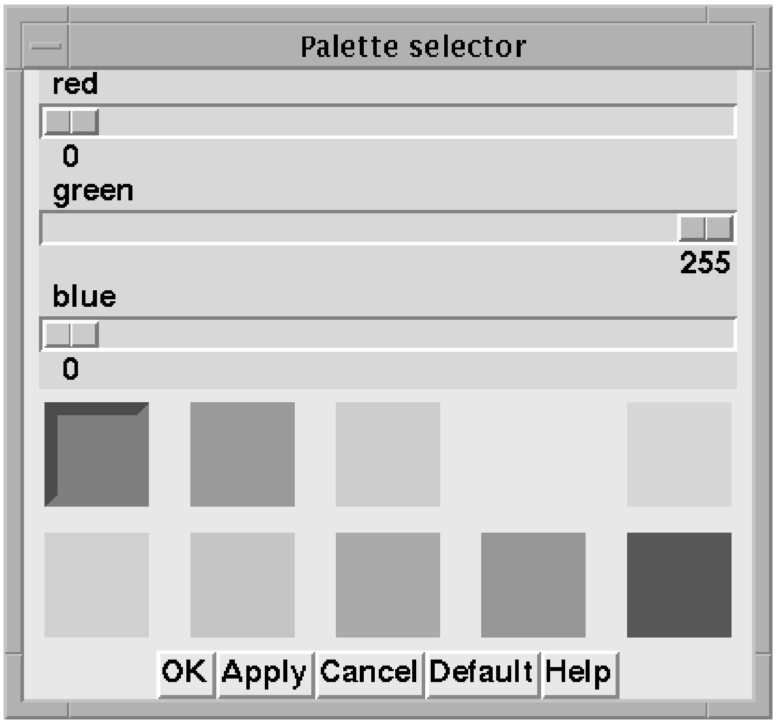

Figure 3-10: Palette Selector Dialog Box

The button bar at the bottom of the window has the following items:

- **OK**
  Apply the current colors to the palette and close the dialog box.
- **Apply**
  Apply the current colors to the palette.
- **Cancel**
  Close the dialog box without applying the current palette.
- **Default**
  Reset all colors to the default values.
- **Help**
  Open the Help window.

Note that since ATOMiC is built with the Tk/Tcl toolkit it does not use a private colormap, and therefore might run out of colors if it is started when another application using a lot of colors is already running. When this happens, the background color of any window that uses color is set to black. Since the foreground color used for text is also black, the text is invisible. In such cases this dialog box can be used to change the black background to white.

3.5.3 Text Search

This dialog box allows you to search (using regular expressions) for a pattern in a window. Enter the search pattern and press Return to start the first search. Click on the ‘‘Next’’ or ‘‘Prev’’ buttons to start subsequent forward or backward searches. The dialog box is shown in Figure 3-11.

Note that to search for regular expression meta-characters (such as \, *, [and]), the escape character \ must be used. To search for the string ‘‘array[4]’’, for example, you must type ‘‘array\[4\]’’.

The ‘‘Reset’’ button clears the search pattern and moves the visible part of the scrolling region being searched to the beginning. The next search starts from there.

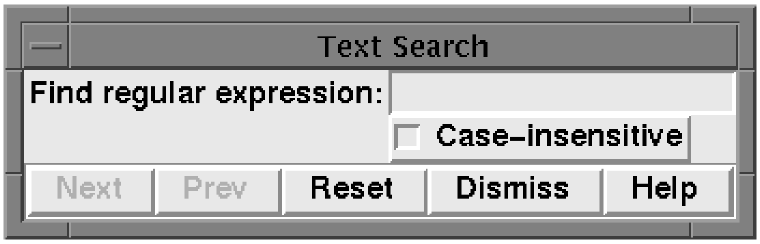

Figure 3-11: Text Search Dialog Box

3.5.4 Location Search

This dialog box, shown in Figure 3-12, allows you to search for a procedure name or an address or a source line number in a particular version of the application.

