# New IDE Application Profiler Enhancements

**Authored by: Elena Laskavaia**

The new Application Profiler features are currently under development for the next release of QNX Momentics. Use the forum and provide us with your comments and suggestions.

---

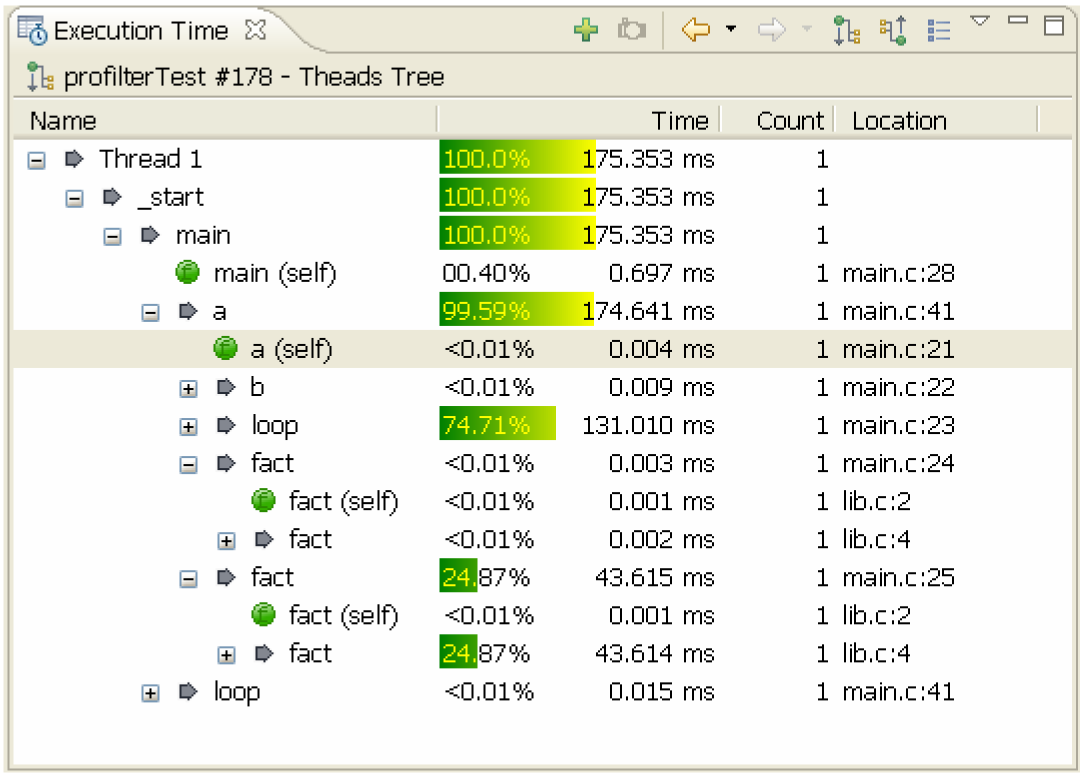
## Profiling Techniques

---

**Function enter/exit time instrumentation**

This type of instrumentation is the most effective way of optimizing bottlenecks in single application. The data collection technique lets you gather precise information about duration of time that the processor spent in each function, and provides stack trace and call count information at the same time.

Now, you can see exactly where application spending you time and how it gets there using a Call Tree. New Time column shows you time spent in the function itself and all of its descendants, as well as the minimum (Min), maximum (Max) and average (Average) total time for all invocations of a function. Also, you can view who called the function, and how much time each function took to execute in context of a caller.

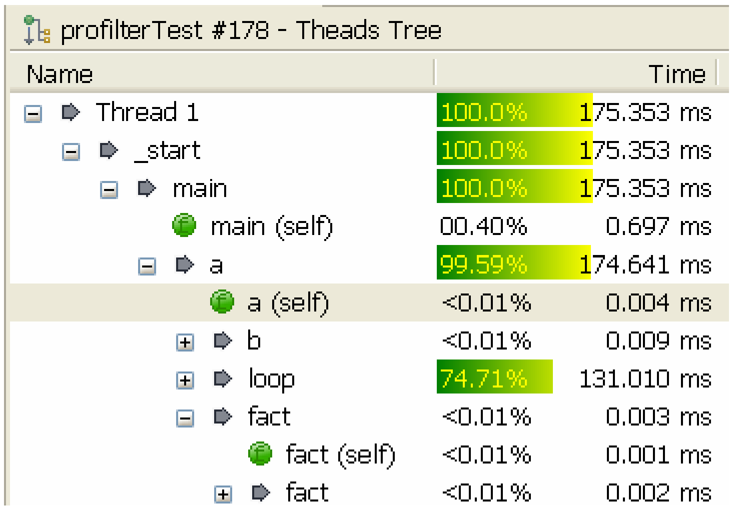


profilterTest #178 - Theads Tree

| Name | Time | | Count | Location |
|---|---|---|---|---|
| Thread 1 | 100.0% | 175.353 ms | 1 | |
| _start | 100.0% | 175.353 ms | 1 | |
| main | 100.0% | 175.353 ms | 1 | |
| main (self) | 00.40% | 0.697 ms | 1 | main.c:28 |
| a | 99.59% | 174.641 ms | 1 | main.c:41 |
| a (self) | <0.01% | 0.004 ms | 1 | main.c:21 |
| b | <0.01% | 0.009 ms | 1 | main.c:22 |
| loop | 74.71% | 131.010 ms | 1 | main.c:23 |
| fact | <0.01% | 0.003 ms | 1 | main.c:24 |
| fact (self) | <0.01% | 0.001 ms | 1 | lib.c:2 |
| fact | <0.01% | 0.002 ms | 1 | lib.c:4 |
| fact | 24.87% | 43.615 ms | 1 | main.c:25 |
| fact (self) | <0.01% | 0.001 ms | 1 | lib.c:2 |
| fact | 24.87% | 43.614 ms | 1 | lib.c:4 |
| loop | <0.01% | 0.015 ms | 1 | main.c:41 |

## UI Changes

**Call Tree Mode** You can drill down from the thread entry function to see how actual time distributes per function descendents. This is the default Call Tree mode for the new instrumented profiling. If you need time aggregated for particular function, use the context menu to explore aggregated Callees tree for this function.

