
Eclipse Tracing with GDB

Tracing Mid-Project Meeting

2010-12-09

marc.khouzam@ericsson.com

CDT DSF/GDB Component Lead

Summary

- › GDB Tracepoints in Eclipse
 - Setting up the tracepoints
 - Tracing the application/system
 - Visualizing the collected data

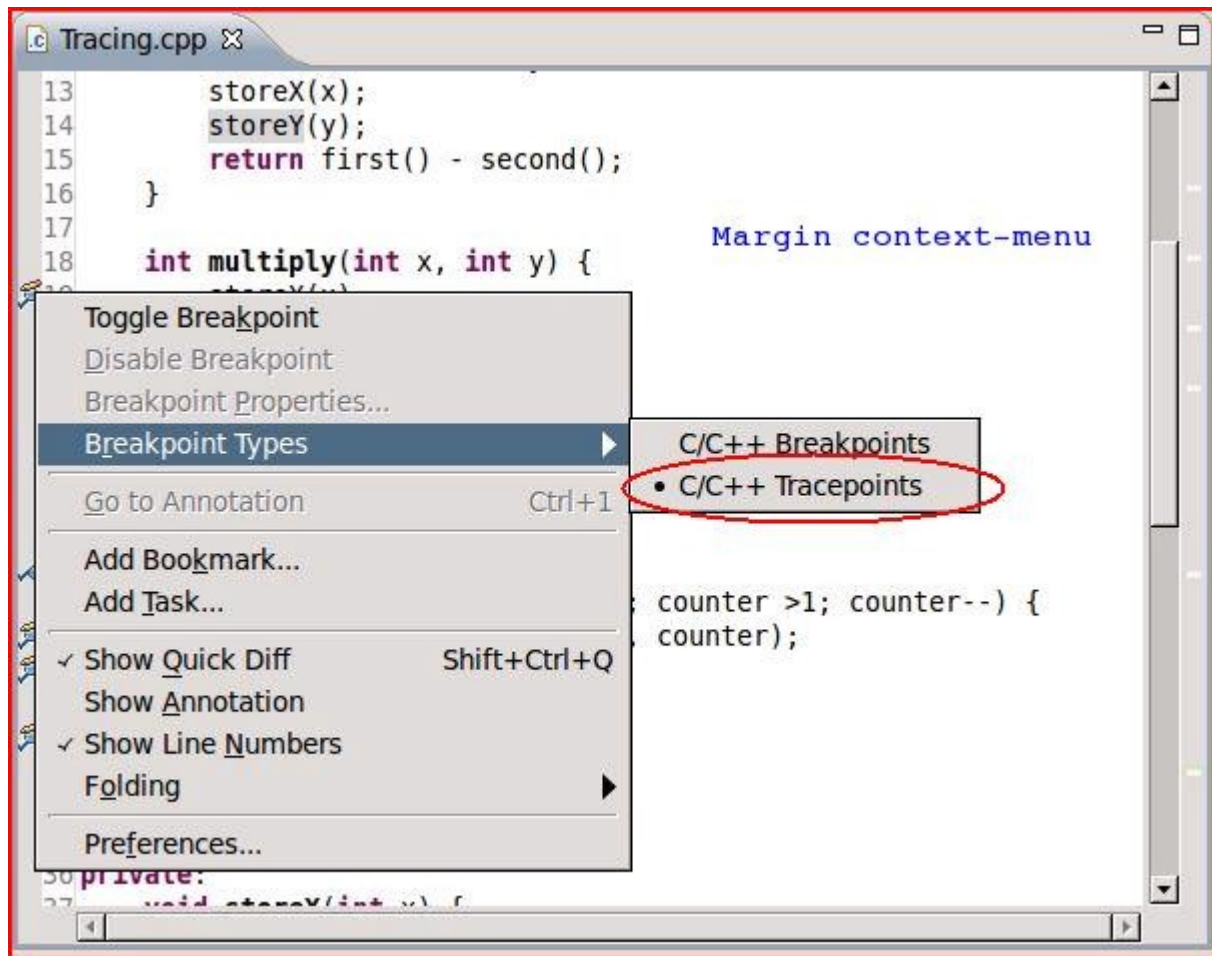
- › Upcoming features

Eclipse Tracepoints

- › Creation of tracepoint as is done as for breakpoints
- › Enable/Disable
- › Dynamic condition
- › Specification of data to be gathered using symbolic expressions and memory addresses (actions)
- › Trace-state variables can be used in conditions and actions
- › Passcount: stopping tracing after the Nth hit

