Debugging with MicroEJ

Overview of MicroEJ Debugging Tools

MICROEJ®

DISCLAIMER

OVERVIEW

- Goal:
  - Provide an overview of the debug tools provided to developers to debug an application
  - Illustrate the use of the debug tools

- Debug tools categories:
  - Runtime & Post-Mortem Debugging Tools
  - Memory Inspection Tools (debug corruption, leaks)
  - Static Analysis Tools
  - GUI Application Debugging Tools (bottlenecks identification, rendering issues)

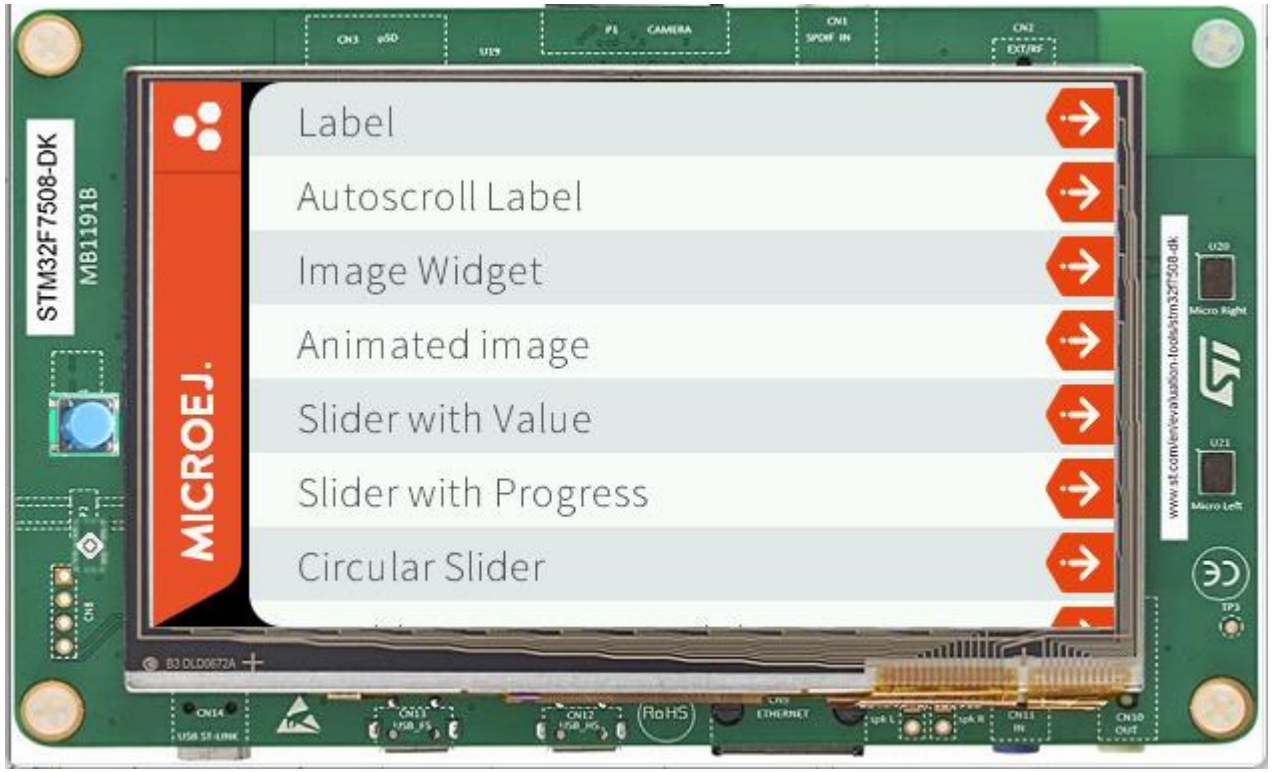
DEBUGGING TOOLS OVERVIEW

MICROEJ

TOOLS	RUNTIME & POST-MORTEM	MEMORY INSPECTION	STATIC ANALYSIS TOOLS	GUI DEBUGGING TOOLS
Core Engine VM Dump	X			
Debug on Device	X			
Simulator Debugger	X			
Port Qualification Tool (PQT)	X			
SystemView	X	X		X
Logging & Message Libraries	X			
Code Coverage	X			
Heap Dumper / Analyzer		X		
Heap Usage Monitoring		X		
Core Engine MEMORY integrity check		X		
SonarQube / Klocwork (Java/C)			X	
Null Analysis			X	
Flush Visualizer				X
MWT & Widget Debug Utilities				X

ENVIRONMENT

- MICROEJ SDK 5.8.1
- Applications:
  - Demo Widget
  - MWT samples
- VEE Port: STM32F7508-DK
  - CPU: STM32F750N8H6
    - 216 MHz Arm Cortex-M7 core
  - ROM:
    - Internal flash size: 64 KB
    - External flash size: 128 Mb
  - RAM:
    - Internal RAM size: 340 KB
    - External RAM size: 64 Mb
  - Screen: 480x276 16BPP, capacitive touch



ENVIRONMENT SETUP

To reproduce the use cases presented in those slides:

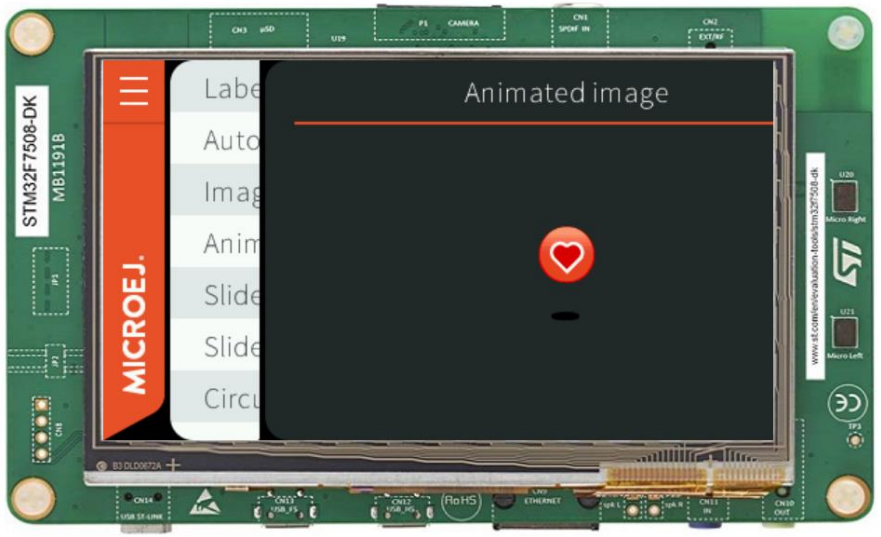
1. Use a VEE Port with GUI capability (MicroUI 3.1 or later)
2. Clone the following repositories:
   - Example-Java-Widget 7.6.0
   - ExampleJava-MWT 2.5.0
3. Get the Git patch **debug_training_example_java_widget.patch** in the training package
4. Apply the Git patch to the Example-Java-Widget repository
5. In MICROEJ SDK, import the following projects:
   - com.microej.demo.widget
   - slide-container