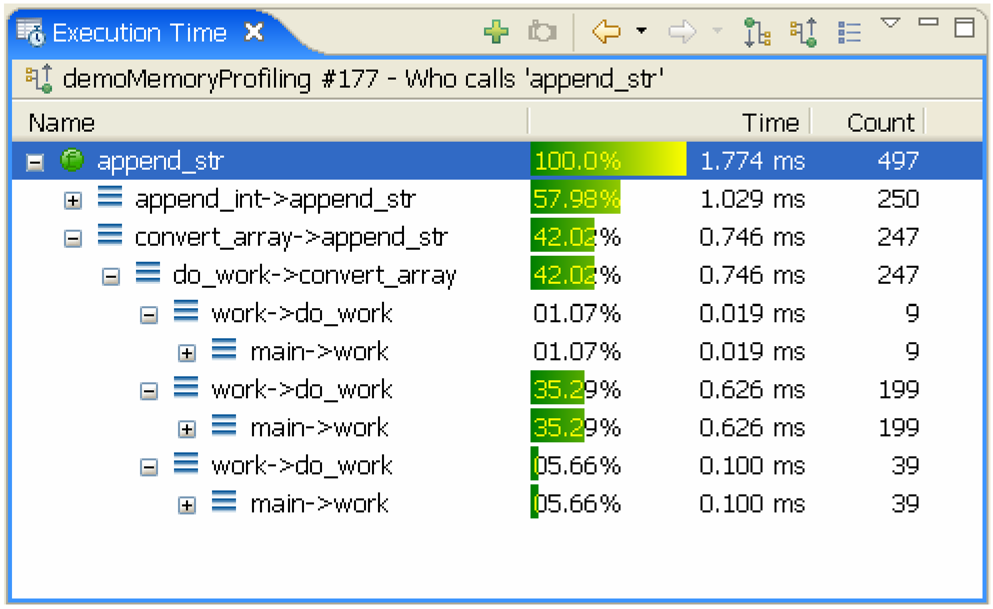


profilterTest #178 - Theads Tree

| Name | | Time |
|---|---|---|
| Thread 1 | 100.0% | 175.353 ms |
| _start | 100.0% | 175.353 ms |
| main | 100.0% | 175.353 ms |
| main (self) | 00.40% | 0.697 ms |
| a | 99.59% | 174.641 ms |
| a (self) | <0.01% | 0.004 ms |
| b | <0.01% | 0.009 ms |
| loop | 74.71% | 131.010 ms |
| fact | <0.01% | 0.003 ms |
| fact (self) | <0.01% | 0.001 ms |
| fact | <0.01% | 0.002 ms |

**Reverse Call Tree Mode**

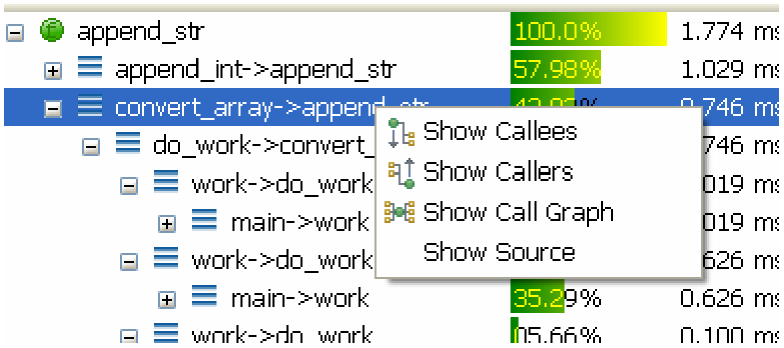
Who calls a specific function, and how is the time distributed per caller? A Reverse Call Tree helps answer this question. You can drill down (up in the stack) and check the callers and their contribution time until you encounter a thread entry function.



| Name | | Time | Count |
|---|---|---|---|
| append_str | 100.0% | 1.774 ms | 497 |
| append_int->append_str | 57.98% | 1.029 ms | 250 |
| convert_array->append_str | 42.02% | 0.746 ms | 247 |
| do_work->convert_array | 42.02% | 0.746 ms | 247 |
| work->do_work | 01.07% | 0.019 ms | 9 |
| main->work | 01.07% | 0.019 ms | 9 |
| work->do_work | 35.29% | 0.626 ms | 199 |
| main->work | 35.29% | 0.626 ms | 199 |
| work->do_work | 05.66% | 0.100 ms | 39 |
| main->work | 05.66% | 0.100 ms | 39 |

**Context Navigation**

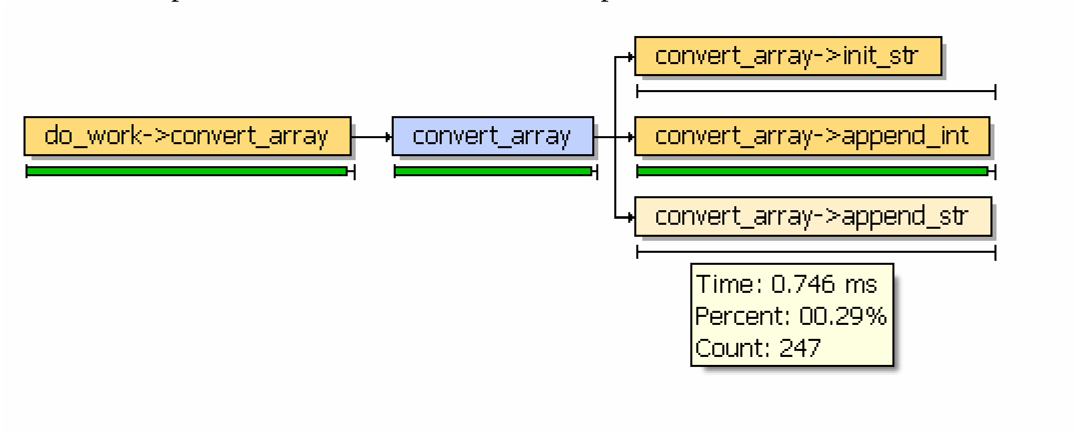
An easy to use context navigation menu is available for each node of the tree, table or call graph. The actions in the context menu are: Show Callees - shows the functions that are called by selected function, Show Callers - lists the functions that called selected function, Show Call Graph - shows an illustration of the runtime call graph.



| | | |
|---|---|---|
| append_str | 100.0% | 1.774 m |
| append_int->append_str | 57.98% | 1.029 m |
| convert_array->append | | 0.746 m |
| do_work->convert_ | | 746 m |
| work->do_work | | 019 m |
| main->work | | 019 m |
| work->do_work | | 626 m |
| main->work | 35.29% | 0.626 m |
| work->do_work | 05.66% | 0.100 m |

