



ERICSSON

A decorative graphic on the left side of the slide, consisting of a cluster of circles in various colors (green, cyan, blue, purple, pink) arranged in a roughly circular pattern, with some circles having a soft shadow effect.

MAKING YOUR DEBUGGING
EFFORTS COUNT:
BEST PRACTICES WITH THE CDT DEBUGGER

Marc Khouzam

ABOUT ME

- Working with CDT Debug since 2007
- CDT project co-lead, lead for Debug component
- Things you don't like about CDT Debug are probably my fault
- You can help get them improved
 - Give feedback
 - Open bugs
 - Contribute



AGENDA

- › Running the debugger
- › No more *Printf-debugging*
- › Examining debugging data
- › Controlling execution
- › Multi-thread and beyond
- › More advanced topics
- › Future plans

And one demo chosen
by YOU



ERICSSON



RUNNING THE DEBUGGER



STAND-ALONE DEBUGGER

- › Easy installation through its own package
 - <https://eclipse.org/cdt/downloads.php>
 - <https://wiki.eclipse.org/CDT/StandaloneDebugger>



CDT also provides its debugger as a stand-alone RCP application which can be downloaded and installed on its own, from: <http://download.eclipse.org/tools/cdt/releases/8.8.1/r/rcp>

The RCP stand-alone debugger has its own p2 software repository to be used within the RCP itself to upgrade it: <http://download.eclipse.org/tools/cdt/releases/8.8.1/r/rcp-repository>



ERICSSON

STAND-ALONE DEBUGGER



› Easy launch: `./cdtdebug -e myBinary`

The screenshot displays the Eclipse C/C++ Stand-alone Debugger interface. The main window shows the program's execution state, including the current thread and the function being executed. The Variables panel shows the current values of variables `i` and `max`. The Console panel shows the output of the program, which is printing the value of `i` from 36 to 42.

Eclipse C/C++ Stand-alone Debugger

File Edit Search Run Window Help

Debug

home_lmckhou_testing_loopfirst [C/C++ Application]

- loopfirst [22889] [cores: 0]
 - Thread #1 [loopfirst] 22889 [core: 0] (Suspended : Step)
 - main() at loopfirst.cc:10 0x400570
 - gdb (7.11)

(x)= Variables

Name	Type	Value
(x)= i	int	42
(x)= max	int	900

loopfirst.cc

```
2 #include <stdio.h>
3
4 int j = 0;
5
6 int main() {
7     int max = 900;
8     for (int i=0;i<max;i++) {
9         printf("i is %d\n",i);
10        sleep(1);
11    }
12    return 0;
13 }
```

home_lmckhou_testing_loopfirst [C/C++ Application] loopfirst

```
i is 36
i is 37
i is 38
i is 39
i is 40
i is 41
i is 42
```



ERICSSON



POST-MORTEM DEBUG

- › Examining a core file: Variables, Registers, Memory
- › `./cddtdebug -c coreFile -e matchingBinary`

The screenshot displays the Visual Studio debugger interface for a C/C++ Postmortem Debugger session. The main window shows the source code for `DSFTTestApp.cpp` with a breakpoint set at `dup();`. The `Debug` window shows the call stack with `main()` at `DSFTTestApp.cpp:244`. The `Variables` window lists local variables: `argc` (int, 1), `argv` (char **, 0x7fffffffdf18), `b` (int [5], 0x7fffffffdcc0), `um` (wchar_t, 32767 'L'翻'), `array_small` (int [4], 0x7fffffffdcf0), and `de` (int, 0). The `Registers` window shows the state of CPU registers, including `rbp` (0x7fffffffde30), `rsp` (0x7fffffff1a0), `r8` (4294967295), `r9` (0), `r10` (34), `r11` (582), `r12` (4199104), and `r13` (140737488346896).

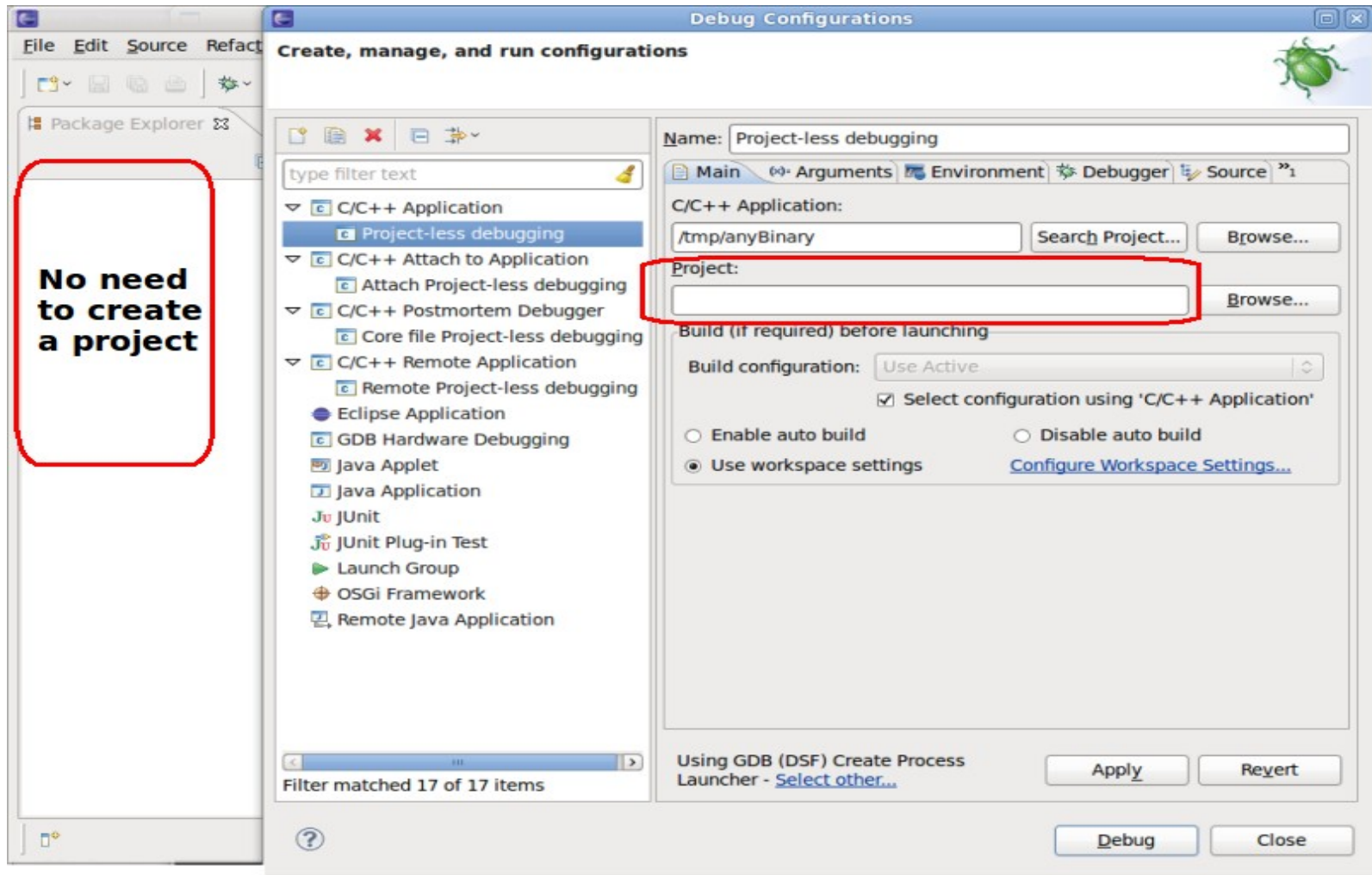
Name	Type	Value
argc	int	1
argv	char **	0x7fffffffdf18
b	int [5]	0x7fffffffdcc0
um	wchar_t	32767 'L'翻'
array_small	int [4]	0x7fffffffdcf0
de	int	0

Name	Value	Descript
rbp	0x7fffffffde30	
rsp	0x7fffffff1a0	
r8	4294967295	
r9	0	
r10	34	
r11	582	
r12	4199104	
r13	140737488346896	

PROJECT-LESS DEBUG



› Debug any binary!