Runtime & Post-Mortem Debugging Tools

RUNTIME & POST-MORTEM DEBUGGING TOOLS

- Tools:
  - Core Engine VM Dump
  - Debug on Device
  - Debug on Simulator
  - Port Qualification Tools (qualify a VEE Port)
  - Event Tracing & Logging*
  - Code Coverage*

- Example:
  - Debug a deadlock in an application in the Simulator and on Device



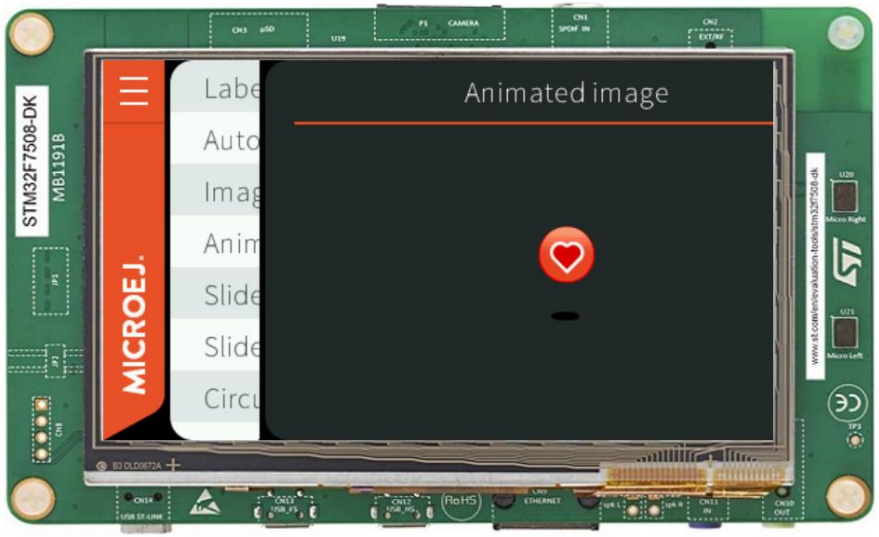
GUI freeze when entering a page

* Tool not introduced in this presentation, visit docs.microej.com for more information.

DEBUG A DEADLOCK IN AN APPLICATION

REPRODUCE THE ISSUE

- Run **com.microej.demo.widget** in Simulation or on the Device
- Enter the **Animated Image** page
- The GUI should freeze during the screen transition:



CORE ENGINE VM DUMP (1/3)

- Core Engine VM Dump is a diagnose tool to investigate unexpected behavior occurring on the target.

- When?
  - Call the LLMJVM_dump() method in the Core Engine task at runtime to diagnose unexpected behavior (ex: UI freeze).
  - Call the LLMJVM_dump() as a last resort in a fault handler to get a snapshot of the Core Engine, to check if the issue comes from a LLAPI or the underlying C code.

- What?
  - Prints the state of the MicroEJ Core Engine to the standard output stream.
  - For each Java thread, the Java stack trace, the name, the state and the priority are printed.

- Requirements:
  - A way to read stdout (usually UART).

CORE ENGINE VM DUMP (2/3)

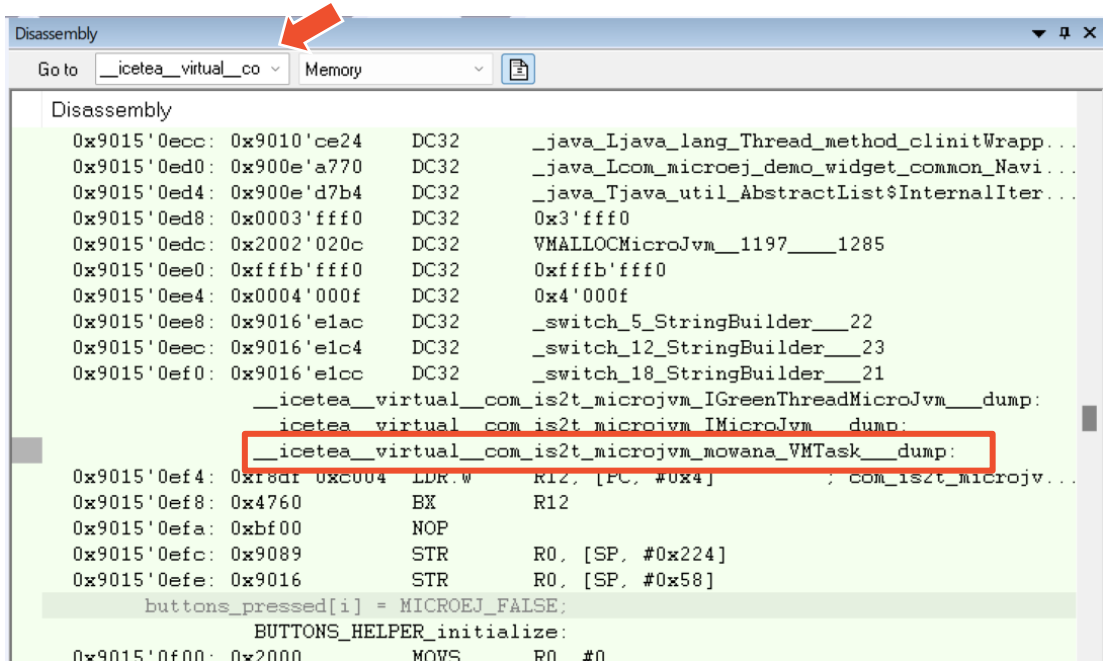
HOW-TO?