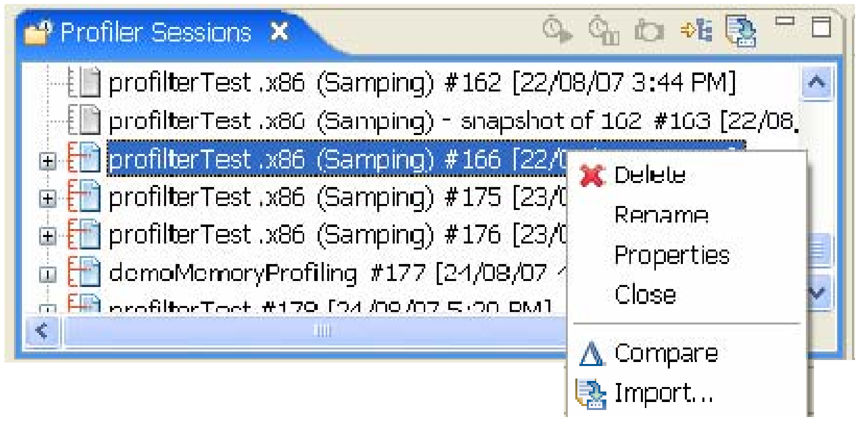
**Call Graph Mode Improvements**

The Call Graph was enhanced to include tooltips and a context menu.



**Profiler Session View with session Persistence**

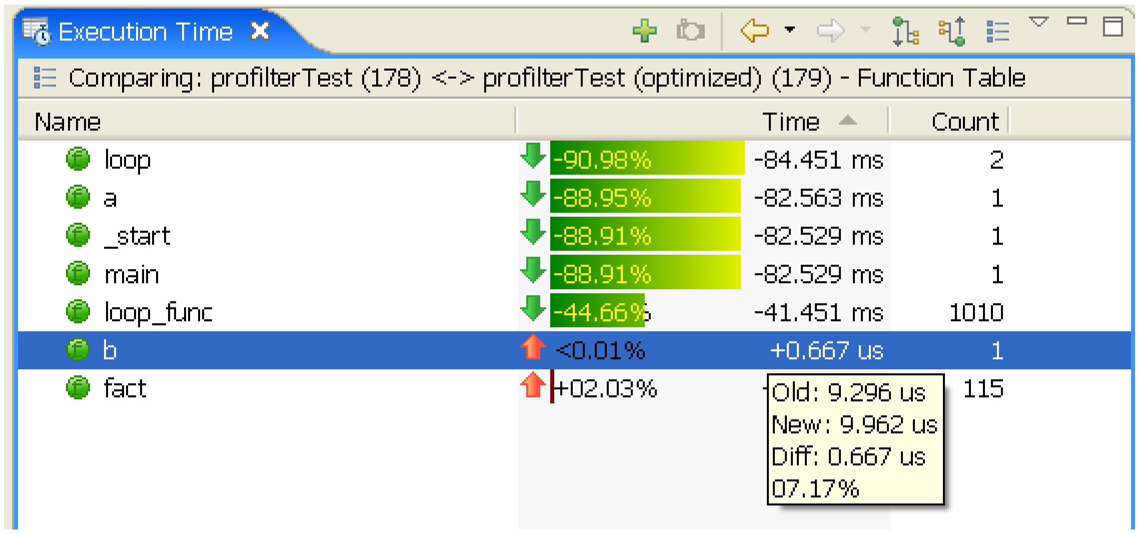
Profiler sessions are now managed in the Profiler Sessions view; separate from the Debug view. Sessions are persistent: saved when the eclipse application is closed, and restored when it is open again. The session view supports standard session management actions such as Delete, Rename, Open, Close, Session Comparison, Session Import, as well as actions applicable for running sessions only: Start/Stop profiling and Take Snapshot.



Profiler Sessions
profilterTest.x86 (Samping) #162 [22/08/07 3:44 PM]
profilterTest.x86 (Samping) - snapshot of 162 #163 [22/08,
profilterTest.x86 (Samping) #166 [22/0
profilterTest.x86 (Samping) #175 [23/0
profilterTest.x86 (Samping) #176 [23/0
demoMemoryProfiling #177 [24/08/07
Delete
Rename
Properties
Close
Compare
Import...

**Comparison Mode**

When you complete optimizing it is useful to know the progress you've made. The new comparison mode allows you easily to see the difference between two runs. You can continue to view data as a Call Tree or a Table - but instead of absolute time values, you would see time differences. Tooltips would provide actual time value as you need them.



Comparing: profilerTest (178) <-> profilerTest (optimized) (179) - Function Table

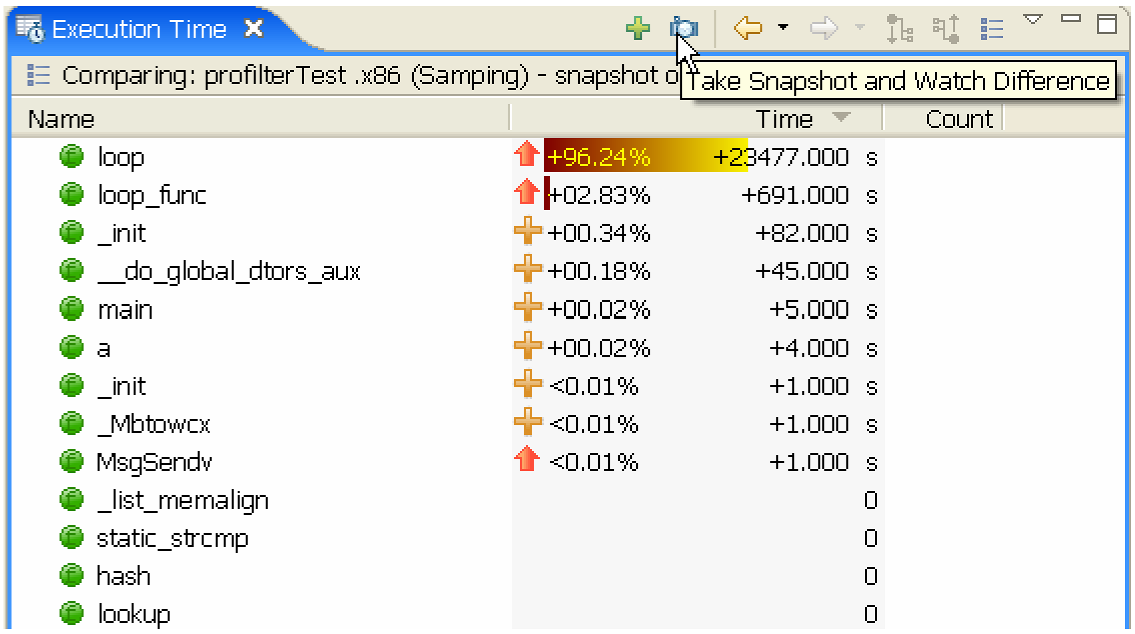
| Name | | Time | Count |
|---|---|---|---|
| loop | -90.98% | -84.451 ms | 2 |
| a | -88.95% | -82.563 ms | 1 |
| _start | -88.91% | -82.529 ms | 1 |
| main | -88.91% | -82.529 ms | 1 |
| loop_func | -44.66% | -41.451 ms | 1010 |
| b | <0.01% | +0.667 us | 1 |
| fact | +02.03% | | 115 |