The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer shows a tree view of debug configurations. A red box highlights the text "No need to create a project" next to the "Project-less debugging" configuration. The main window is the "Debug Configurations" dialog, titled "Create, manage, and run configurations". The "Name" field is "Project-less debugging". The "C/C++ Application" field is "/tmp/anyBinary". The "Project" field is empty and highlighted with a red box. The "Build (if required) before launching" section is expanded, showing "Build configuration" set to "Use Active" and "Use workspace settings" selected. At the bottom, there are "Apply", "Revert", "Debug", and "Close" buttons. A status bar at the bottom indicates "Using GDB (DSF) Create Process Launcher - Select other...".

No need to create a project

Debug Configurations

Create, manage, and run configurations

Name: Project-less debugging

C/C++ Application: /tmp/anyBinary

Project:

Build (if required) before launching

Build configuration: Use Active

Select configuration using 'C/C++ Application'

Enable auto build Disable auto build

Use workspace settings [Configure Workspace Settings...](#)

Using GDB (DSF) Create Process Launcher - [Select other...](#)

Apply Revert

Debug Close



ERICSSON



NO MORE PRINTF-DEBUGGING

PRINTF DEBUGGING

- ◆ Still much too popular
 - Comfortable, familiar, easy

- ◆ Costly efficiency limitations
 - Expensive debug cycle
 - 1.Recompiling
 - 2.Redeploying to target
 - 3.Repeating steps to reproduce issue
 - Info provided is fixed per debug cycle
 - Multiple such debug cycles

DYNAMIC-PRINTF

› Familiarity meets flexibility and efficiency!



- › Printf dynamically inserted by debugger in executing program
- › Prints in same location as compiled-printfs
- › Same syntax as compiled-printf
- › No recompiling! No redeploying!



DYNAMIC-PRINTF

- Toggle Breakpoint Shift+Ctrl+B
- Add Breakpoint... Ctrl+Double Click
- Add Dynamic Printf...**
- Disable Breakpoint Shift+Double Click
- Breakpoint Properties... Ctrl+Double Click
- Breakpoint Types
- Build Selected File(s)
- Clean Selected File(s)
- Go to Annotation Ctrl+1
- Add Bookmark...
- Add Task...
- ✓ Show Quick Diff Shift+Ctrl+Q
- Show Annotation
- ✓ Show Line Numbers
- Folding
- Preferences...

```

230 testMethod(),
231 fArray[0] = 8.0;
232 fArray[29] = 12.0;
233
234

```

› Handled as CDT breakpoints

```

r_size (8)) =
() + samename

```

Properties for C/C++ Line Dynamic Printf

Common

Class: C/C++ Line Dynamic Printf

File: /home/lmckhou/runtime-TestDSF/DSFTestApp/src/DSFTestApp.cpp

Line number: 226

Enabled

Condition:

Ignore count: 0

printf("Hit line %d of DSFTestApp.cpp\n", 226

Cancel OK



ERICSSON



EXAMINING DEBUGGING DATA



ADVANCED DEBUG HOVER



```
return NULL;
}

// Send some content to port 10010 at variable intervals of 1 to 4 seconds
void *produce(void *ptr) {
    printf("I am the %s thread\n", (char*)ptr);

    struct sockaddr_in addr;
    int sd;

    if ((sd = socket(PF_...
        perror("Socket...
        abort();
    }
    ...
}
```

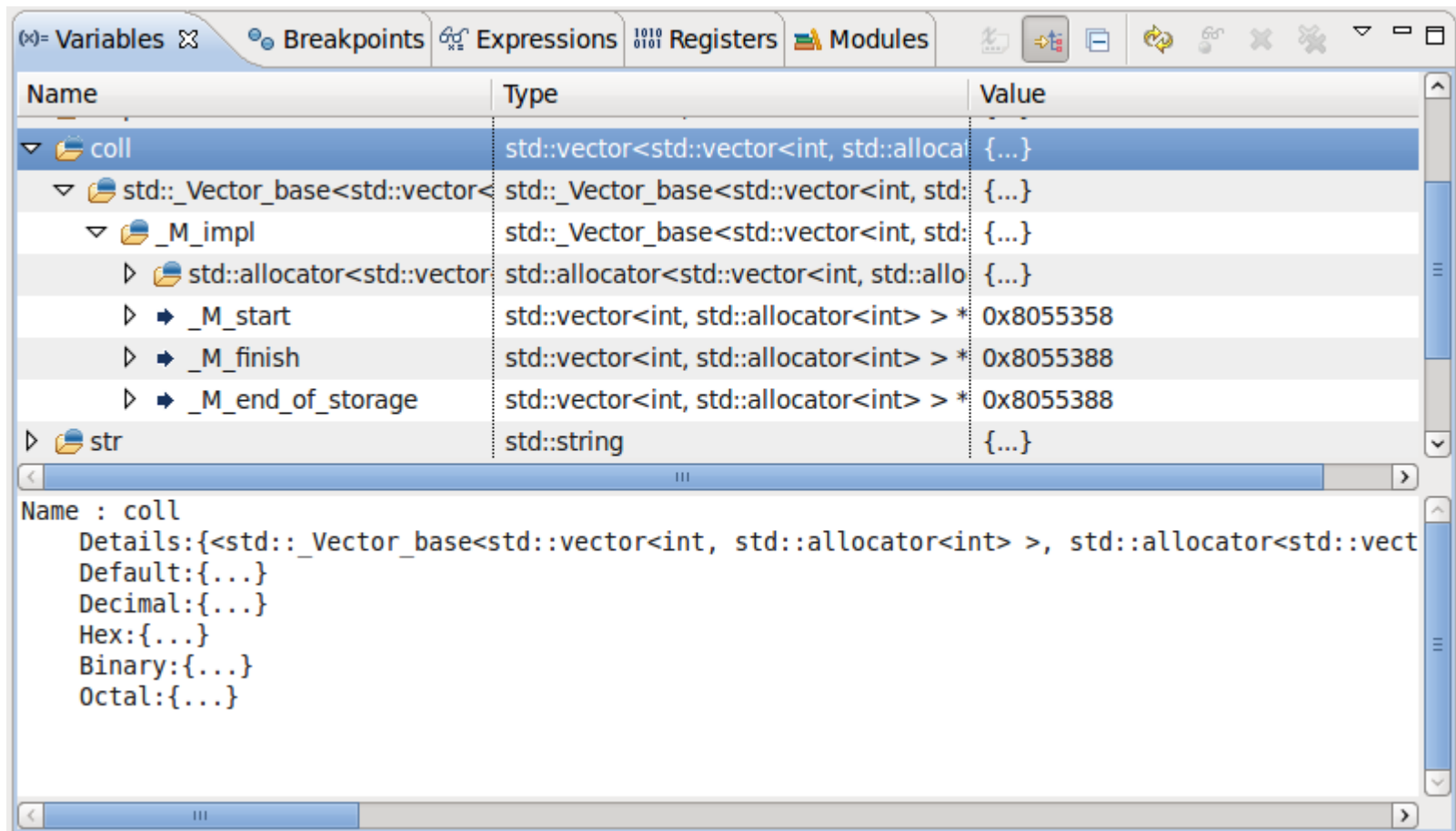
Expression	Type	Value
addr	sockaddr_in	{...}
sin_family	sa_family_t	2
sin_port	in_port_t	6695

Name : addr
Details:{sin_family = 2, sin_port = 6695, sin_addr = {s_addr = 16777343}, sin_zero = {0}}

- > In-hover expression view
- > Detail pane
- > User can modify data directly

PRETTY-PRINTING

- › STL classes inspect poorly e.g., Vector, List, Map



The screenshot shows a debugger's Variables window with the following structure:

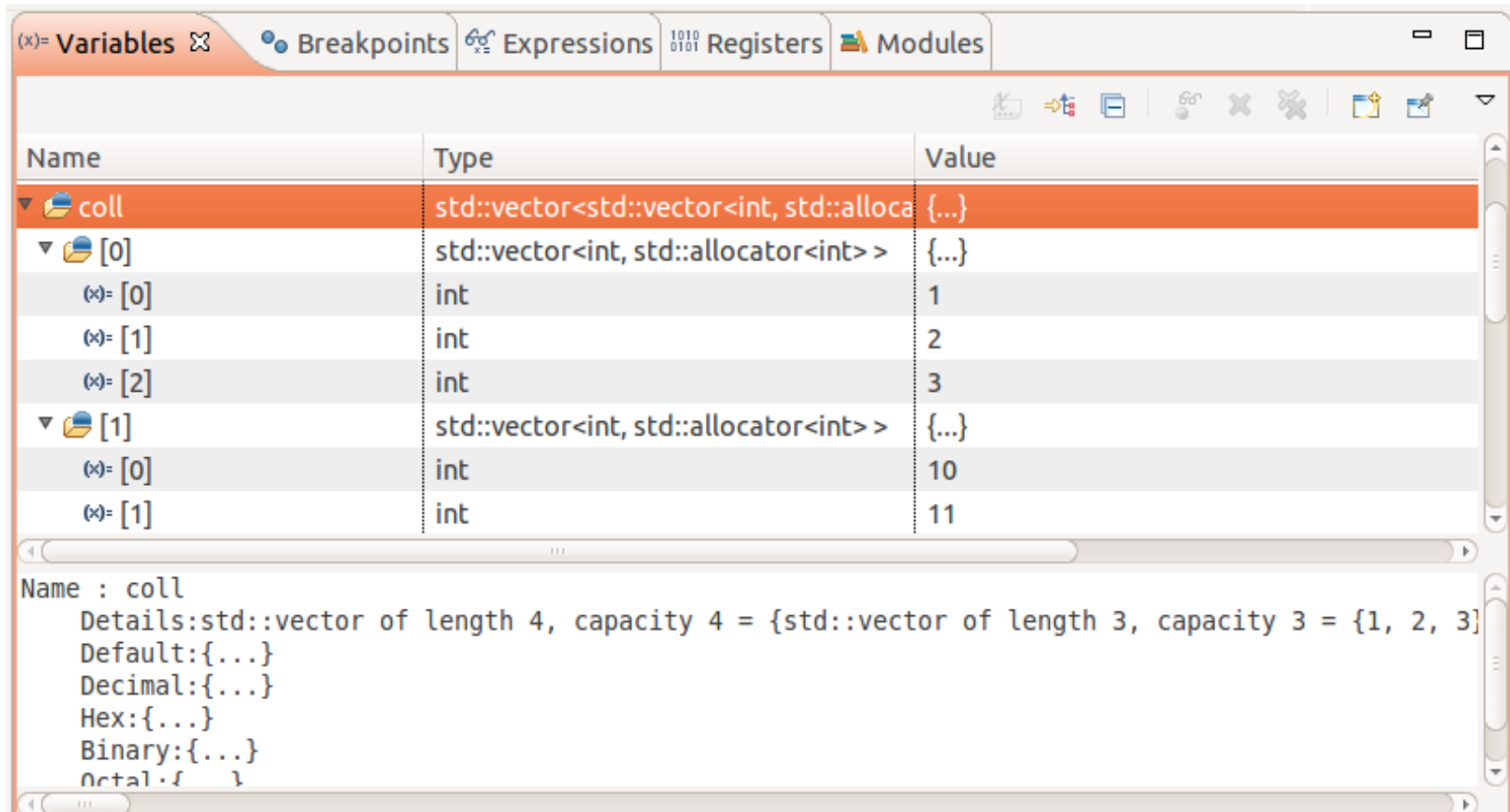
Name	Type	Value
coll	std::vector<std::vector<int, std::allocator<int> >, std::allocator<std::vector<int, std::allocator<int> > > >	{...}
std::_Vector_base<std::vector<int, std::allocator<int> >, std::allocator<std::vector<int, std::allocator<int> > > >	std::_Vector_base<std::vector<int, std::allocator<int> >, std::allocator<std::vector<int, std::allocator<int> > > >	{...}
_M_impl	std::_Vector_base<std::vector<int, std::allocator<int> >, std::allocator<std::vector<int, std::allocator<int> > > >	{...}
std::allocator<std::vector<int, std::allocator<int> > >	std::allocator<std::vector<int, std::allocator<int> > >	{...}
_M_start	std::vector<int, std::allocator<int> > *	0x8055358
_M_finish	std::vector<int, std::allocator<int> > *	0x8055388
_M_end_of_storage	std::vector<int, std::allocator<int> > *	0x8055388
str	std::string	{...}

Below the table, the debugger shows the details for the selected variable 'coll':

```
Name : coll
Details: {<std::_Vector_base<std::vector<int, std::allocator<int> >, std::allocator<std::vector<int, std::allocator<int> > > >, std::allocator<std::vector<int, std::allocator<int> > > >, std::allocator<std::vector<int, std::allocator<int> > > > >
Default: {...}
Decimal: {...}
Hex: {...}
Binary: {...}
Octal: {...}
```


PRETTY-PRINTING

- › Pretty-printers provided with STL library
- › Values of elements can even be modified by user!



The screenshot shows a debugger's 'Variables' window. The top bar includes tabs for 'Variables', 'Breakpoints', 'Expressions', 'Registers', and 'Modules'. The main area is a table with columns 'Name', 'Type', and 'Value'. The variable 'coll' is expanded to show a vector of two vectors. The first inner vector contains elements 1, 2, and 3. The second inner vector contains elements 10 and 11. Below the table, a detailed view for 'coll' shows its type as 'std::vector of length 4, capacity 4 = {std::vector of length 3, capacity 3 = {1, 2, 3}}'.

Name	Type	Value
coll	std::vector<std::vector<int, std::allocator<int>>>	{...}
[0]	std::vector<int, std::allocator<int>>	{...}
(x)= [0]	int	1
(x)= [1]	int	2
(x)= [2]	int	3
[1]	std::vector<int, std::allocator<int>>	{...}
(x)= [0]	int	10
(x)= [1]	int	11

Name : coll
Details:std::vector of length 4, capacity 4 = {std::vector of length 3, capacity 3 = {1, 2, 3}}
Default:{...}
Decimal:{...}
Hex:{...}
Binary:{...}
Octal:{...}



ERICSSON

PER-ELEMENT FORMAT



- › Ability to set format per element
- › Variables, Expressions, Registers views

The screenshot shows the Visual Studio Expressions window with the following table:

Expression	Type	Value	Address
▼ *	Group-pattern	54 unique matches	
▶ a	int [2]	0x7fffffffdbf0	0x7fffffffdbf0
(x)= aba	int	100100 (Binary)	0x7fffffffcd4
(x)= argc	int	1	
▶ argv	char **	0x7fffffffddf8	
▶ array_large	int [111]	0x7fffffffdd6e0	
▶ array_small	int [4]	0x7fffffffdbd0	
▶ b	int [5]	0x7fffffffdba0	
▶ bbb	int *[5]	0x7fffffffdae0	
(x)= c	char	0 '\0'	
(x)= de	int	0 (Binary)	

A context menu is open over the 'aba' row, showing the 'Number Format' menu with 'Binary' selected and 'Octal' highlighted by the mouse. The 'Details' pane for 'aba' is visible at the bottom left:

```
Name : aba
Details:36
Default:36
Decimal:36
Hex:0x24
Binary:100100
Octal:044
```



ERICSSON

MODIFYING DATA



Memory Browser Console Memory

Monitors

0xb6516360

&delay

0xb6516360 : 0xB6516360 <Hex>

Address	0 - 3	4 - 7	8 - B	C - F
B6516360	01000000	05000000	00000042	0200271A
B6516370	7F000000	00000000	00000000	002E0B6B
B6516380	00000000	00000000	886451B6	6E69FBB7
B6516390	46900408	02000000	02000000	02000000
B65163A0	446451B6	00000000	00000000	00000000
B65163B0	00000000	00000000	706B51B6	00000000
B65163C0	00000000	00000000	00000000	00000000
B65163D0	00000000	00000000	00000000	00000000
B65163E0	00000000	00000000	00000000	00000000

› Modifying data during execution:

- Memory view
- Variables view
- Registers view
- Expressions view
- Hover

Variables

Name	Type	Value
ptr	void *	0x8049046
msg	char	0 '\000'
i	int	12345
addr	sockaddr_in	{...}
sd	int	5
delay	int	1



RETURN VALUE DISPLAY



> Return value shown after step-return

The screenshot shows the Visual Studio debugger interface. The top toolbar has a red circle around the 'Step Return' button. The 'Variables' window shows a table with the following data:

Name	Type	Value	Location
computeValue() returned	int	84	
i	int	28	0x7fffffff53c

Below the table, the details for 'computeValue() returned' are shown: Details:84, Default:84, Decimal:84, Hex:0x54, Binary:1010100, Octal:0124.