- LLMJVM Dump triggered from User button press (blue button):

buttons_manager.c

```
28 static void BUTTONS_MANAGER_event(Button_TypeDef Button)
29 {
30     uint8_t button_pressed;
31
32     if (HAL_GPIO_ReadPin(BUTTON_PORT[Button], BUTTON_PIN[Button]) == GPIO_PIN_SET)
33     {
34         // GPIO == 1
35
36         if (BUTTON_REVERSE[Button] == MICROEJ_TRUE)
37         {
38             // GPIO == 1 means "released"
39             button_pressed = MICROEJ_FALSE;
40         }
41         else
42         {
43             // GPIO == 1 means "pressed"
44             button_pressed = MICROEJ_TRUE;
45             LLMJVM_dump();
46         }
47     }
```

- Trigger the LLMJVM Dump from the debugger (IAR):



__icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump

CORE ENGINE VM DUMP (3/3)

EXAMPLE OF DUMP

- Dead lock is identified in the stack trace, lock between threads “Thread1” and “UI Pump”
- The UI Tread (UI Pump) is locked → GUI Freeze
- Use the Stack Trace Reader to decode the stack trace

```
================================== VM Dump ==================================
Java threads count: 3
Peak java threads count: 3
Total created java threads: 4
Last executed native function: 0x9014DDFB
Last executed external hook function: 0x00000000
State: idle, not notified
-----------------------------------------------------------------------------
Java Thread[1794]
name="Thread1" prio=5 state=MONITOR_QUEUED max_java_stack=492 current_java_stack=183
Locked on: java/lang/Object@0xC0081C4C (owned by thread[1281])

java/lang/Thread@0xC0082150:
   at com/microej/demo/widget/animatedimage/widget/AnimatedImage$1.run(AnimatedImage.java:190)
      Object References:
         - com/microej/demo/widget/animatedimage/widget/AnimatedImage$1@0xC00821B0
         - java/lang/Object@0xC0081C48
         - java/lang/Object@0xC0081C4C

-----------------------------------------------------------------------------
Java Thread[1281]
name="UIPump" prio=5 state=MONITOR_QUEUED max_java_stack=1296 current_java_stack=850
Locked on: java/lang/Object@0xC0081C48 (owned by thread[1794])

java/lang/Thread@0xC008047C:
   at com/microej/demo/widget/animatedimage/widget/AnimatedImage.renderContent(AnimatedImage.java:233)
      Object References:
         - com/microej/demo/widget/animatedimage/widget/AnimatedImage@0xC0081C2C
         - ej/microui/display/GraphicsContext@0xC008042C
         - java/lang/Object@0xC0081C4C
         - java/lang/Object@0xC0081C48
```

USING THE DEBUGGER (1/4)

DEBUG ON SIMULATOR

- Use of JDWP (Java Debug Wire Protocol) to use Eclipse debugger
- Use mocks to simulate and debug corner cases of the target

- Debugger features:
  - Breakpoints
  - Step-by-step execution
  - Variables and fields value monitoring
  - Thread execution stacks list

DEBUG ON DEVICE

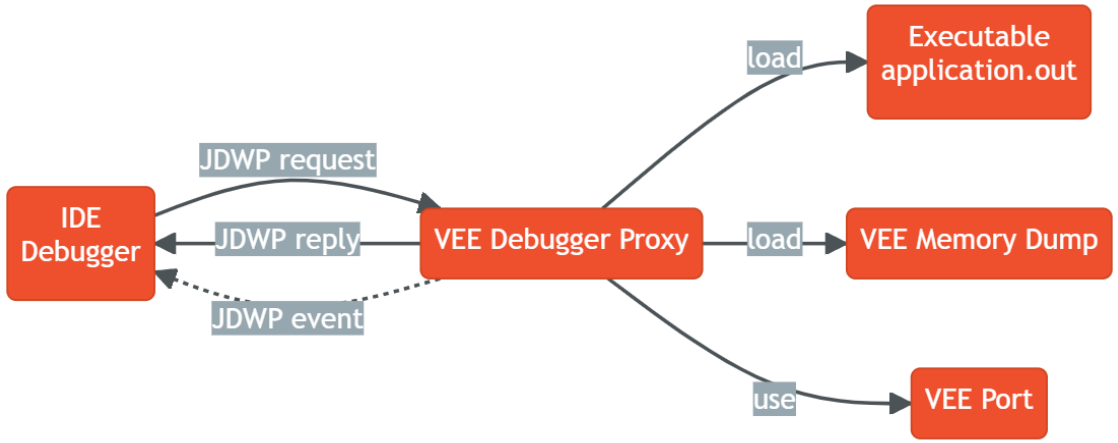
- Use of JDWP (Java Debug Wire Protocol) to use Eclipse debugger
- Need to setup the a VEE Debugger Proxy
- Postmortem debug from a snapshot of the memory
- Debugger features:
  - Breakpoints
  - Step-by-step execution (planned)
  - Variables and fields value monitoring
  - Thread execution stacks list

Note: import the Foundation Library Sources to the debugger to get the exact source code which is executed.

USING THE DEBUGGER (2/4)

DEBUG ON DEVICE

- VEE Debugger Proxy principle:



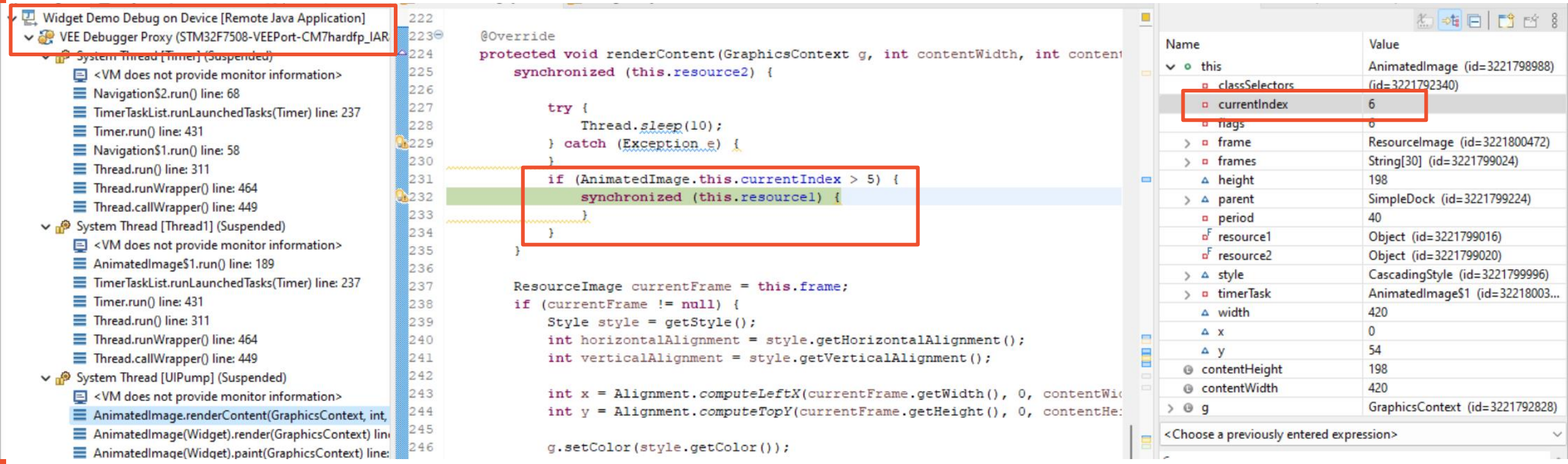
- Available since Architecture 8.1

- No VEE Port update required
- Steps:
  1. Generate a VEE memory dump script for the target / toolchain
  2. Run the application Executable on target
  3. Dump the memory of the running Executable using the C Debugger using the VEE memory dump script
  4. Run the VEE Debugger Proxy in a Command Prompt
  5. In MICROEJ SDK, run a Remote Java Application Debugging session

USING THE DEBUGGER (3/4)

DEBUG ON DEVICE

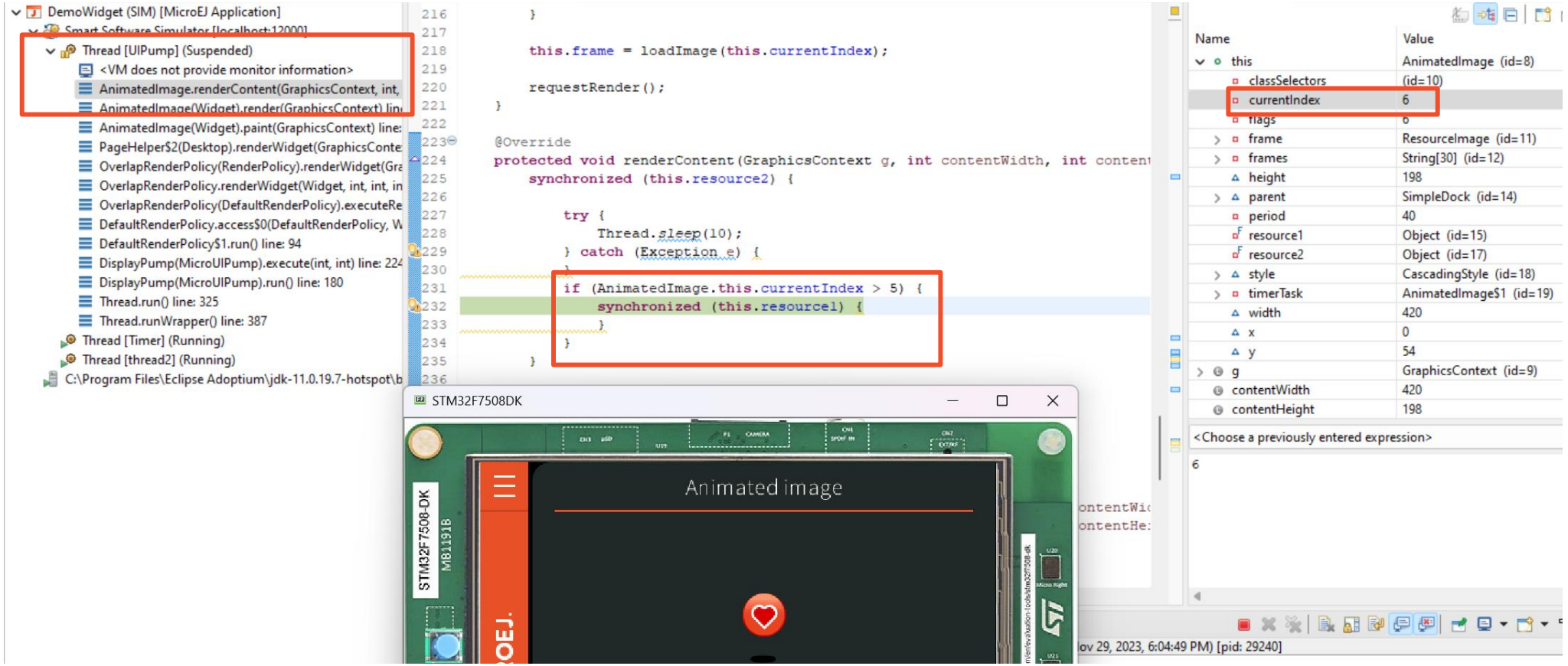
- To debug an application on device, first run the **VEE Debugger Proxy**, and run a **Remote Java Application** launch:



USING THE DEBUGGER (4/4)