Old: 9.296 us
New: 9.962 us
Diff: 0.667 us
07.17%

---

**Session Snapshots Tool**

The Take Snapshot feature lets you freeze the current state of the application profiler data, and while actual session data keeps changing, snapshot data remains frozen and can later be compared with the final results, or other snapshots of the same session. In the Execution Time View, this action also automatically switches to view a Comparison mode to dynamically show the updated difference between the current state and the snapshot.



Comparing: profilerTest .x86 (Samping) - snapshot o

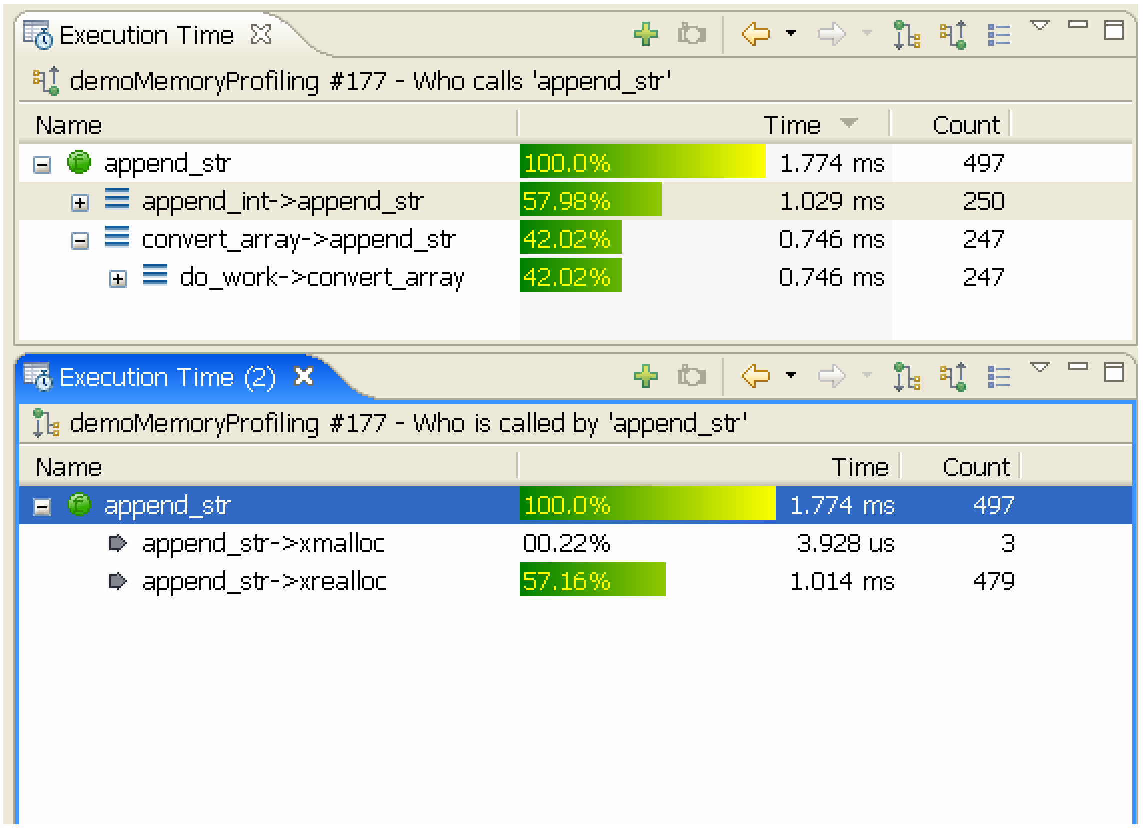
| Name | | Time | Count |
|---|---|---|---|
| loop | +96.24% | +23477.000 s | |
| loop_func | +02.83% | +691.000 s | |
| _init | +00.34% | +82.000 s | |
| __do_global_dtors_aux | +00.18% | +45.000 s | |
| main | +00.02% | +5.000 s | |
| a | +00.02% | +4.000 s | |
| _init | <0.01% | +1.000 s | |
| _Mbtowcx | <0.01% | +1.000 s | |
| MsgSendv | <0.01% | +1.000 s | |
| _list_memalign | | 0 | |
| static_strcmp | | 0 | |
| hash | | 0 | |
| lookup | | 0 | |

**Pause/Resume Profiling**

Occasionally, too much data is the same as having no data at all. You can take control of when to enable profiling during an application execution using the Pause/Resume Profiling actions.

**Cloning of the view page**

You can now create a second Execution Time View to see data side-by-side using the action Duplicate View. The new view is disconnected from Profiler Sessions, but keeps track of its own history.



Execution Time

demoMemoryProfiling #177 - Who calls 'append_str'

| Name | | Time | Count |
|---|---|---|---|
| append_str | 100.0% | 1.774 ms | 497 |
| append_int->append_str | 57.98% | 1.029 ms | 250 |
| convert_array->append_str | 42.02% | 0.746 ms | 247 |
| do_work->convert_array | 42.02% | 0.746 ms | 247 |