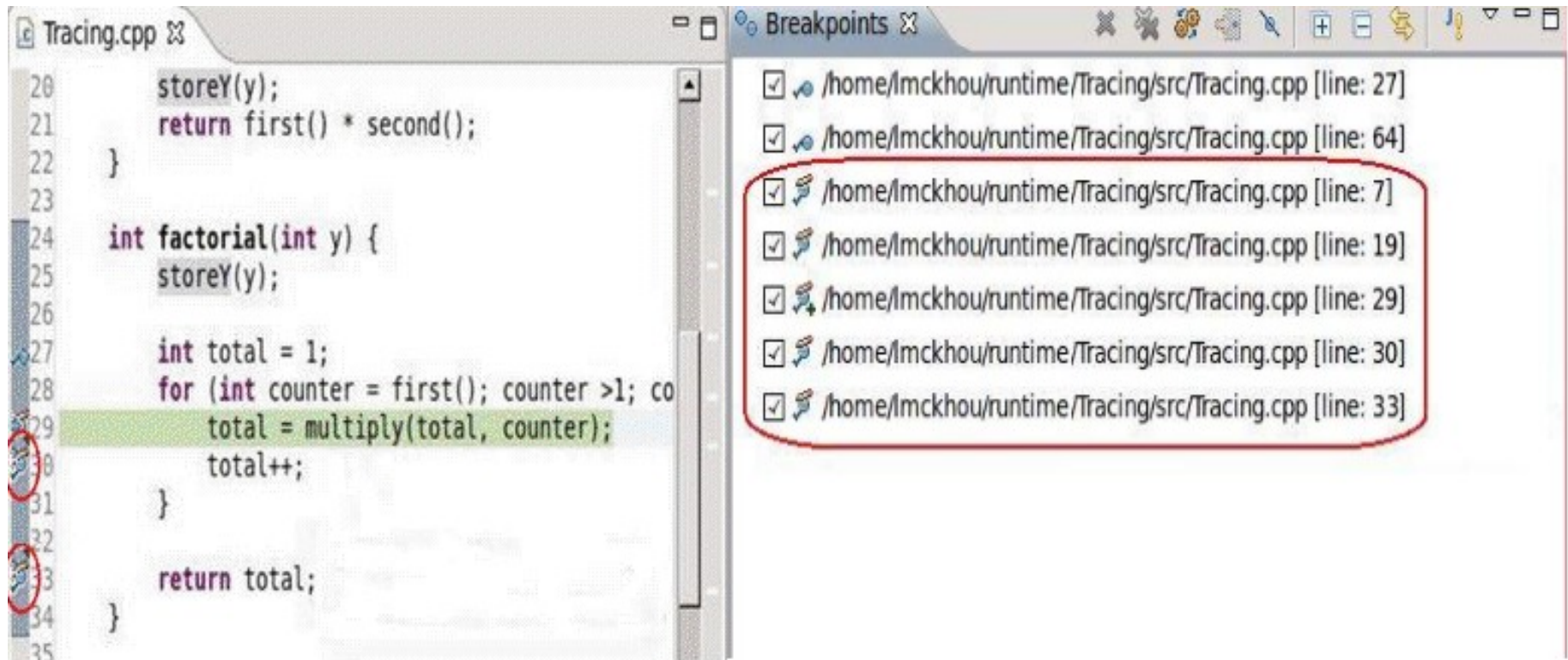
Eclipse Tracepoints Selection

- › Tracepoints treated as breakpoints



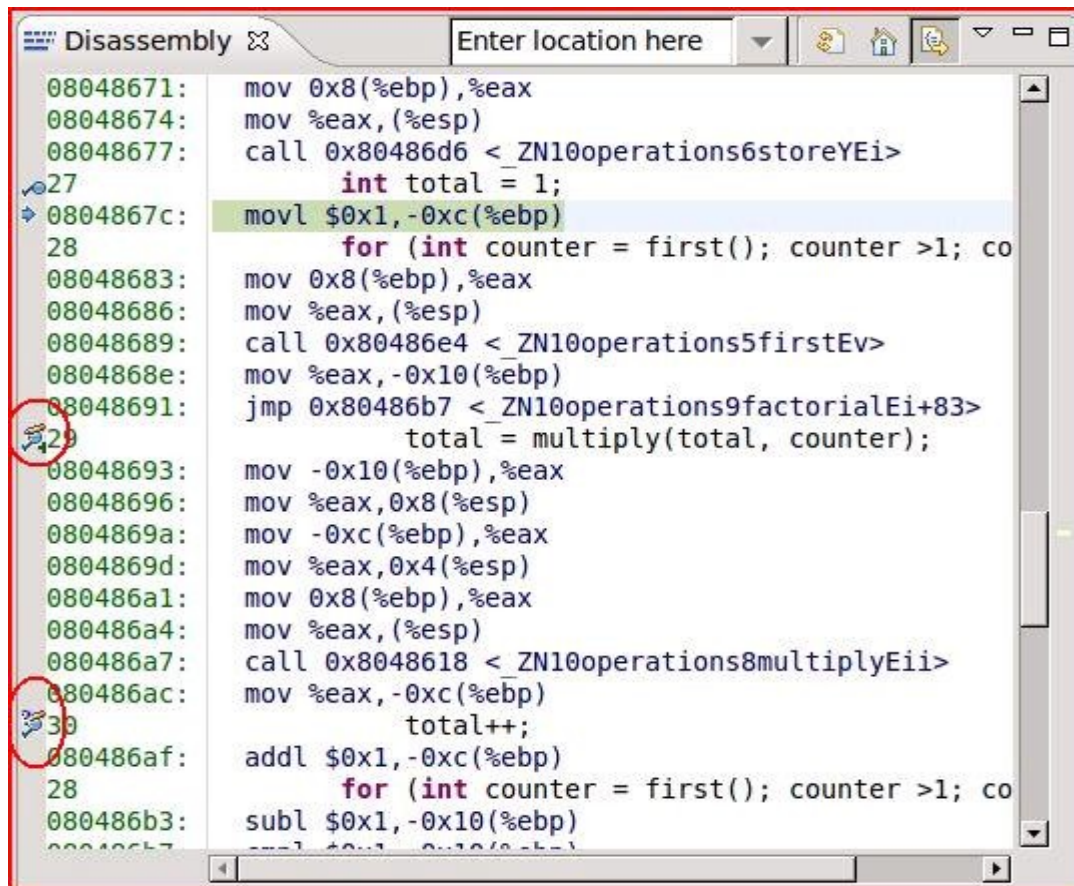
Eclipse Tracepoints Display