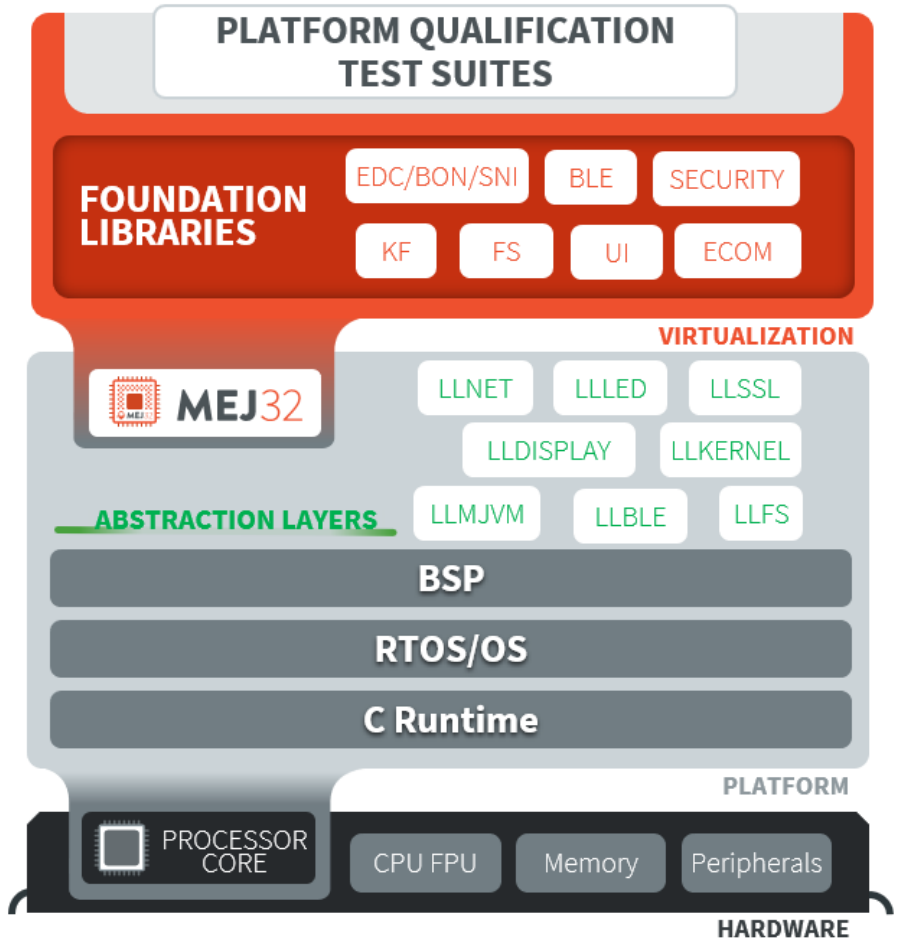
DEBUG ON SIMULATOR

- To debug an application on Simulator, select it in the left panel then right-click and select **Debug As > MicroEJ Application:**



PORT QUALIFICATION TOOL (1/2)

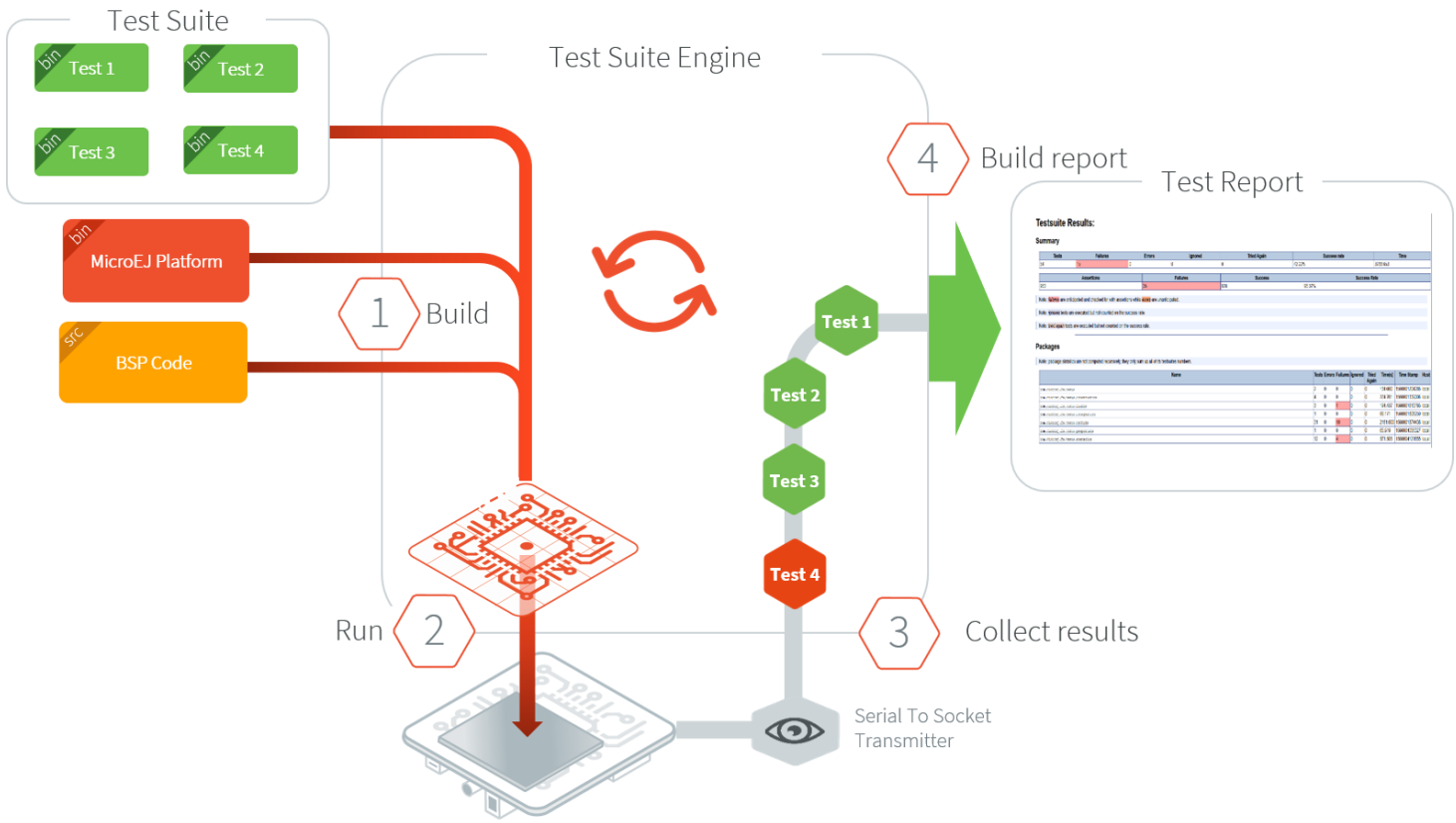
- The Port Qualification Tool (PQT) project provides the tools required to validate each component of a MicroEJ VEE Port.
- After porting or adding a Foundation Library to a MicroEJ VEE Port, it is necessary to validate its integration.
- For each Low Level API, an Abstraction Layer implementation is required. The validation of the Abstraction Layer implementation is performed by running tests at two-levels:
  - In C, by calling Low Level APIs (usually manually).
  - In Java, by calling Foundation Library APIs (usually automatically using Platform Test Suite).
- PQT tests can be extended by the developer to support custom Foundation Libraries.
- Please refer to the Platform Qualification documentation for more information.