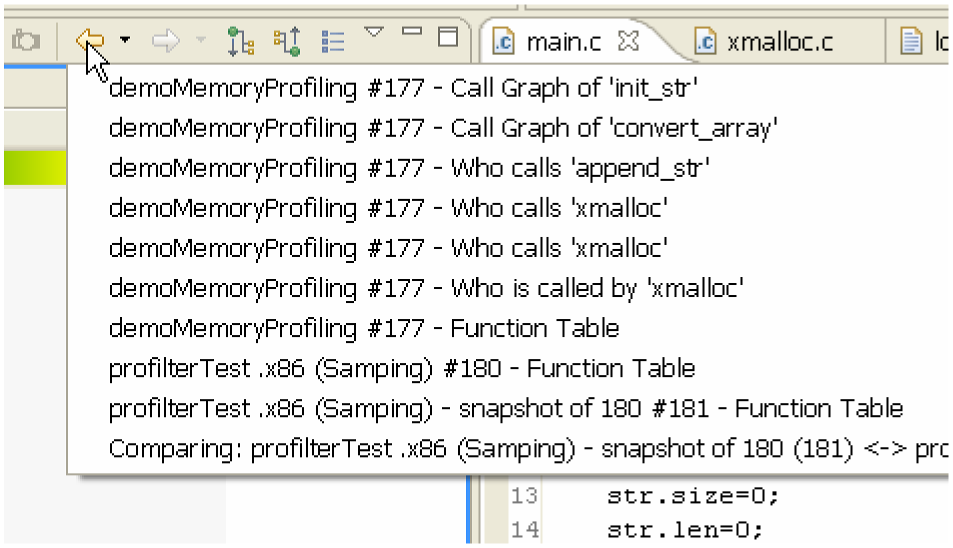
Execution Time (2)

demoMemoryProfiling #177 - Who is called by 'append_str'

| Name | | Time | Count |
|---|---|---|---|
| append_str | 100.0% | 1.774 ms | 497 |
| append_str->xmalloc | 00.22% | 3.928 us | 3 |
| append_str->xrealloc | 57.16% | 1.014 ms | 479 |

**History Navigation**

The Execution Time View keeps track and maintains a record of where have been. You can go back and forward right from the toolbar, or select a particular entry in the navigation history. The navigation history size is customizable in the view Preferences.

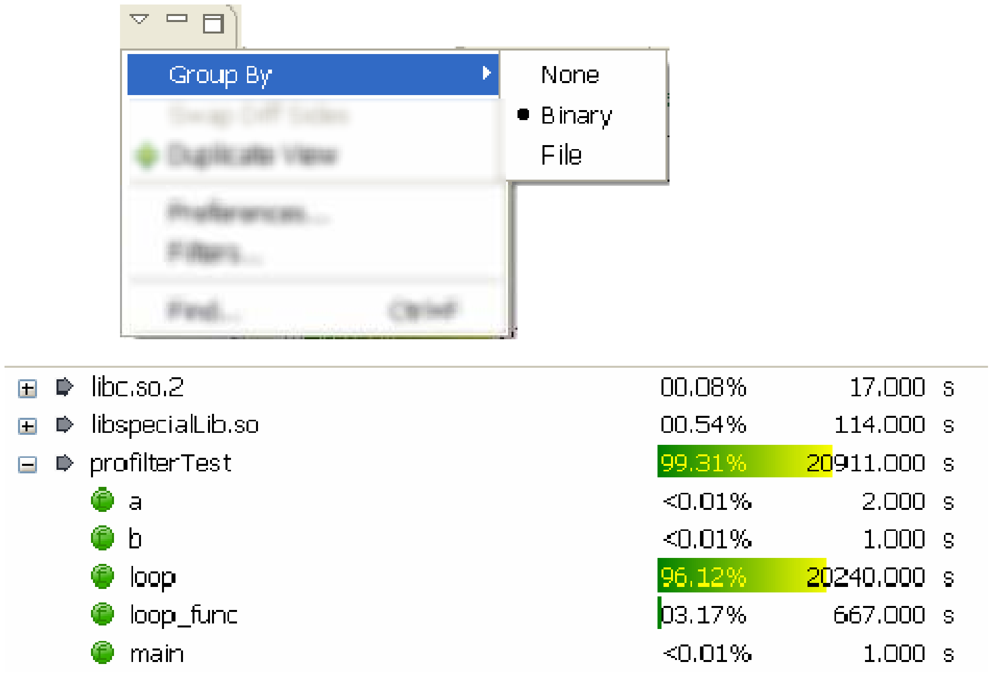


demoMemoryProfiling #177 - Call Graph of 'init_str'
demoMemoryProfiling #177 - Call Graph of 'convert_array'
demoMemoryProfiling #177 - Who calls 'append_str'
demoMemoryProfiling #177 - Who calls 'xmalloc'
demoMemoryProfiling #177 - Who calls 'xmalloc'
demoMemoryProfiling #177 - Who is called by 'xmalloc'
demoMemoryProfiling #177 - Function Table
profilterTest .x86 (Samping) #180 - Function Table
profilterTest .x86 (Samping) - snapshot of 180 #181 - Function Table
Comparing: profilterTest .x86 (Samping) - snapshot of 180 (181) <-> prc

```
str.size=0;
str.len=0;
```