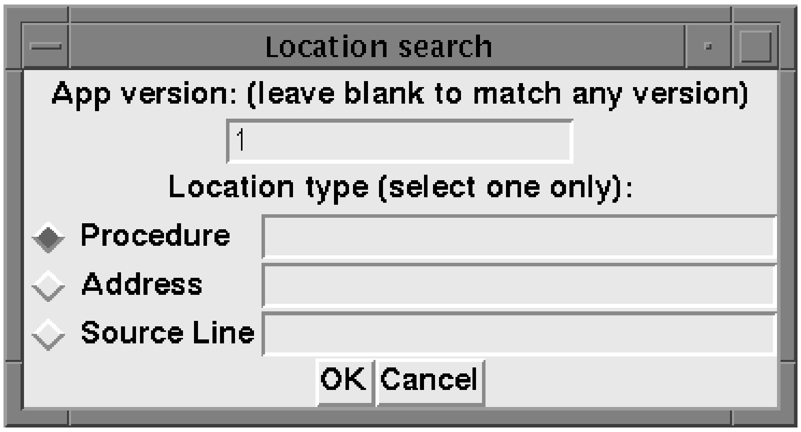

Figure 3-12: Location Search Dialog Box

3.5.5 Set Window Height

This dialog box, shown in Figure 3-13, allows you to set the number of lines to appear in the pane of the Application window from which the dialog box was called. The defaults are ten lines for the Procedures pane and fifteen lines for the Source and Assembly panes. To change this value, move the slider left or right on the scale and click ‘‘OK.’’ The minimum and maximum number of lines are one and twenty.

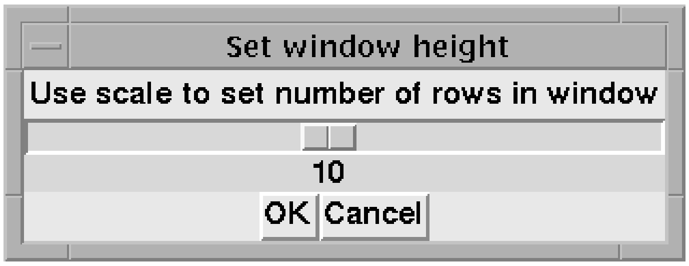

Figure 3-13: Set Window Height Dialog Box

Installation A

The ATOMiC software package can be retrieved in any of the following ways:

– Anonymous **FTP**

– E-mail

– A **World-Wide-Web** browser

The package is stored in compressed archive form at the following location: ftp.digital.com/pub/DEC/atomic/atomic_V1.0.tar.Z.

A.1 How to Retrieve ATOMiC

To transfer the file via anonymous FTP, use the following steps:

1. Connect to the DEC FTP site using the ftp program:

```
% ftp ftp.digital.com
220 gatekeeper.dec.com FTP server (Version 5.173 Thu Apr 27 ...
```

2. Log in as the user anonymous, giving your e-mail address as the password:

```
Name (ftp.digital.com:toms): anonymous
331 Guest login ok, send ident as password.
Password: your_email_address
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
```

3. Change directory to the atomic repository, and change the transfer mode to binary:

```
ftp> cd /pub/DEC/atomic
250 CWD command successful.
ftp> bin
200 Type set to I.
```

4. Retrieve the file using the get command:

```
ftp> get atomic_V1.0.tar.Z
200 PORT command successful.
150 Opening BINARY mode data connection for atomic_V1.0.tar.Z ...
226 Transfer complete.
1276881 bytes received in 52 seconds (24 Kbytes/s)
ftp> quit
```

To transfer the file using Mosaic or another Web browser, start the browser and open the URL: ftp://ftp.digital.com/pub/DEC/atomic/atomic_V1.0.tar.Z. The browser will retrieve the file and prompt you for a location in which to save it.