PORT QUALIFICATION TOOL (2/2)

- PQT tests are provided with a Test Suite project, to run tests automatically (CI or locally)
  → Agility in the development flow

- A Test Suite contains one or more tests. For each test, the Test Suite Engine will:
  - Build a MicroEJ Firmware for the test.
  - Program and Run the MicroEJ Firmware onto the device.
  - Retrieve the execution traces.
  - Analyze the traces to determine whether the test has PASSED or FAILED.
  - Append the result to the Test Report.
  - Repeat until all tests of the Test Suite have been executed.



VEE Port Test Suite on Device Overview

KEY TAKEWAYS

1. PQT: validate the vertical integration: Foundation Library > Abstraction Layer > C Library > Driver

2. Event Tracing & Logging: instrument the application with debug logs

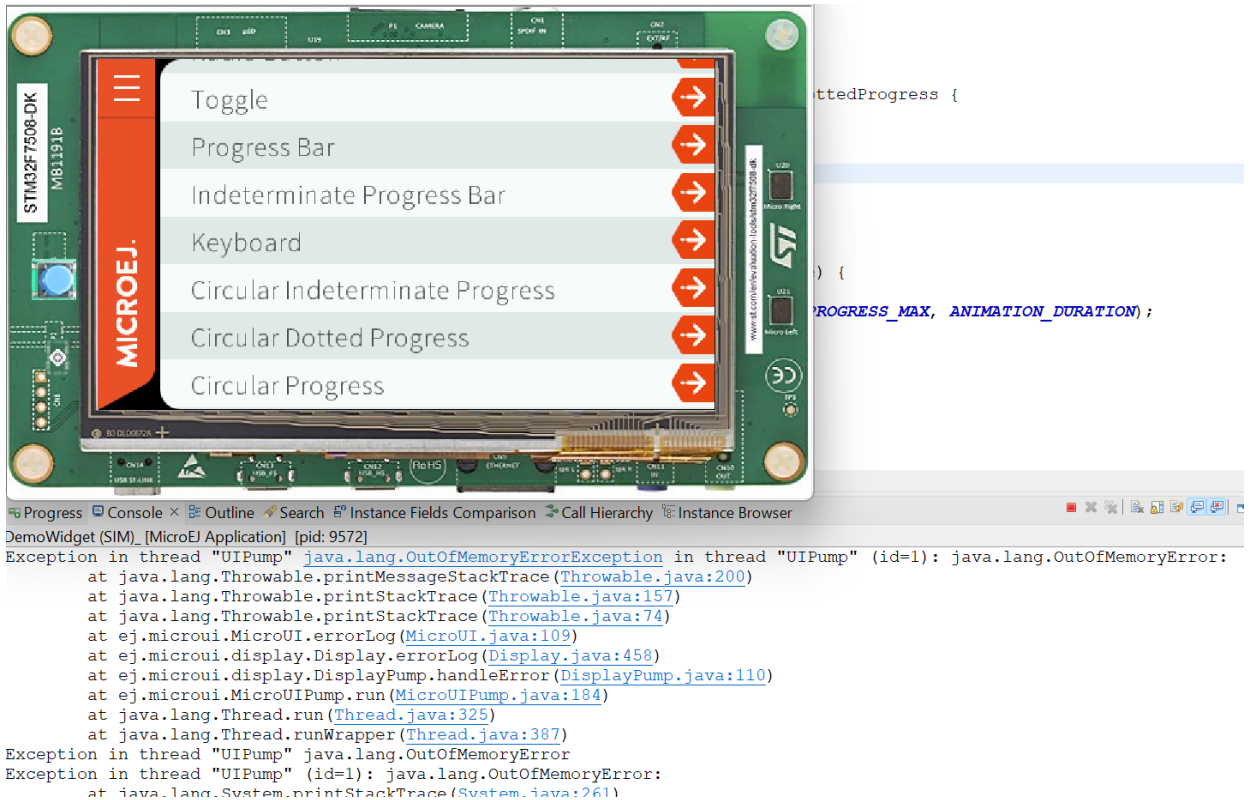
3. Core Engine VM Dump: diagnosis tool to display the state of the MicroEJ Runtime and the MicroEJ threads on target (name, priority, stack trace, etc.)

4. Debugger (on device & simulator): analysis of an applicative issue

Memory Inspection Tools

MEMORY INSPECTION TOOLS

- Tools:
  - Heap Dumper & Heap Analyzer
  - Core Engine Memory integrity check
  - Heap Usage Monitoring Tool*
- Examples:
  - Investigate memory leaks
  - Detect memory corruption of the Core Engine heap



Out Of Memory exception in a GUI application

* Tool not introduced in this presentation, visit docs.microej.com for more information.

HEAP DUMPER & HEAP ANALYZER (1/4)

- Heap Dumper is a tool that takes a snapshot of the heap.
  Generated files (.heap extension) are available in the application output folder.

- Heap Analyzer is a tool that allows to inspect the heap dumps.
  It provides the following features:
  - Memory leaks detection
  - Objects instances browse
  - Heap usage optimization (using immortal or immutable objects)
  - Comparison between Heap Dumps

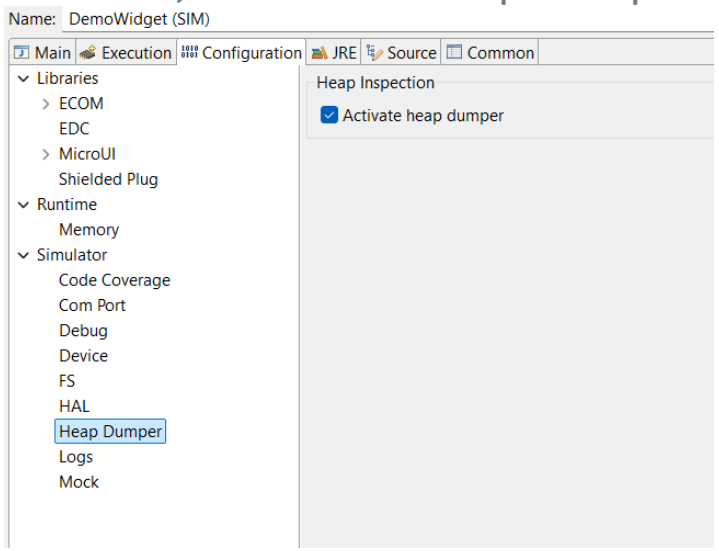
- To generate .heap dump files, **System.gc()** must be called explicitly in the application code.