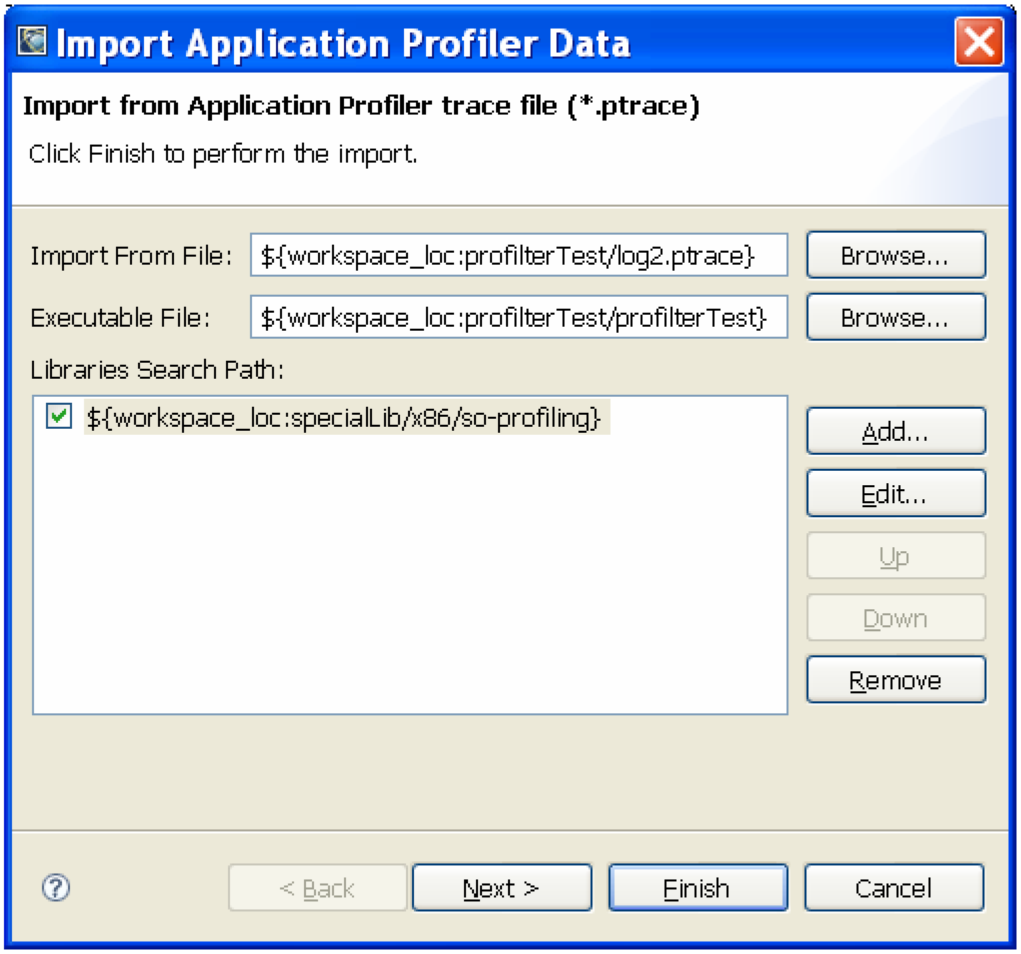
**Grouping**

The Grouping feature helps for the organization of huge function tables for better navigation and analysis. This is the easy way to see aggregated time results for each software component (binary or file).



| | | |
|---|---|---|
| libc.so.2 | 00.08% | 17.000 s |
| libspecialLib.so | 00.54% | 114.000 s |
| profilterTest | 99.31% | 20911.000 s |
| a | <0.01% | 2.000 s |
| b | <0.01% | 1.000 s |
| loop | 96.12% | 20240.000 s |
| loop_func | 03.17% | 667.000 s |
| main | <0.01% | 1.000 s |

**Session Import** Now, you can import from .gmon, .kev or .ptrace files, generated on the target side by Profiler, using the new Import action right from the session view, or using the standard Eclipse Import action.


Import Application Profiler Data

Import from Application Profiler trace file (*.ptrace)

Click Finish to perform the import.

Import From File: ${workspace_loc:profilterTest/log2.ptrace}

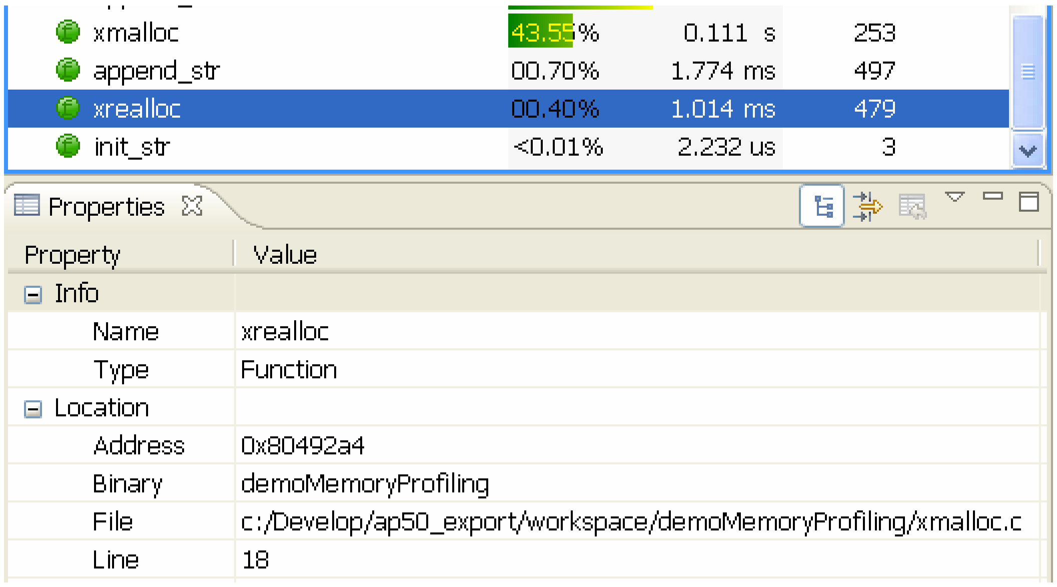
Executable File: ${workspace_loc:profilterTest/profilterTest}

Libraries Search Path:

✓ ${workspace_loc:specialLib/x86/so-profiling}

**Properties View Integration**

It is now easy to obtain additional information about a view element. Simply select an element, and the standard Eclipse Properties view will show element properties:



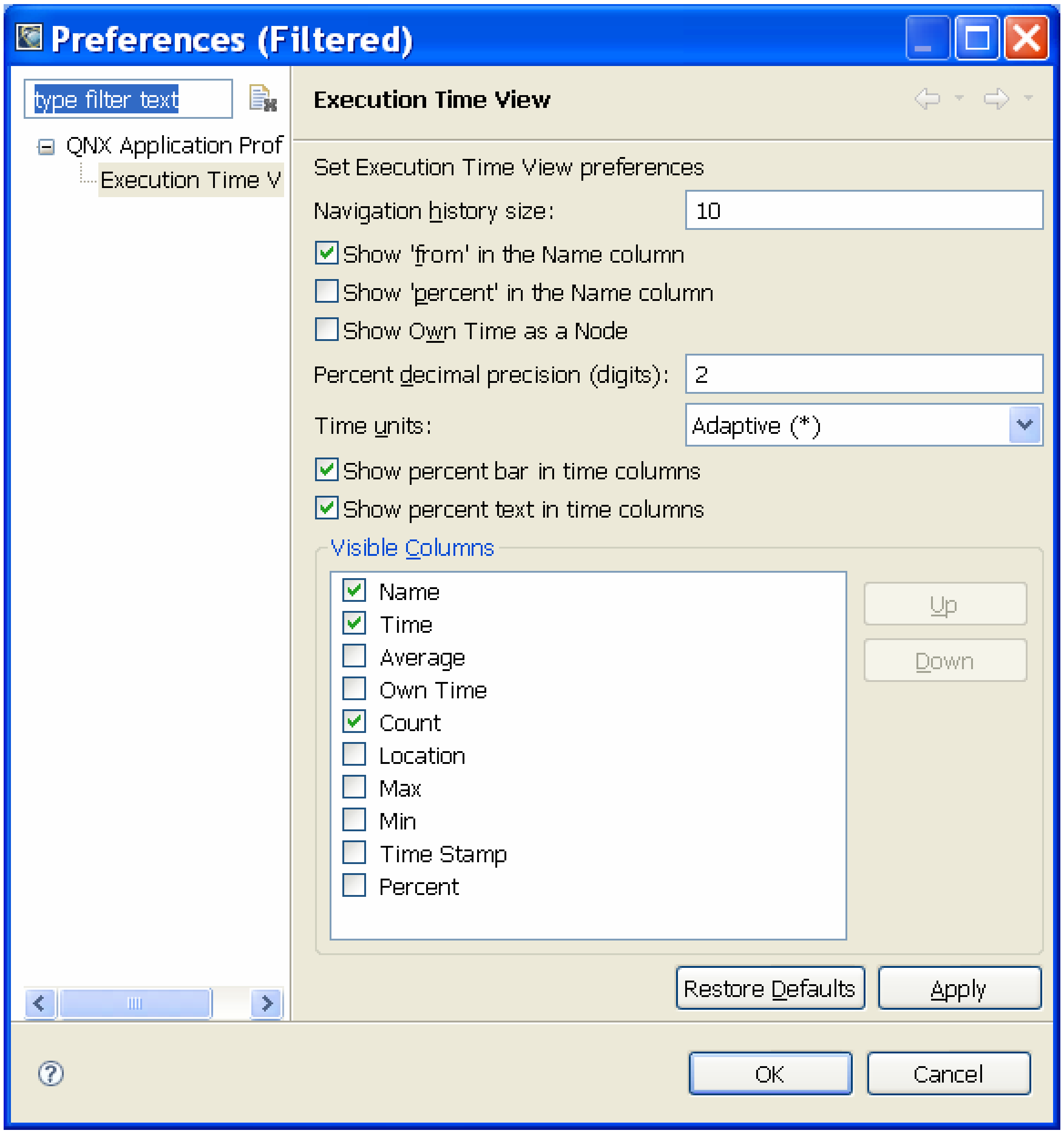
| | | | |
|---|---|---|---|
| xmalloc | 43.55% | 0.111 s | 253 |
| append_str | 00.70% | 1.774 ms | 497 |
| xrealloc | 00.40% | 1.014 ms | 479 |
| init_str | <0.01% | 2.232 us | 3 |

Properties

| Property | Value |
|---|---|
| Info | |
| Name | xrealloc |
| Type | Function |
| Location | |
| Address | 0x80492a4 |
| Binary | demoMemoryProfiling |
| File | c:/Develop/ap50_export/workspace/demoMemoryProfiling/xmalloc.c |
| Line | 18 |

**View Customization**

Now, you can use the Execution Time View Preference Page to customize the number of columns you want to have in the view, their order and the format of the data they show.



Preferences (Filtered)

type filter text

QNX Application Prof
Execution Time V

Execution Time View

Set Execution Time View preferences

Navigation history size: 10

- [✓] Show 'from' in the Name column
- [ ] Show 'percent' in the Name column
- [ ] Show Own Time as a Node

Percent decimal precision (digits): 2

Time units: Adaptive (*)

- [✓] Show percent bar in time columns
- [✓] Show percent text in time columns

Visible Columns

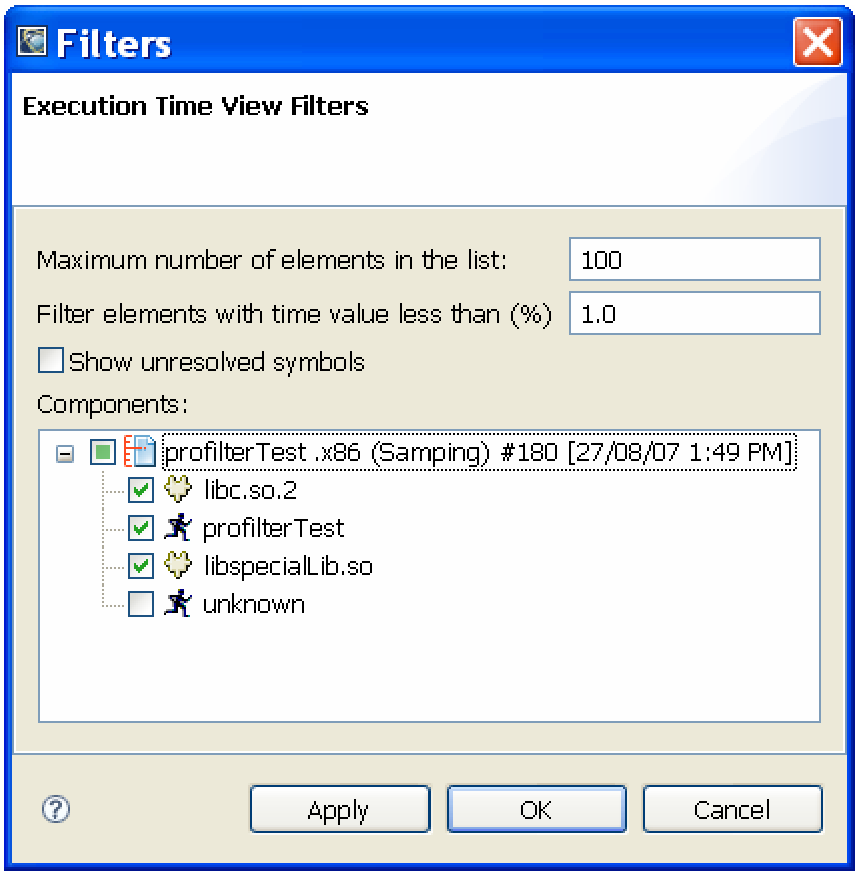
- [✓] Name
- [✓] Time
- [ ] Average
- [ ] Own Time
- [✓] Count
- [ ] Location
- [ ] Max
- [ ] Min
- [ ] Time Stamp
- [ ] Percent

**Copy To Clipboard**

At any time, if you want to see the table or tree data in text format, use the platform standard copy method to obtain the text version of the visible data, which will be copied to your clipboard. Paste it in e-mail and share with your manager your success in software optimization.

**Filtering**

When grouping does not help, you can use filters to remove some rows from the table. Component filtering lets you see only these records related to the component, or you can use Data filtering to filter based on timing values.