Users with Internet-capable electronic mail can use the FTPmail service to retrieve files from anonymous FTP sites even without direct Internet access. To retrieve the file using FTPmail, send a mail with the following message body to ftpmail@gatekeeper.dec.com:

```
connect
binary
chdir /pub/DEC/atomic
get atomic_V1.0.tar.Z
quit
```

You will receive an acknowledgement of your request first, followed by the file itself, encoded using the *uuencode* utility. The contents will be spread over a number of mail messages, as FTPmail splits the file into chunks of 64 KB by default. FTPmail is a low-priority service, so it might take a number of days for the file to be sent. If you do not receive an acknowledgement, then the FTPmail service might be temporarily broken, and could take up to a week to fix.

A.2 How to install ATOMiC

To install ATOMiC, you must have super-user privileges:

1. Uncompress and untar the package. A subdirectory called ATC100 is created in the current directory:

   ```
   % uncompress atomic_V1.0.tar.Z
   % tar xf atomic_V1.0.tar
   % ls
   ATC100 atomic_V1.0.tar
   ```

2. Log in as root:

   ```
   % su
   Password: root_password_here
   ```

3. Run the **setld** command and answer any questions it asks:

   ```
   # setld -l ATC100
   ```

The ATOMiC program is now installed, and the executable placed in /usr/ucb/atomic.

Adding a New Tool B

This appendix describes how to add a new ATOM tool to ATOMiC. The instrumentation and analysis source code files must exist in the directory /usr/lib/atom/tools. For information on how to create ATOM tools, see the *ATOM Reference Manual*. Super-user privileges might be required to place new ATOM tools in /usr/lib/atom/tools, and modify files in the ATOMiC installation directory.

There are three stages in running a performance analysis tool:

1. Instrument the executable image with a tool
2. Run the instrumented image
3. View the output of the tool

The file AtomicToolRegister.tcl in the ATOMiC installation directory is used to control the three stages. This file is a Tcl script which defines how ATOMiC should carry out each stage for each of the registered tools. It is not necessary to understand Tcl to add a new tool to ATOMiC, because you can simply copy and modify lines from a current entry. However, if you wish to add tools which require non-standard handling, then some knowledge is required. This section describes only how to add a simple tool. The file AtomicToolRegister.tcl contains a detailed description of its format, if you need more details on the tool registration process.

This example assumes that the ATOM tool is called **stack** and that the files stack.anal.c and stack.inst.c are in the directory /usr/lib/atom/tools. Each tool entry consists of the keyword AtomicTool and the following fields, all enclosed in double-quotes:

- The name of the tool
- A short description of the tool
- The keyword AtomicAtomInstrument and the tool name
- The keyword AtomicRun
- The keyword AtomicDefaultView and a number of arguments

To add a new tool, copy an existing entry and change the tool name. The following example shows a tool entry for stack:

```
AtomicMenuSeparator

AtomicTool "stack"\
    "stack is an atom tool used to measure\
     the growth of the runtime stack."\
    "AtomicAtomInstrument stack"\
    "AtomicRun"\
    "AtomicDefaultView 0 0 \"{}\" stack stack.out"
```

To distinguish new tools from existing entries, the keyword AtomicMenuSeparator can be placed at the end of the default file, before any new tool entries. This adds a separator bar in the ATOMiC Tools menu.

The example shown is only designed to display a text output file. The last argument (stack.out) on the last line is the name of the output file to view. ATOMiC searches for this file, but if it cannot find it, it treats the argument as a prefix, and searches for a file which begins with this string. This allows a suffix such as a process ID to be added to the file name.

The file is a Tcl script, so it must be syntactically valid Tcl code. In particular, you must ensure that special Tcl characters are preceded by a backslash (\) character, as on the last line of the example. Other Tcl special characters are the dollar ($), double-quote ("), square bracket ([,]) and accolades ({, }).

Glossary

This glossary contains definitions of some of the terms used in this guide.

3rd tool
: An atom tool to check code for references to memory which does not contain valid data.

assembly code
: Instructions which can be interpreted directly by a processor.