- › Tracepoints
- › Tracepoints with actions



Eclipse Tracepoints Disassembly

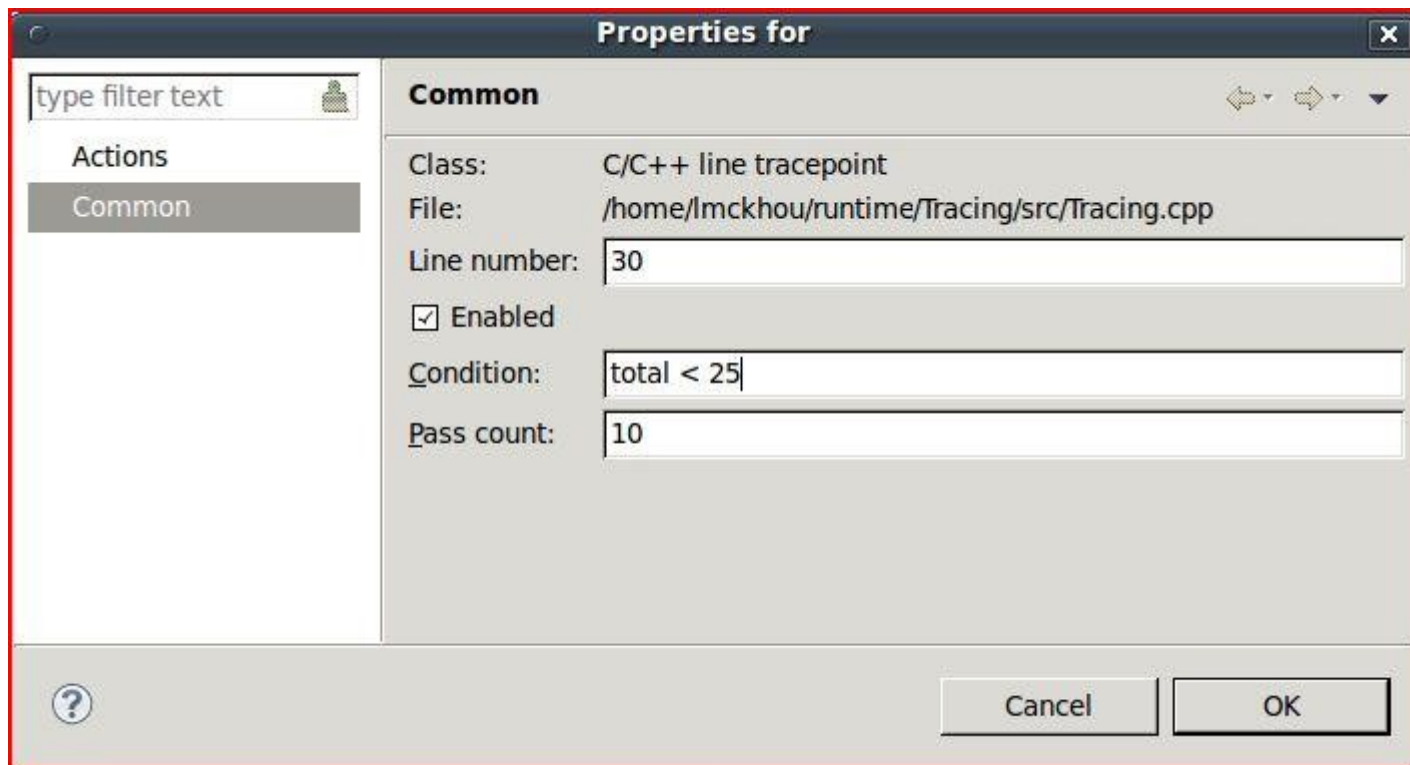
- › Disassembly view support for Tracepoints
- › Tracepoint with condition