Filters

Execution Time View Filters

Maximum number of elements in the list: 100

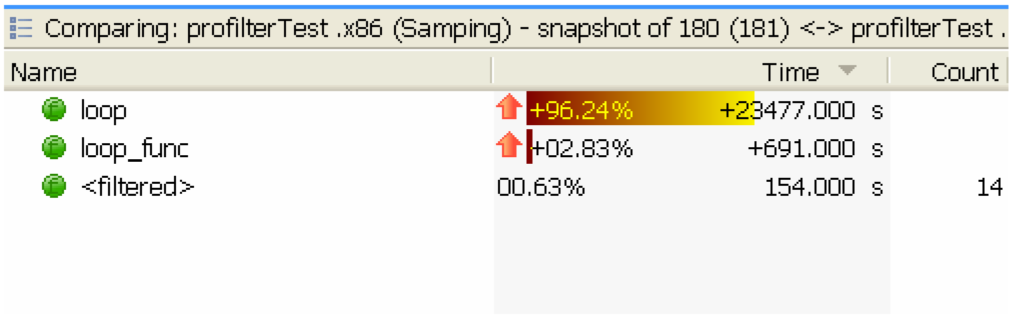
Filter elements with time value less than (%) 1.0

☐ Show unresolved symbols

Components:

- profilterTest .x86 (Samping) #180 [27/08/07 1:49 PM]
  - ✓ libc.so.2
  - ✓ profilterTest
  - ✓ libspecialLib.so
  - ☐ unknown

When filtering is applied, the "<filtered>" element remains in the view as a remainder of filtered elements, and the total number of these elements is visible in the Count column.

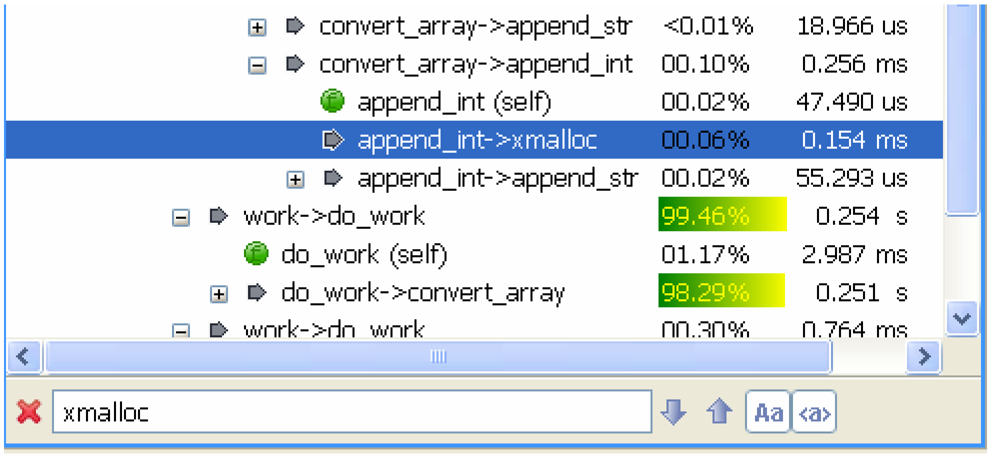


Comparing: profilterTest .x86 (Samping) - snapshot of 180 (181) <-> profilterTest .

| Name | Time | | Count |
|---|---|---|---|
| loop | +96.24% | +23477.000 s | |
| loop_func | +02.83% | +691.000 s | |
| <filtered> | 00.63% | 154.000 s | 14 |

**Search**

The new "Find..." action includes an easy to use Find bar that will pop up at the bottom of the view. The view automatically expands and highlights nodes in the tree with given text when a string is located.

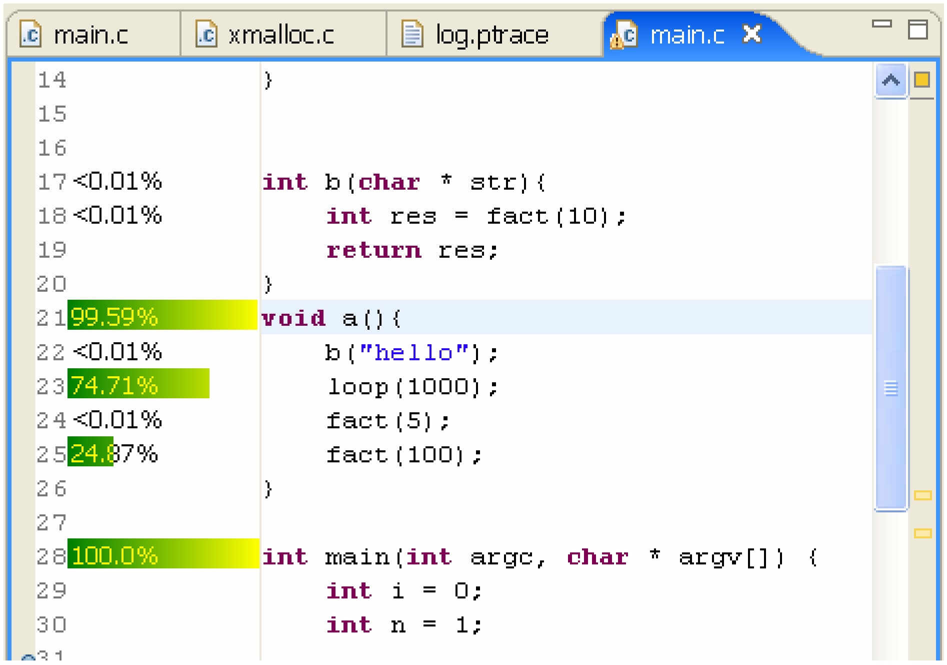


```
⊞ convert_array->append_str      <0.01%    18.966 us
⊟ convert_array->append_int      00.10%     0.256 ms
      append_int (self)          00.02%    47.490 us
      append_int->xmalloc        00.06%     0.154 ms
  ⊞ append_int->append_str       00.02%    55.293 us
⊟ work->do_work                  99.46%     0.254  s
    do_work (self)               01.17%     2.987 ms
  ⊞ do_work->convert_array       98.29%     0.251  s
⊟ work->do_work                  00.30%     0.764 ms
```

---

**Annotated Editor Improvements**

Editor annotations now have the same look and feel as table annotations; they have text and can be customized using preferences.



```
           }


<0.01%     int b(char * str){
<0.01%         int res = fact(10);
               return res;
           }
99.59%     void a(){
<0.01%         b("hello");
74.71%         loop(1000);
<0.01%         fact(5);
24.87%         fact(100);
           }

100.0%     int main(int argc, char * argv[]) {
               int i = 0;
               int n = 1;
```