The source code window shows the following code:

```
1 int computeValue(int i) {  
2     return i*3;  
3 }  
4  
5 int foo(int i) {  
6     if (computeValue(i) > 0)  
7         return 1;  
8     else  
9         return 2;  
10 }  
11  
12 int main() {  
13     foo(28);  
14     return 0;  
15 }  
16
```

A red arrow points from the 'Step Return' button to line 2 of the code. A red box highlights the 'computeValue() returned' row in the Variables window. A red text box says: "When at line 2, pressing step-return will trigger showing the return value".



ERICSSON

RETURN VALUE ON STEP-OVER



- › Currently return value shown only after step-return

When at line 2, pressing step-return will trigger showing the return value

Name	Type	Value	Location
computeValue() returned	int	84	
i	int	28	0x7fffffff55c

```
1 int computeValue(int i) {  
2     return i*3;  
3 }  
4  
5 int foo(int i) {  
6     if (computeValue(i) > 0)  
7         return 1;  
8     else  
9         return 2;  
10 }
```

- › Plans to show return value after a step-over
- › Could be multiple values for a line such as:
 - add (multiply(6,2), divide(9, 3));



ERICSSON

ENHANCED-EXPRESSIONS



- > Shell-like pattern-matching for variables and registers

The screenshot shows the 'Expressions' window in a debugger. The window title is 'Expressions' and it contains a table with three columns: 'Expression', 'Type', and 'Value'. The table is organized into several groups, each starting with a folder icon and a pattern-matching expression. The first group is 'i ; =\$eax ; =array[1,6]' with 4 unique matches. The second group is '\$ebp* ; =um?' with 5 unique matches. The third group is '=array[305-308]' with 4 unique matches. The fourth group is '*' with 54 unique matches. The table is scrollable and has a search bar at the top.

Expression	Type	Value
⌵ i ; =\$eax ; =array[1,6]	Group-pattern	4 unique matches
(x) i	int	9
(x) \$eax	int32_t	-1
(x) array[1]	int	1
(x) array[6]	int	6
⌵ =\$ebp* ; =um?	Group-pattern	5 unique matches
➔ \$ebp	void *	0xbffefe8
(x) \$ebx	int32_t	-1073747096
(x) um2	wchar_t	129 L'\201'
(x) um3	wchar_t	255 L'ÿ'
(x) um4	wchar_t	256 L'Ã'
⌵ =array[305-308]	Group-pattern	4 unique matches
(x) array[305]	int	305
(x) array[306]	int	306
(x) array[307]	int	307
(x) array[308]	int	308
⌵ *	Group-pattern	54 unique matches
▶ a	int [2]	0xbffef08
(x) aba	int	882090000



ERICSSON

ENHANCED-EXPRESSIONS



- › Support for pattern-matching and expressions groups
- › Provides alphabetical sorting

- Pattern-matched local variables

- =v?r – Show all local vars matching pattern
- =* – Show all local vars alphabetically

- Array ranges

- =myarray[30-40] – Show elements 30 to 40
- =myarray[1-3,20,23-24] – Show elements 1,2,3,20,23,24



- › Support for defining expressions and expressions groups
 - Pattern-matched registers
 - `=$xmm*` – Show all registers starting with xmm
 - `=$*` – Show all registers

 - Semi-colon-separated groups
 - `var1; var2` – Group which children are var1 and var2
 - `var1;=*` – Show all local vars with var1 being shown first



ENHANCED-EXPRESSIONS



> Super-set of Variables and of Registers views

Expression	Type	Value
⌵ i ; =\$eax ; =array[1,6]	Group-pattern	4 unique matches
(x)- i	int	9
(x)- \$eax	int32_t	-1
(x)- array[1]	int	1
(x)- array[6]	int	6
⌵ =\$eb* ; =um?	Group-pattern	5 unique matches
➔ \$ebp	void *	0xbffefe8
(x)- \$ebx	int32_t	-1073747096
(x)- um2	wchar_t	129 L'\201'
(x)- um3	wchar_t	255 L'ÿ'
(x)- um4	wchar_t	256 L'Ã'
⌵ =array[305-308]	Group-pattern	4 unique matches
(x)- array[305]	int	305
(x)- array[306]	int	306
(x)- array[307]	int	307
(x)- array[308]	int	308
⌵ *	Group-pattern	54 unique matches
▶ a	int [2]	0xbffef08
(x)- aba	int	882090000



ERICSSON



CONTROLLING EXECUTION



RUN-TO-LINE



ERICSSON

› Run-to-line

- Ctrl+R – Execute program until selected code line
- Or right-click on selected line in editor for menu option

The screenshot shows a code editor with a context menu open over a selected line of code. The code is as follows:

```
253 int* ptr = &i;
254 food* ffPtr = &ff;
255 int aba;
256 bar s = ff;
257 wchar_t um = 0xe4; // ä
258 wchar_t um2 = 0x81; // Invisible.
259 wchar_t um3 = 0xFF; // ÿ
260 wchar_t um4 = 0x100; // Ā
261
262 returnArray(b);
263
264 returnFood(ff);
265
266 int longarray[1000];
267 int *longarraypointer;
268 longarraypointer=&longarray[0];
269 int shortarray[10];
270 shortarray[0] = 260;
271 int *shortarraypointer;
272 shortarraypointer=&shortarray[0];
273
274 int array_small[4] = {0x41424344, 0x45464748}; // Decimal: "1094861636", "116
```

The context menu is open over line 264, and the 'Run to Line' option is highlighted. The menu items are:

- Step Into Selection (Ctrl+F5)
- Show IASTNode in DOM View
- Run to Line (Ctrl+R)
- Move To Line
- Resume At Line
- Add Watch Expression...
- Profile As
- Debug As
- Run As
- Team
- Compare With
- Replace With
- Preferences...
- Input Methods



STEP-INTO-SELECTION



- › Ability to specify *which* method to step into
 - One step to step into 'subtract' instead of 5

```
1295 int result;  
1296 int a = 4, b = 5;  
1297  
1298 add(4,7);  
1299 multiply(5,4);  
1300  
1301 result = subtract( multiply( add( a, b ), 3 ), 5 );  
1302
```

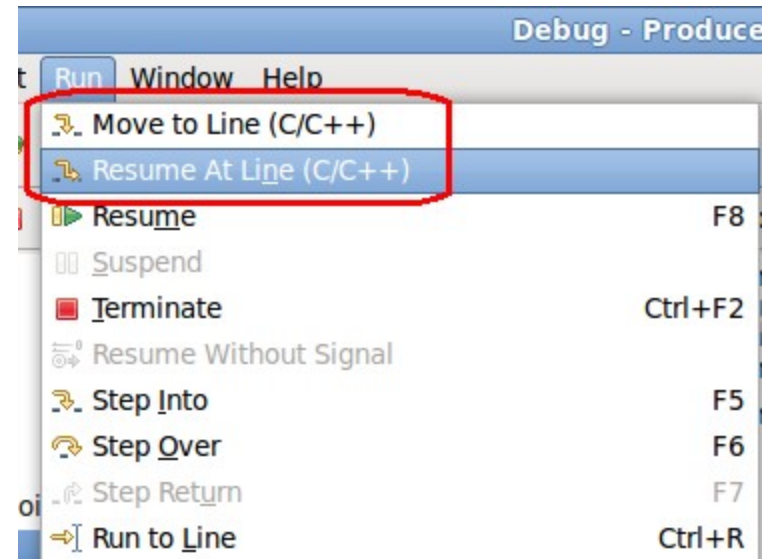


MOVE-TO-LINE & RESUME-AT-LINE



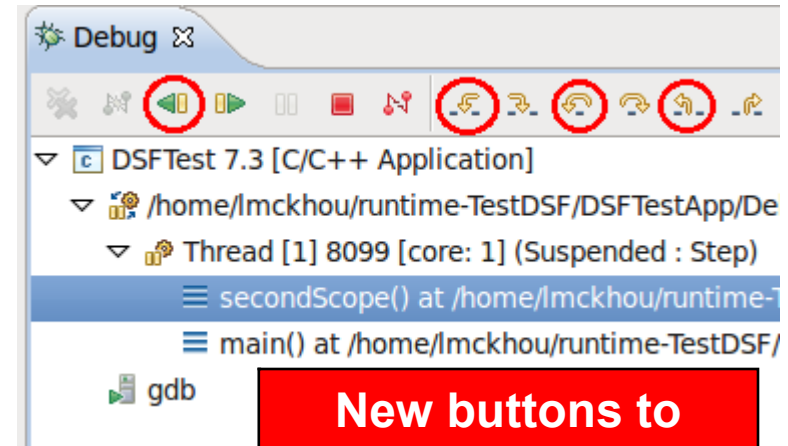
ERICSSON

- › Move-to-line: set execution line to selected one
- › Resume-at-line: move-to-line and automatically resume
- › From *Run* menu or *editor right-click* menu