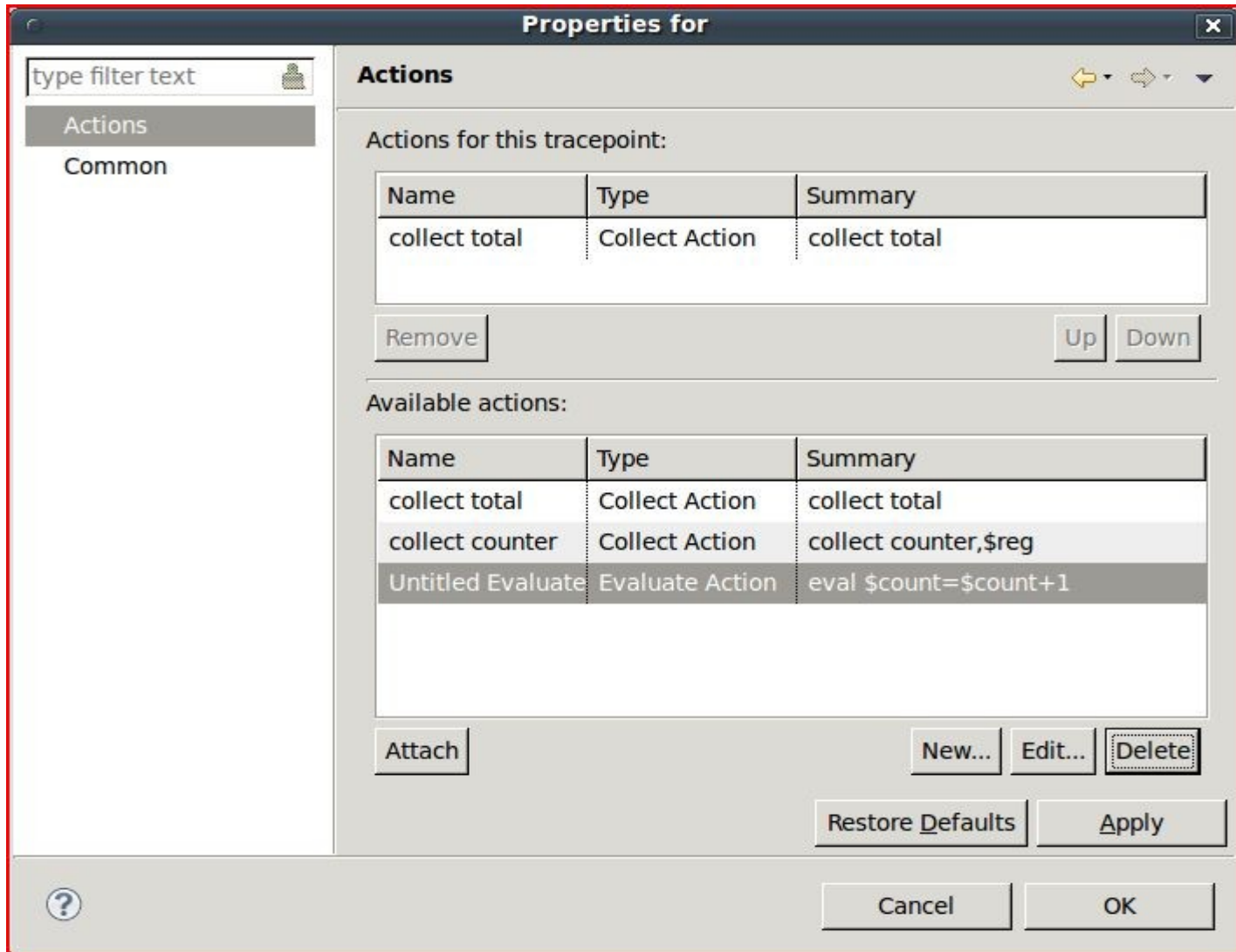
The screenshot shows the Eclipse IDE's Disassembly view. The window title is "Disassembly" and it contains a search bar "Enter location here". The assembly code is displayed in a list format with addresses on the left and instructions on the right. A red circle highlights the instruction at address 08048691: `jmp 0x80486b7 <_ZN10operations9factorialEi+83>`. Another red circle highlights the instruction at address 080486ac: `mov %eax, -0xc(%ebp)`. The code includes comments in C++ style, such as `int total = 1;` and `for (int counter = first(); counter > 1; counter++)`. The instructions are: `08048671: mov 0x8(%ebp), %eax`, `08048674: mov %eax, (%esp)`, `08048677: call 0x80486d6 <_ZN10operations6storeYEi>`, `27: int total = 1;`, `0804867c: movl $0x1, -0xc(%ebp)`, `28: for (int counter = first(); counter > 1; counter++)`, `08048683: mov 0x8(%ebp), %eax`, `08048686: mov %eax, (%esp)`, `08048689: call 0x80486e4 <_ZN10operations5firstEv>`, `0804868e: mov %eax, -0x10(%ebp)`, `08048691: jmp 0x80486b7 <_ZN10operations9factorialEi+83>`, `29: total = multiply(total, counter);`, `08048693: mov -0x10(%ebp), %eax`, `08048696: mov %eax, 0x8(%esp)`, `0804869a: mov -0xc(%ebp), %eax`, `0804869d: mov %eax, 0x4(%esp)`, `080486a1: mov 0x8(%ebp), %eax`, `080486a4: mov %eax, (%esp)`, `080486a7: call 0x8048618 <_ZN10operations8multiplyEii>`, `080486ac: mov %eax, -0xc(%ebp)`, `30: total++;`, `080486af: addl $0x1, -0xc(%ebp)`, `28: for (int counter = first(); counter > 1; counter++)`, `080486b3: subl $0x1, -0x10(%ebp)`, `080486b7: movl $0x1, -0x10(%ebp)`.