ATOM
: Analysis Tools with OM. A system for building customized program analysis tools available with DEC Alpha OSF/1 platforms.

basic block
: A series of assembly language instructions which is guaranteed to be executed in sequence and in its entirety.

bin sizes
: A percentage range of values represented by a single color in ATOMiC. For example, zero to five percent is represented by the color green.

BLT
: Bacon, Lettuce and Tomato. A simple hypertext-based help system for the Tk/Tcl toolkit.

branch tool
: An ATOM tool which identifies code for branch instructions, and examines the effect on program performance.

cache
: A block of memory internal to the processor which is used as a high-speed store for data.

call-graph
: A representation of the hierarchy of function calls made within an application.

classes tool
: An ATOM tool which analyzes the types of instructions used in an application.

dirty block
: A block of data stored in memory to which modifications have been made, meaning that the block must later be saved to disk or secondary memory.

dtb
: Data Translation Buffer.

FTP
: File Transfer Protocol. A standard communications protocol used to copy files between any two machines running TCP/IP software.

gprof
: Graph Profile. The name of a UNIX software development tool, and a

corresponding ATOM tool, which computes the hierarchy of function calls made within a program.

inline tool
An ATOM tool which examines the calling sequence of functions in an application.

instrumented application, instrumented executable
An executable image file which has extra code inserted by an ATOM tool to generate performance-related data when the image is executed.

io tool
Input/Output tool. An ATOM tool which examines the input and output functions used in an application.

io2 tool
An ATOM tool which examines the input and output files used in an application.

iprof tool
Instruction Profile tool. An ATOM tool which examines the number of instructions in each function of an application.

liprof tool
Line Instruction Profile tool. An ATOM tool which examines the number of instructions in each basic block of an application.

lmemsys tool
Line Memory System tool. An ATOM tool which examines the load and store instructions in an application and estimates the time taken to handle the resulting cache events.

lpipe tool
An ATOM tool which examines the actual and potential minimum number of cycles required to execute basic blocks of code.

malloc tool
An ATOM tool which examines the use of the *malloc(2)* system call, and displays the size of each piece of memory allocated.

memsys tool
Memory System tool. An ATOM tool which simulates the use of the data and instruction caches of an Alpha chip. It estimates the time spent handling cache events on a per-function basis.

non-shared libraries
Archive libraries containing object code which are linked into an executable image at the link stage, rather than at run-time. The resulting image is much larger than one linked with shared libraries.

pane
A section of a window, usually with its own title bar, for example, the Source pane of the Application window.

pixie
The name of a UNIX software development tool, and a corresponding ATOM tool, which identifies basic execution blocks in an executable image and counts how many times each is executed.

processor cache
A block of memory residing inside a CPU, used to store frequently or recently

used data.

RCS directory
Revision Control System directory. A repository for storing multiple revisions of files belonging to a project.

relocation information
Information in an image file required by ATOM. This information is removed from the executable by the linker unless the -r flag is used.

source code
Software that, usually, is written in a high-level structured programming language such as C.

syscall tool
An ATOM tool which shows the usage of UNIX system calls in an application. System calls are documented in Section 2 of the UNIX man pages.

tb tool
Translation Buffer tool. An atom tool to simulate the behavior of the translation buffer. See also translation buffer.

Tk/Tcl toolkit
Toolkit/Tool Command Language. A public-domain software package used to develop X Window System user interface applications.

translation buffer
An area of memory on the Alpha processor used to store temporary data for high-speed access.

unalign tool
An atom tool which examines code for instructions that are not aligned on long-word or quad-word boundaries.

URL
Unique Resource Locator. A string which uniquely identifies any resource on the World-Wide Web. It consists of a protocol specification, an Internet node name, and a path to a file name.

wall-clock time
The duration of time as perceived by a user running a task. The total time duration includes time spent running the users task, and also time spent on tasks for other users.

word
A unit of four bytes of memory. A quad-word is eight bytes and a long-word is sixteen bytes.

World-Wide Web
An Internet protocol which allows point-and-click navigation through documents on different machines connected in a network.