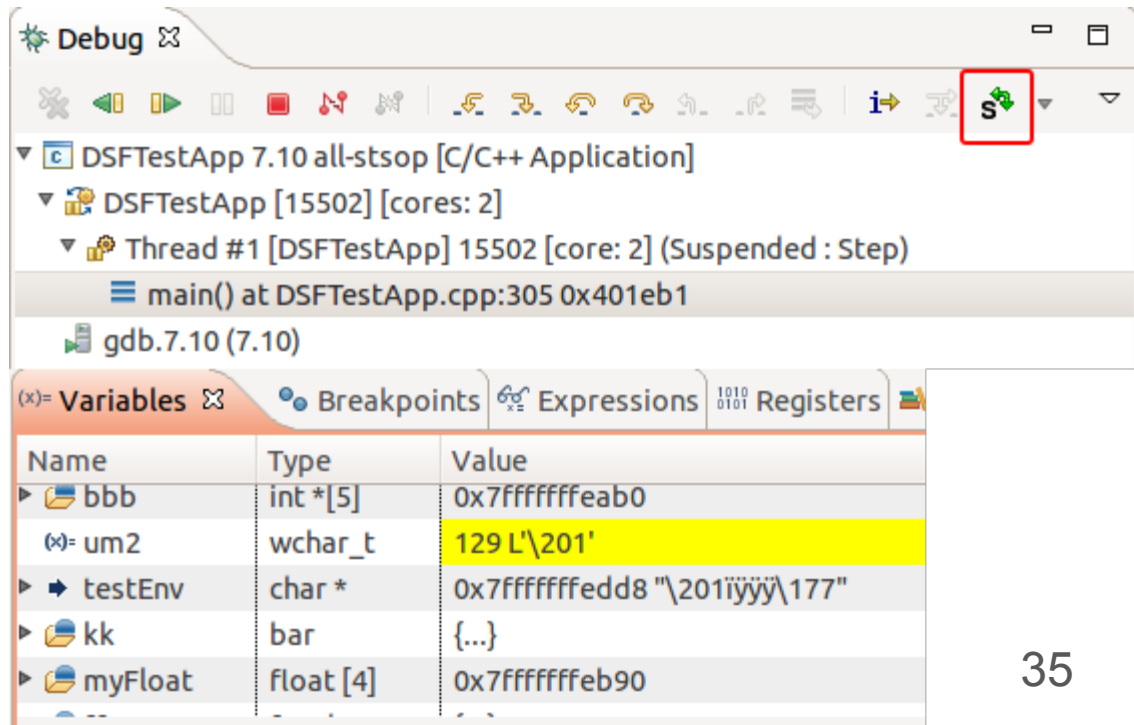
STEP PROGRAM BACKWARDS

- › Recording of program execution
- › Replay in reverse
- › Allows to examine past execution without restarting it
- › Reverse-step, reverse-resume
- › Can use breakpoints set in the 'past'



New buttons to control reverse execution

- › Software recording
 - Code path
 - Variables changes
 - Register changes
 - Memory changes

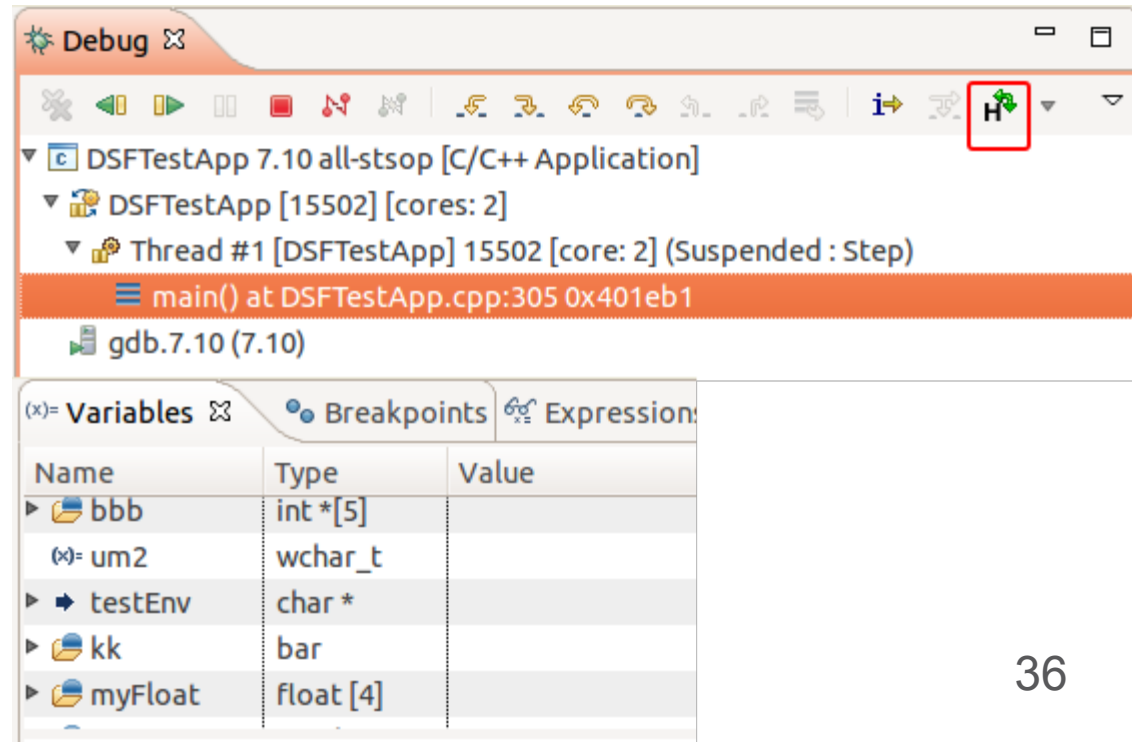


The screenshot shows the Visual Studio Debug window for a C/C++ application. The 'Debug' toolbar at the top right contains a green 'S' icon with a circular arrow, which is highlighted with a red box, representing the software recording feature. Below the toolbar, the application is identified as 'DSFTestApp 7.10 all-stsop [C/C++ Application]'. The current thread is 'Thread #1 [DSFTestApp] 15502 [core: 2] (Suspended : Step)', and the execution is paused at 'main() at DSFTestApp.cpp:305 0x401eb1'. The 'Variables' window is open, displaying the following data:

Name	Type	Value
bbb	int *[5]	0x7fffffffefeb0
um2	wchar_t	129 L'\201'
testEnv	char *	0x7fffffffed8 "\201ïÿÿ\177"
kk	bar	{...}
myFloat	float [4]	0x7fffffffefeb0

› Hardware recording

- Code path only
- Requires Intel(R) processor



The screenshot shows the Visual Studio Debug window for a C/C++ application named DSFTTestApp. The application is running on a multi-core processor (2 cores). The current thread is Thread #1, which is suspended at a step in the main() function at DSFTTestApp.cpp:305. The hardware recording icon (a green 'H' with a refresh symbol) is highlighted with a red box in the top toolbar. Below the Debug window, the Variables window is visible, showing a list of variables and their values.