Eclipse Tracepoints Properties

- › Tracepoints properties
 - Location
 - Enablement
 - Condition
 - Pass count

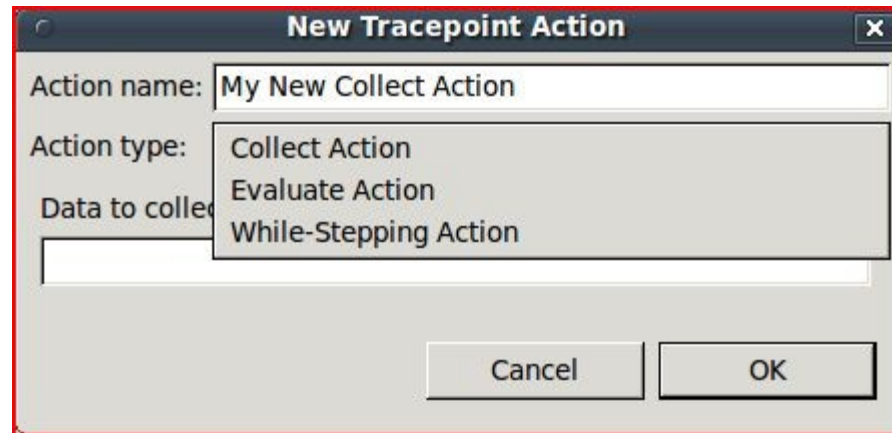


Eclipse Tracepoints Actions

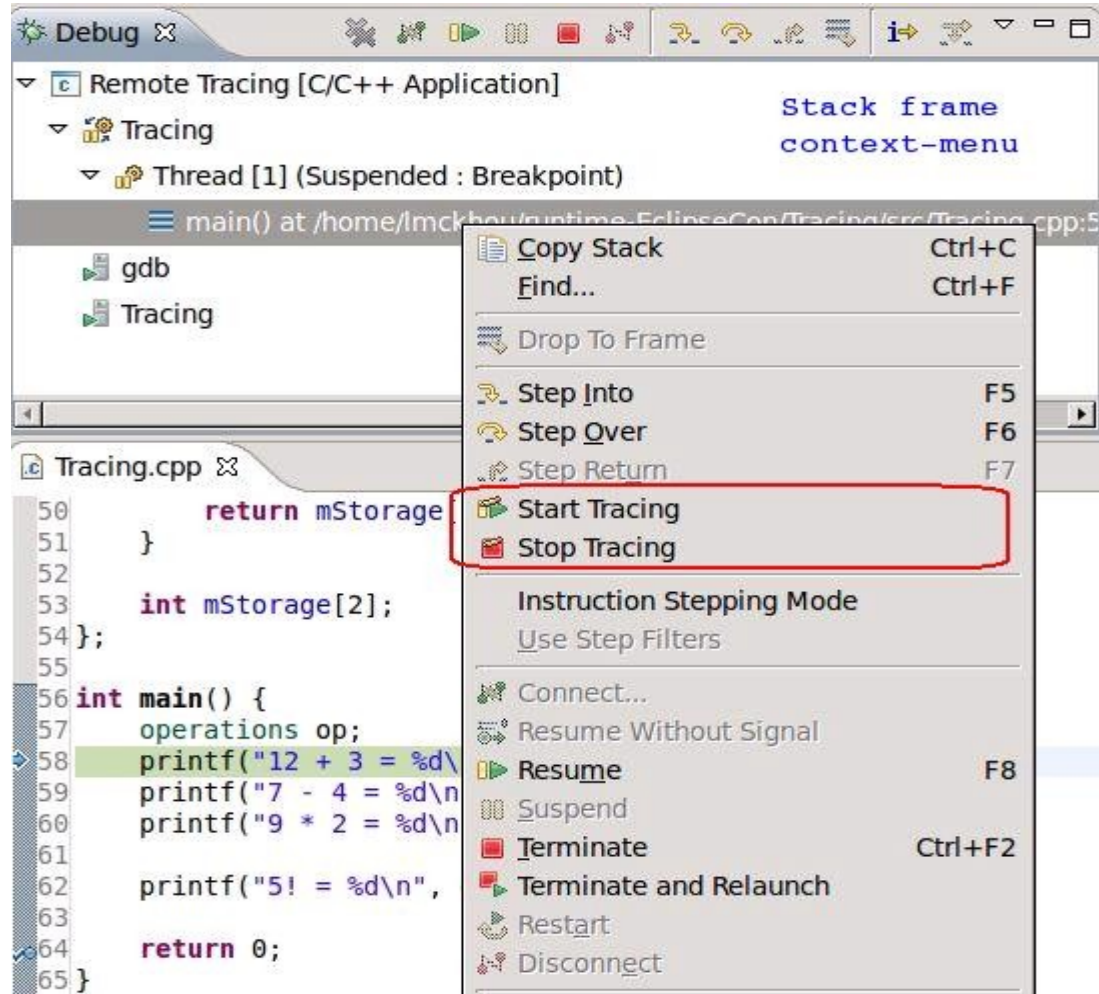


Eclipse Tracepoints Actions

- › Tracepoints action types
 - Collect
 - Evaluate
 - While-Stepping
 - › Collect
 - › Evaluate