Name	Type	Value
bbb	int *[5]	
um2	wchar_t	
testEnv	char *	
kk	bar	
myFloat	float [4]	



ERICSSON



MULTI-THREAD AND BEYOND

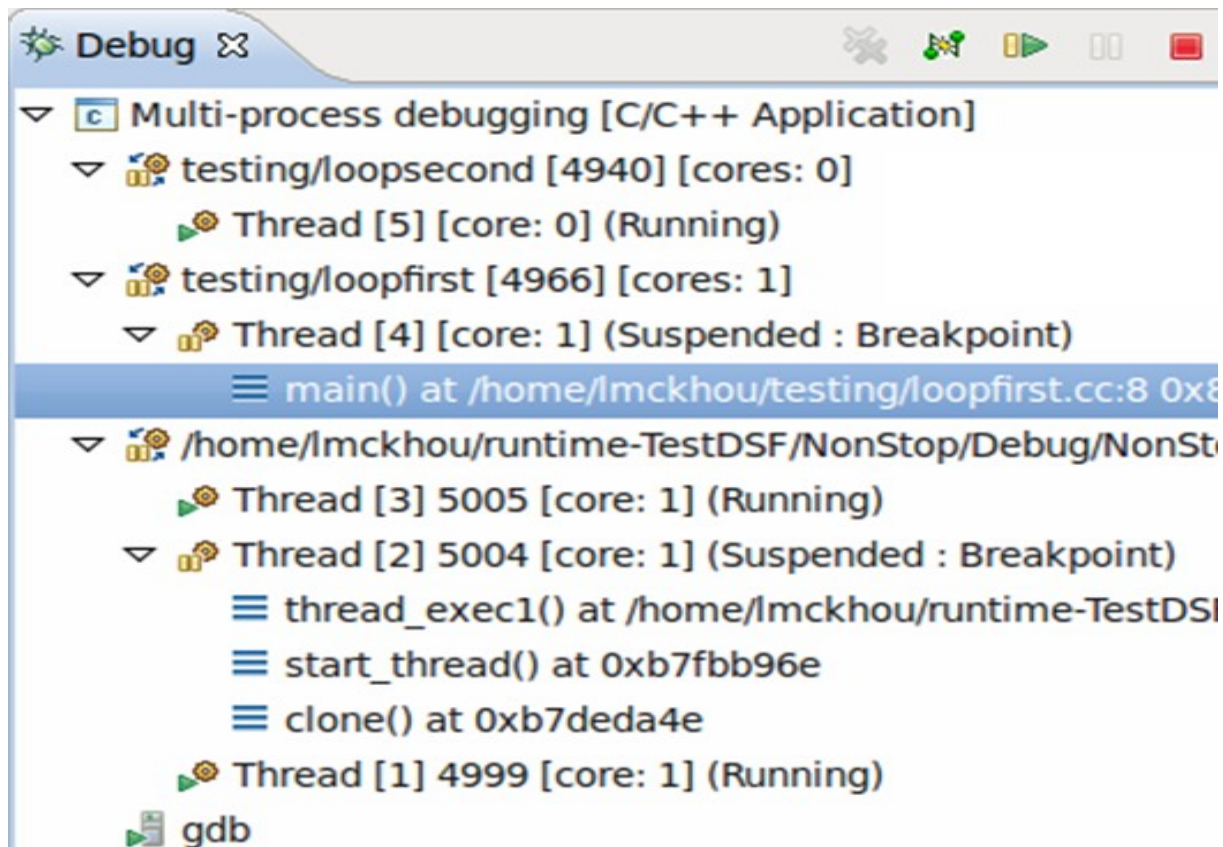


ERICSSON



NON-STOP DEBUGGING

- › Program continues execution while suspending some threads
- › Reduced intrusiveness





ERICSSON

MULTI-PROCESS DEBUGGING



One gdb
controlling many
processes

Debug

- ProducerConsumer [C/C++ Application]
 - /home/lmckhou/runtime-TestDSF/Consumer/Debug/Consumer [12825] [cores: 0,1]
 - Thread [6] 12830 [core: 1] (Suspended : Breakpoint)
 - consume() at /home/lmckhou/runtime-TestDSF/Consumer/src/Consumer.cpp:127 0x8048cce
 - start_thread() at 0xb7fb896e
 - clone() at 0xb7deaa4e
 - Thread [5] 12829 [core: 0] (Running)
 - Thread [4] 12828 [core: 1] (Running)
 - Thread [3] 12827 [core: 1] (Running)
 - Thread [2] 12825 [core: 1] (Running)
 - /home/lmckhou/runtime-TestDSF/Producer/Debug/Producer [12805] [cores: 0,1]
 - Thread [10] 12835 [core: 0] (Suspended : Breakpoint)
 - produce() at /home/lmckhou/runtime-TestDSF/Producer/src/Producer.cpp:127 0x8048cc9
 - start_thread() at 0xb7fb896e
 - clone() at 0xb7deaa4e
 - Thread [9] 12834 [core: 1] (Running)
 - Thread [8] 12833 [core: 1] (Running)
 - Thread [7] 12832 [core: 0] (Running)
 - Thread [1] 12805 [core: 1] (Running)
 - gdb

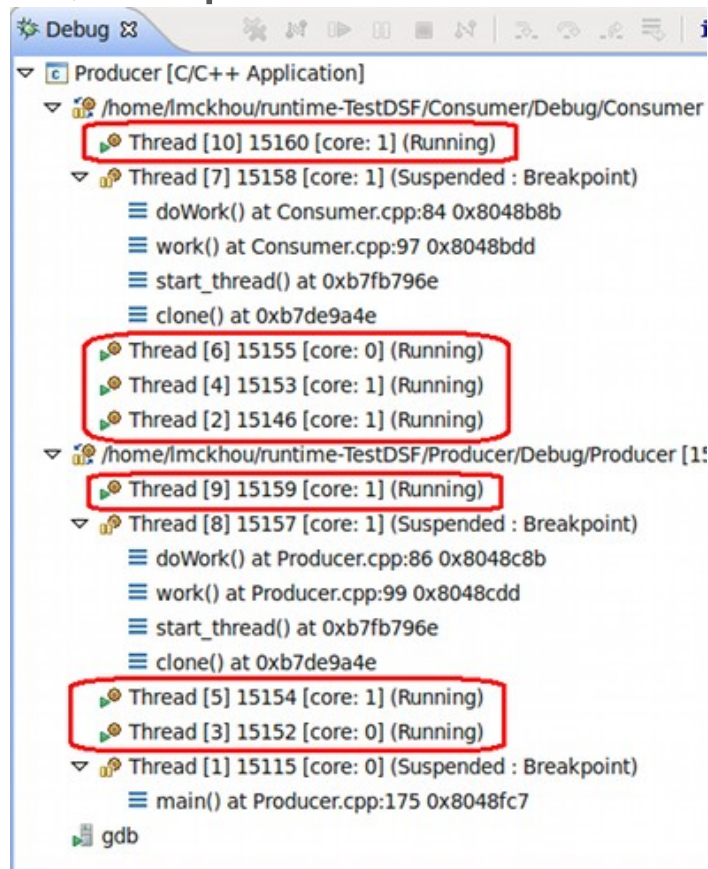
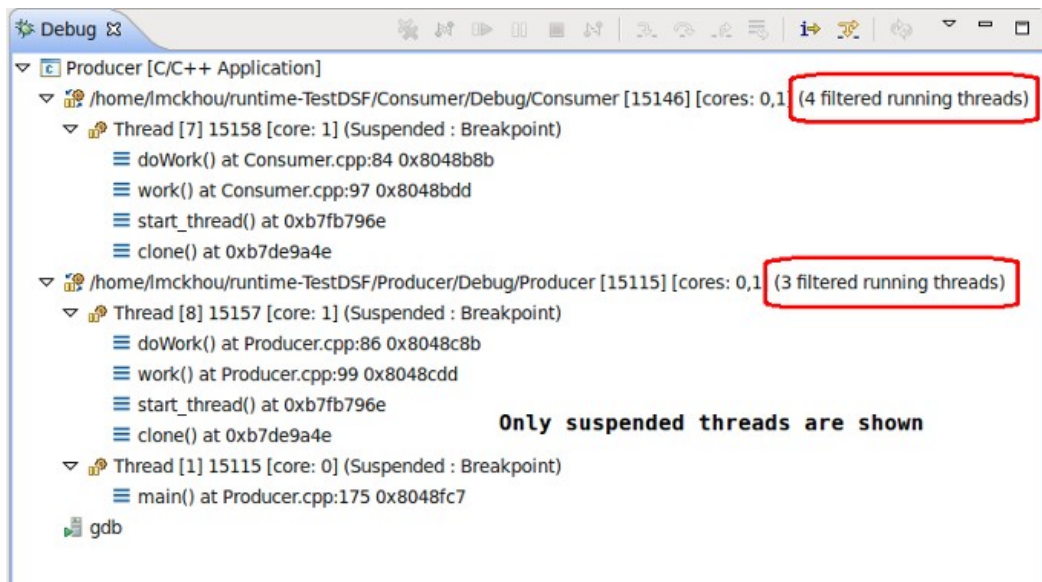


FOCUS ON SUSPENDED THREADS



ERICSSON

- › Crowded display when program has many threads
 - What is really of interest?
 - Threads actively being debugged, i.e., suspended
 - Enable from preferences





ERICSSON



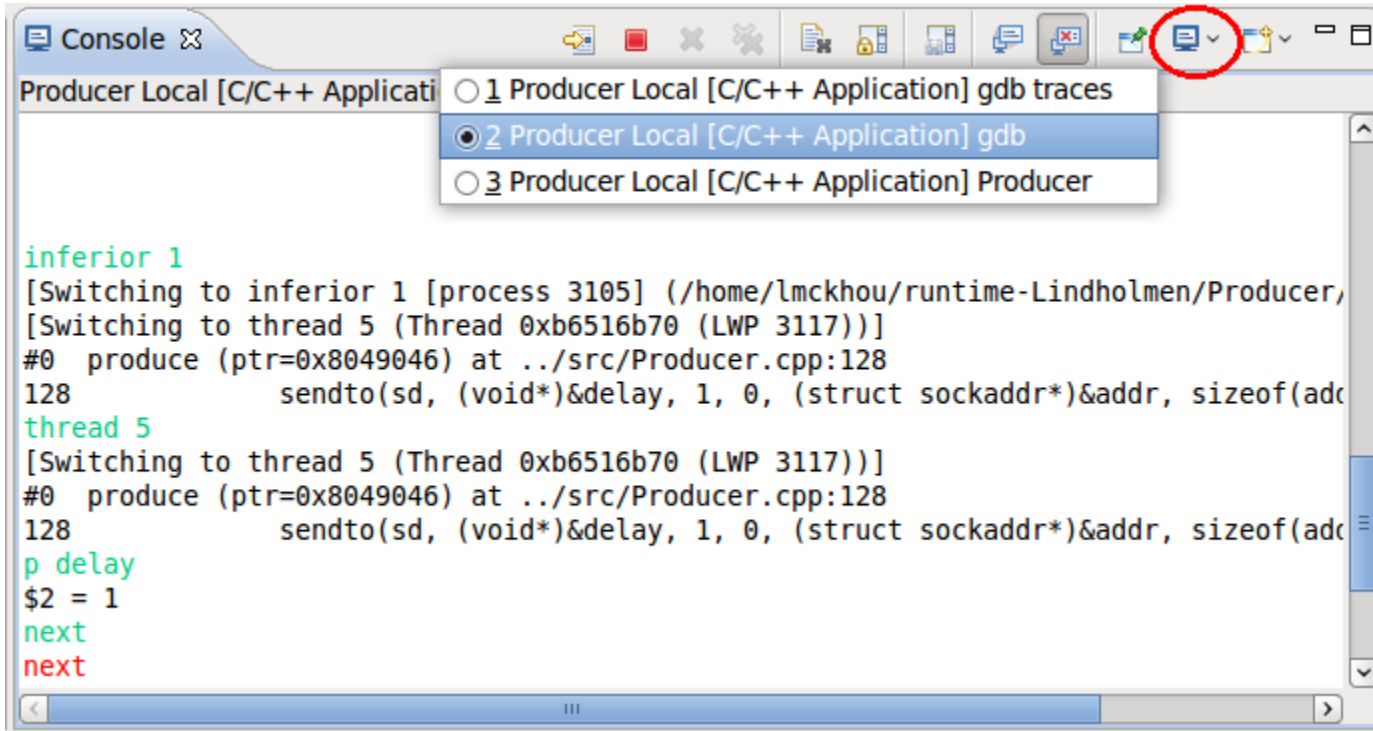
MORE ADVANCED TOPICS



GDB AND GDB CONSOLE



- › GDB is the brains behind CDT Debug
- › Can use gdb command-line from eclipse
- › Currently very basic.

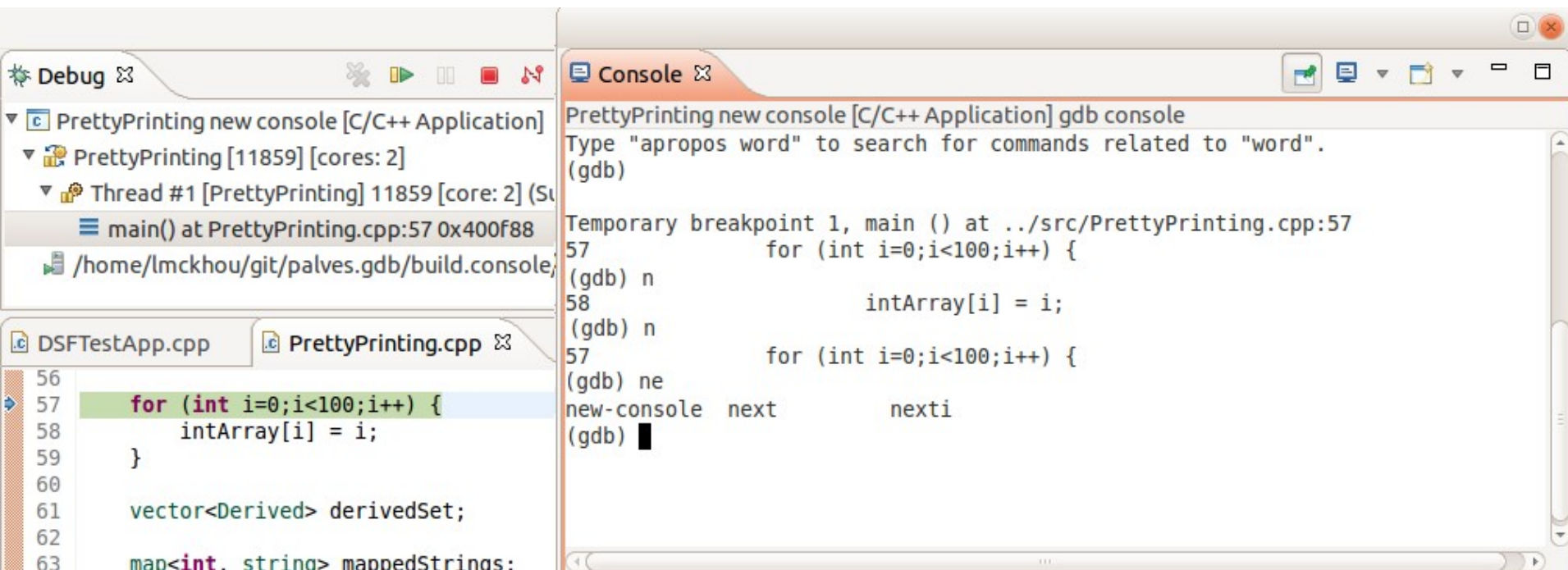




FULL GDB CONSOLE



- › Targeting CDT 9.1 and GDB 7.12 (by September 2016)