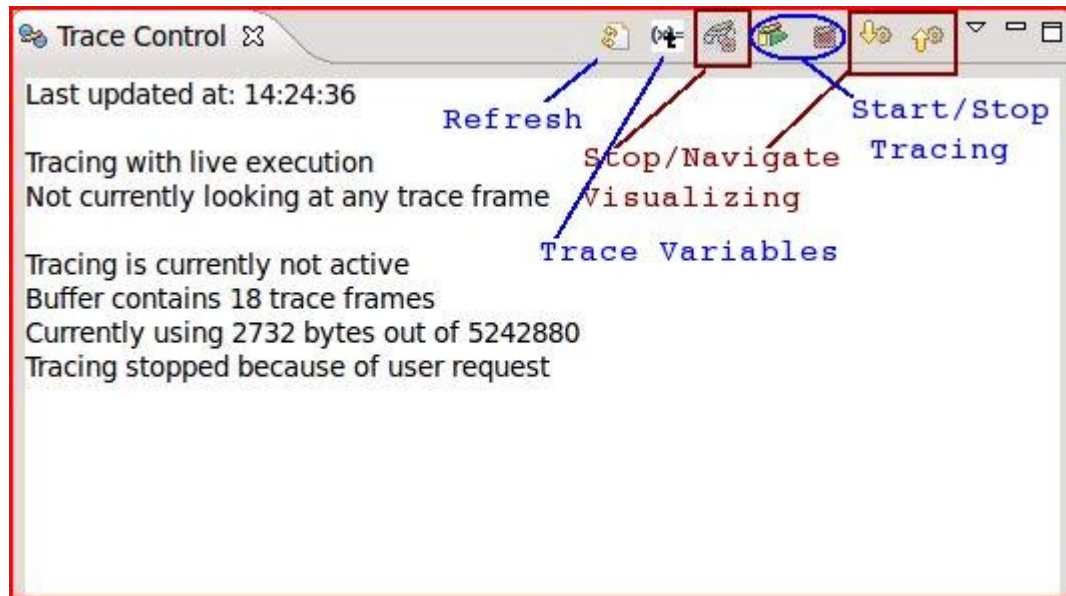


Eclipse Tracepoints Control

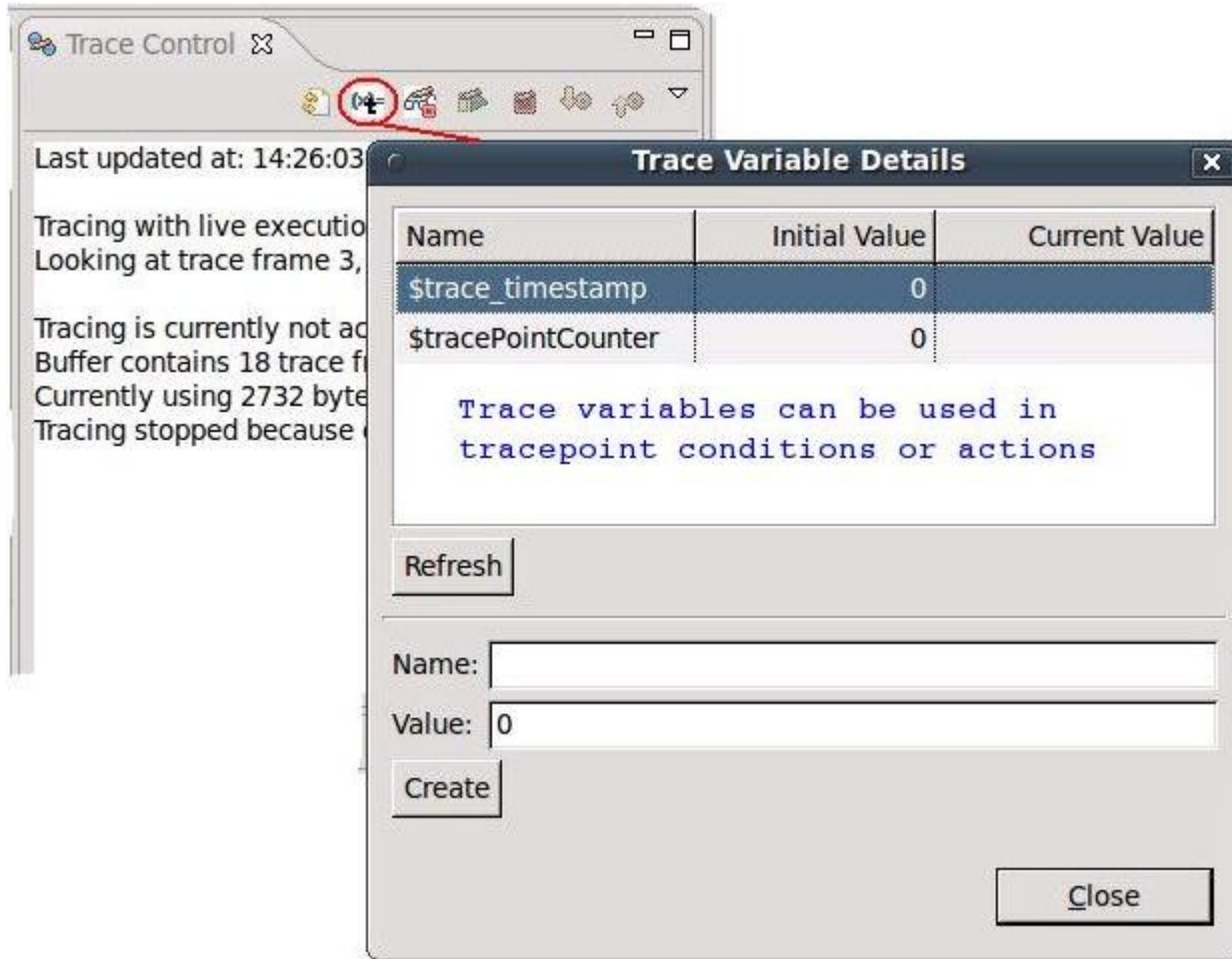


Eclipse Tracepoints Control

- › Trace Control View
 - Refreshing info
 - Trace Variables
 - Start/Stop Tracing
 - Navigate during Visualization
 - Stop Visualization