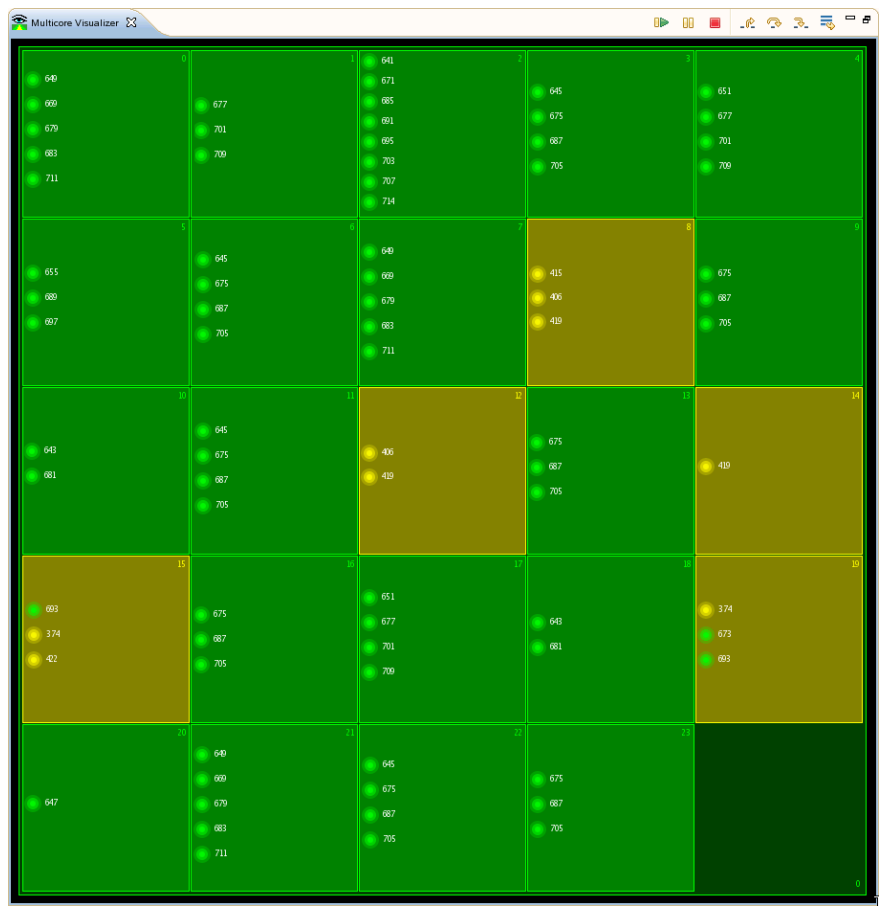
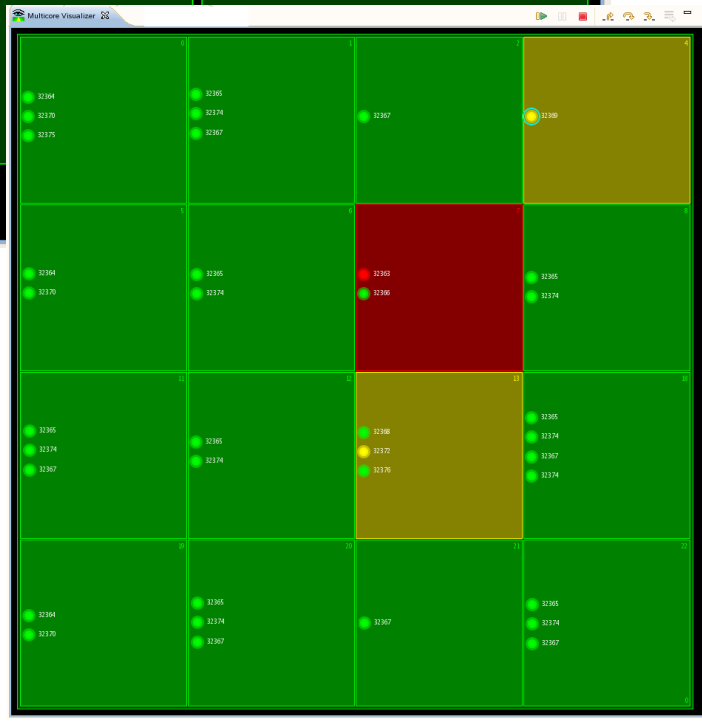
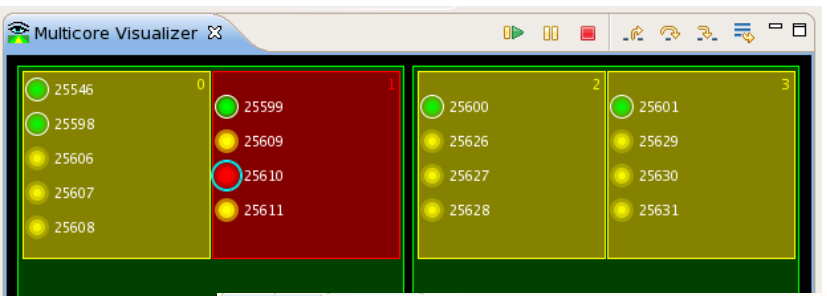
DISASSEMBLY VIEW



- › Shows disassembly of code (optionally with source)
- › Supports breakpoints like in editor (and dynamic-printf!)
- › Step/resume/suspend from Disassembly view

```
Disassembly 
Enter location here       
08048e71: call 0x80487b0 <pthread_create@plt>
171      printf("Main thread waiting for all other tl
08048e76: movl $0x8049064, (%esp)
08048e7d: call 0x80487d0 <puts@plt>
173      pthread_join(hbSendingThread, NULL);
▶ 08048e82: mov 0x2c(%esp), %eax
08048e86: movl $0x0, 0x4(%esp)
08048e8e: mov %eax, (%esp)
08048e91: call 0x80486e0 <pthread_join@plt>
174      pthread_join(hbReceivingThread, NULL);
08048e96: mov 0x28(%esp), %eax
08048e9a: movl $0x0, 0x4(%esp)
08048ea2: mov %eax, (%esp)
08048ea5: call 0x80486e0 <pthread_join@plt>
175      pthread_join(workerThread, NULL);
08048eaa: mov 0x24(%esp), %eax
```


MULTICORE VISUALIZER





ERICSSON



FUTURE PLANS



› Contribution to Linux Kernel ongoing

Applies to every process

Auto attach when hit

Un-started or short lived process



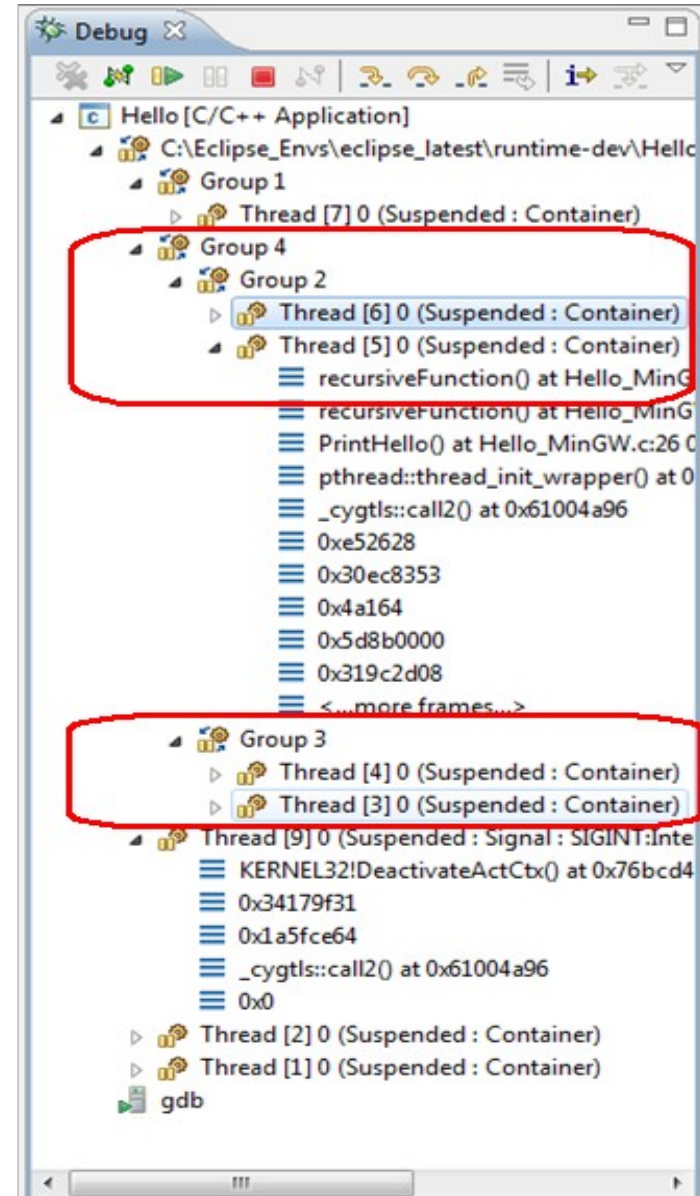


ERICSSON

ITSETS



- IT Sets to control groups of elements
- About multicore
 - Step group of threads or processes
 - Set breakpoint on a subset of threads
 - Resume execution on a core or set of cores





ERICSSON

MORE IDEAS



- Improved handling of breakpoints
 - Showing each installed location per breakpoint
 - ...
- Improved Memory view
 - Showing registers and variables
 - ...
- Evolving Visualizer
 - Better support when dealing with hundreds of cores
 - ...



ERICSSON



CONCLUSION



ERICSSON

CONCLUSION



- › Don't accept *printf-debugging*. This is 2016!
- › Debugger will save you time
- › Debugging does not have to be difficult
- › Help your team improve
 - Lead by example
 - Share knowledge, success stories

Evaluate the Sessions

Sign in and vote at **eclipsecon.org**



- 1 0 + 1



ERICSSON

SOME REFERENCES



- › CDT Project, <http://www.eclipse.org/cdt>
- › CDT FAQ, <http://wiki.eclipse.org/CDT/User/FAQ>
- › CDT Debug workgroup
<http://wiki.eclipse.org/CDT/MultiCoreDebugWorkingGroup>
- › CDT Wiki, <http://wiki.eclipse.org/CDT>

FINAL Q&A