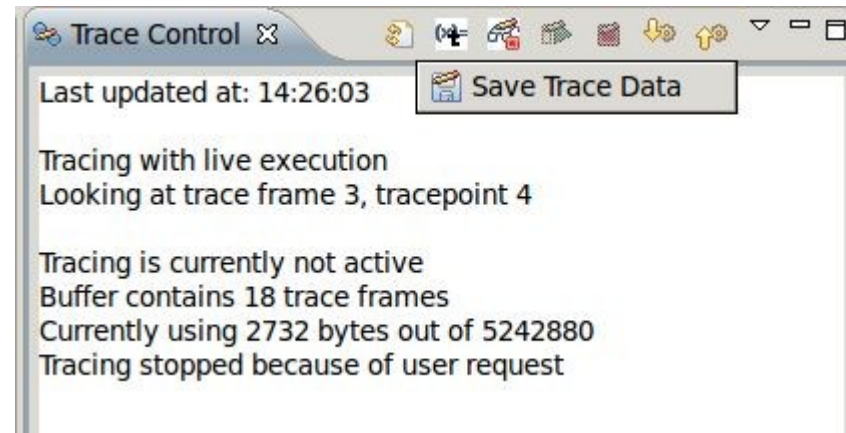


Eclipse Tracepoints Variables



Eclipse Trace Data

- › Resulting trace data
 - can be stored to file
 - can be visualized in Eclipse immediately or in the future



Eclipse Trace Data Visualization

- › Each data record is a snapshot of debug information
- › Records can be examined using standard debugger views
 - As if debugger was attached at a specific point in time
 - Only collected information can be shown
- › All collected data of a record can also be dumped as plain text

Eclipse Trace Data Visualization

The screenshot displays the Eclipse IDE interface during a debug session. The main window shows the source code of `NonStop.cpp` with a breakpoint at line 44. The `Variables` view on the right shows the state of variables at the breakpoint. The `Trace Control` view at the bottom right shows the current trace configuration.

Collected data (Variables view):

Name	Type	Value
i	int	3
thread3	pthread_t	3066370928
thread	unsigned long [30]	0xbfcc88ac
message3	char *	0x8048821 "Thread 3"
iret3	int	0
thread2	pthread_t	3074763632
message2	char *	0x8048818 "Thread 2"
iret2	int	0

Line where trace was collected (Source code view):

```
40  iret2 = pthread_create( &thread2, NULL, thread_exec1, (void*) message2);
41  iret3 = pthread_create( &thread3, NULL, thread_exec2, (void*) message3);
42
43  for (int i=0;i<15;i++) {
44  printf("another loop\n");
45  sleep(1);
46  pthread_create( &thread[i], NULL, thread_exec3, (void*) &i);
47  }
48  /* Wait till threads are complete before main continues. Unless we */
49  /* wait we run the risk of executing an exit which will terminate */
50  /* the process and all threads before the threads have completed. */
51
52  pthread_join(thread2, NULL);
53  printf("Thread 2 returns: %d\n", iret2);
54
55  pthread_join(thread3, NULL);
56  printf("Thread 3 returns: %d\n", iret3);
57
58  return 0;
59 }
```

Tracepoint that collected data (Breakpoints view):

- NonStop.cpp [line: 44] [condition: i < 10]

Trace Control (Trace Control view):

Last updated at: 09:53:29

Tracing with live execution
Looking at trace frame 4, tracepoint 1

Tracing is currently not active
Buffer contains 11 trace frames
Currently using 6090 bytes out of 5242880
Tracing stopped because of user request

Planned Tracepoint Features

- › Support for Static Tracepoints (GDB/UST)
- › Support for Fast Tracepoints
 - Explicit or implicit support?
- › Support for Observer mode
- › Support for Global Actions (affecting all tracepoints)

Planned Tracepoint Features

- › Enable/Disable Tracepoints *during* Tracing
- › Tracepoints Enhanced Visualization:
 - Currently the user must have an idea of what has been collected
 - Goal is to directly and only show what has been collected
- › Fast Tracepoints on 3-byte instruction
 - Currently fast tracepoints are 5-byte jumps insert in the code
 - New 3-byte jump to a nearby location to the 5-byte jump

Planned Tracepoint Features

- › Pending Tracepoints (for dynamically loaded code)
- › Thread-specific Tracepoints
- › Generalization of Tracepoints
 - Hardware Tracepoints
 - Tracing Watchpoints
 - Tracing Catchpoints

Other Planned Features of Interest

- › Tracepoints technology is being extended to breakpoints
 - Breakpoint conditions evaluated on the target
 - Byte-code translation
 - (Trace) State variables extended to breakpoints

- › Global breakpoints
 - Setting a breakpoint for any process of the system
 - Can even affect future processes
 - Great for short-lived processes, or debugging start-up sequence

Conclusion

- › Can perform the entire tracing scenario in Eclipse
 - Setup
 - Trace
 - Visualize

- › Many enhancements planned

Questions?



ERICSSON