- .heap dump files can be generated in simulation and also dumped from the device.

HEAP DUMPER & HEAP ANALYZER (2/4)

REPRODUCE THE ISSUE

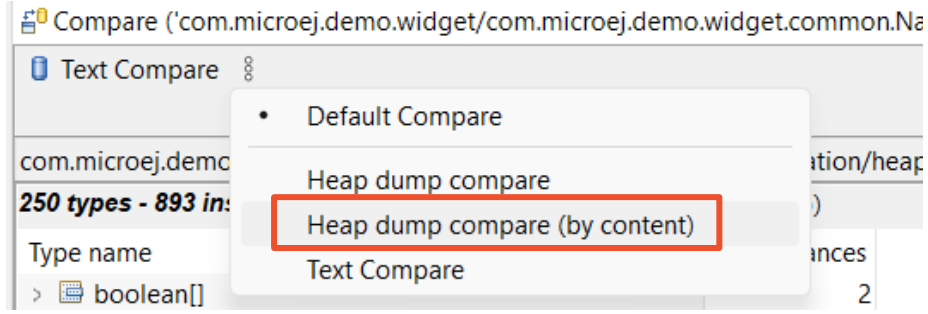
- In the **com.microej.demo.widget** Simulator launcher, enable the Heap Dumper:

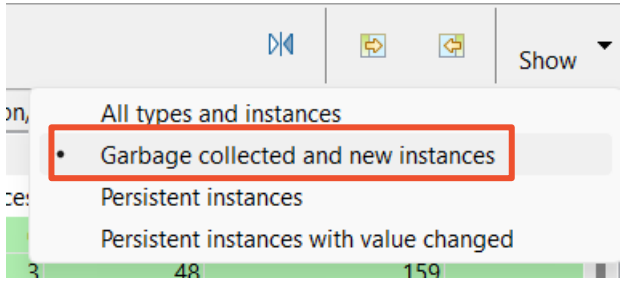


- Run on Simulator. Heap dumps are performed every 4 seconds (see Navigation.java)
- Enter / leave the **Circular Dotted Progress page** ~10 times
- Get the error trace in the console

OPEN HEAP DUMPS

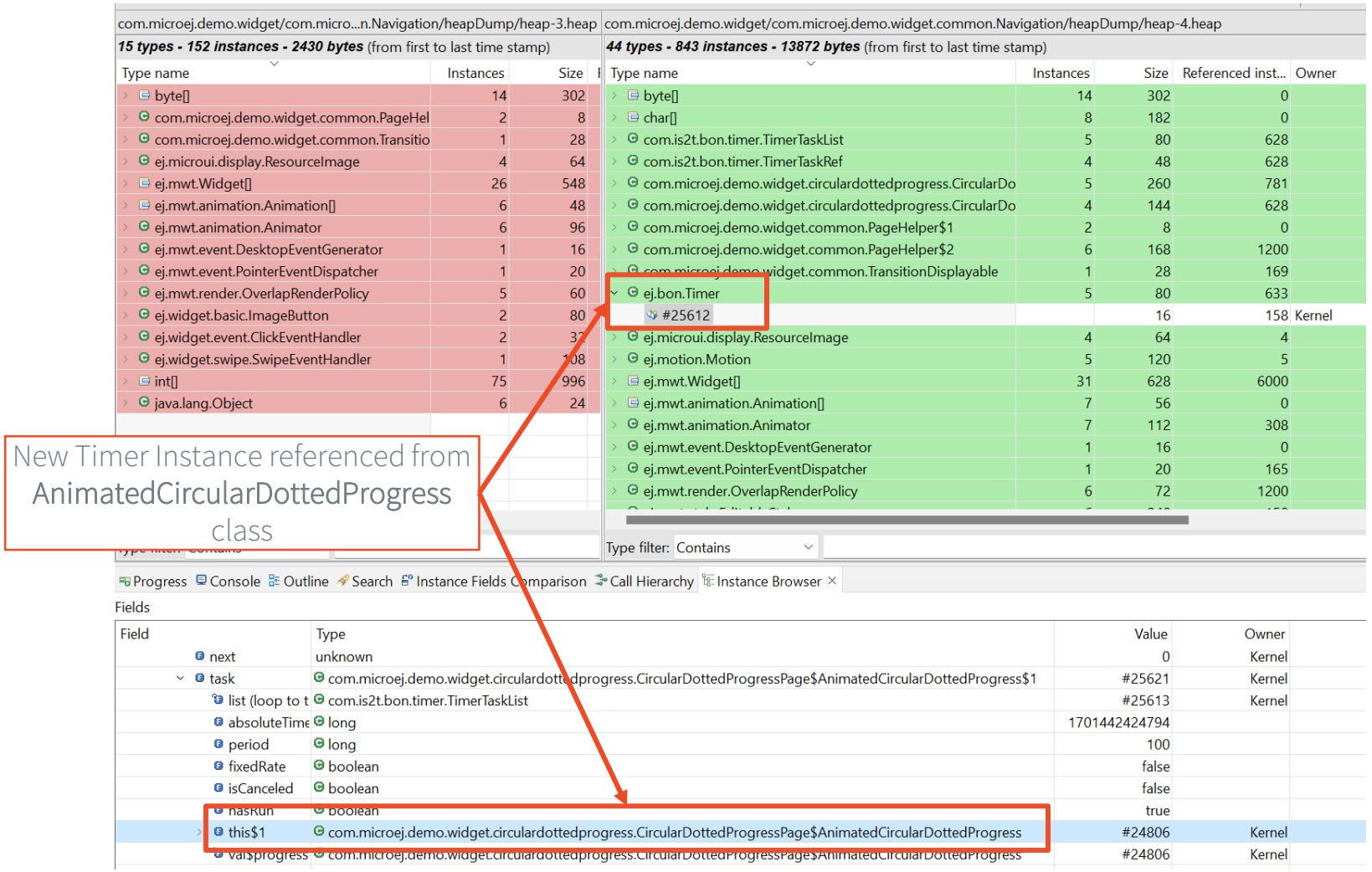
- Heap Dumps are generated in the **com.microej.demo.widget/com.microej.demo.widget.common.Navigation/heapDump** folder.
- Right-Click on 2 consecutive **.heap** files.
- Click on **Compare With → Each Other**.
- The Heap Viewer opens, select the following configuration:





HEAP DUMPER & HEAP ANALYZER (3/4)

- Heap Compare between .**heap-3** and .**heap-4:**



Guidelines:

- Lots of new objects have been created (691 new instances)
- Use the **compare by content** option to discard objects that moved but have the same content
- Look for new objects that can have an impact (Thread, Timer, Page, Widget, StyleSheet)

  → knowledge of the application required, need to understand the objects hierarchy
- Once an object has been picked, look its parent in the **Instance Browser**

HEAP DUMPER & HEAP ANALYZER (4/4)

ROOT CAUSE ANALYSIS

- New Timer instance created each time the **CircularDottedProgressPage** is shown:

*CircularDottedProgressPage.java

```
81      @Override
82      protected void onShown() {
83          System.gc();
84          this.startTime = Util.platformTimeMillis();
85          final AnimatedCircularDottedProgress progress = this;
86          Timer timer = new Timer();
87          TimerTask task = new TimerTask() {
88
89              @Override
90              public void run() {
91                  progress.tick();
92              }
93          };
94          timer.schedule(task, 0, 100);
95      }
96
97      public void tick() {
98          this.angle = (this.angle + 15) % 360;
99          setProgress(this.angle);
100
101         requestRender();
102     }
```

→ Memory leak is due to the useless Timer instances keeping a reference on the widget **AnimatedCircularDottedProgress**
Also, the TimerTask is never canceled

FIX

- Retrieve a global Timer instance (defined at application startup)
- Cancel the TimerTask once the **CircularDottedProgressPage** is hidden

```
@Override
protected void onShown() {
    this.startTime = Util.platformTimeMillis();
    final AnimatedCircularDottedProgress progress = this;
    Timer timer = ServiceFactory.getService(Timer.class, Timer.class);
    this.task = new TimerTask() {

        @Override
        public void run() {
            progress.tick();
        }
    };
    timer.schedule(this.task, 0, 100);
}

@Override
protected void onHidden() {
    if (this.task != null) {
        this.task.cancel();
    }
    this.task = null;
}
```

CORE ENGINE MEMORY INTEGRITY CHECK (1/3)

- The **LLMJVM_checkIntegrity** API checks the internal memory structure integrity of the Core Engine with the LLMJVM_checkIntegrity API to detect memory corruptions in native functions.

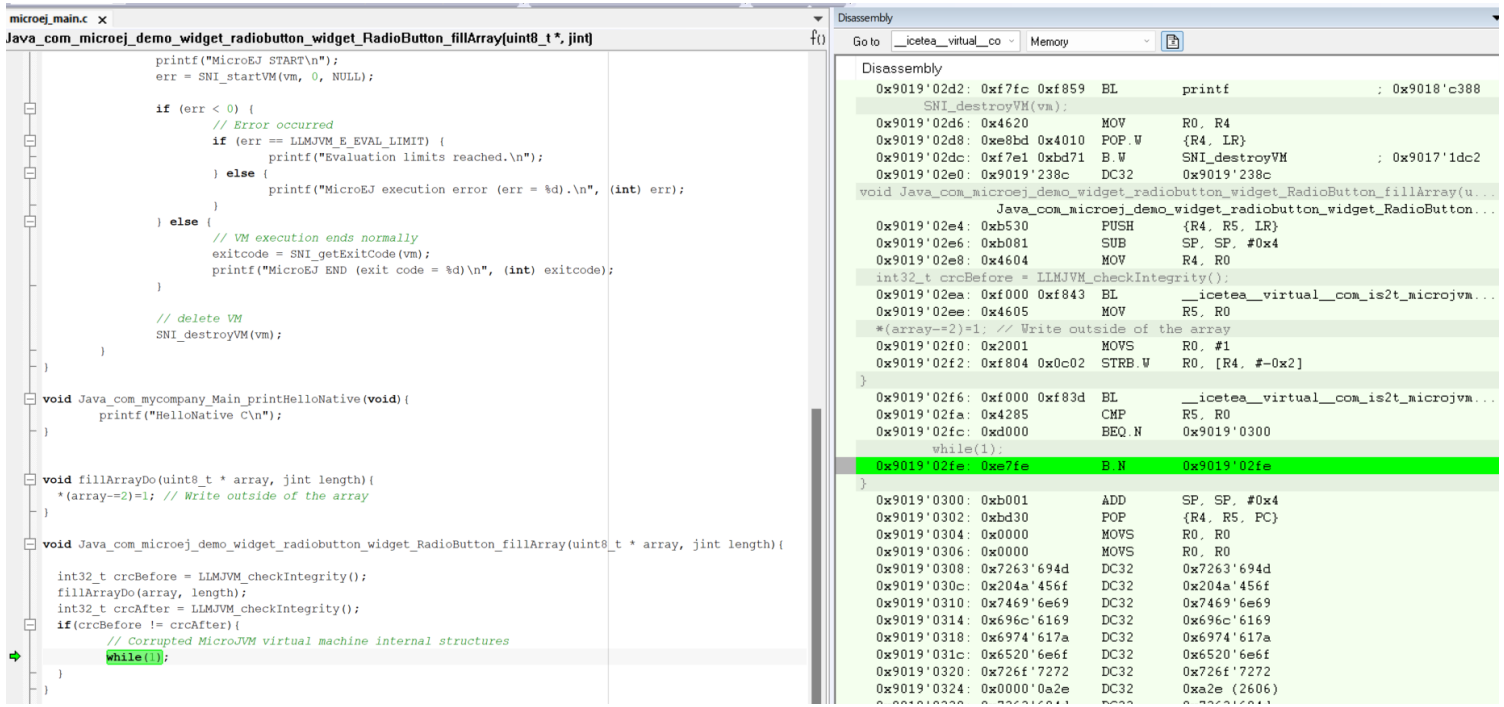
- This feature is for Applications deployed on hardware devices only:
  - If an integrity error is detected, the **LLMJVM_on_CheckIntegrity_error** hook is called and this method returns 0.
  - If no integrity error is detected, a non-zero checksum is returned.

- Note: this function affects performance and should only be used for debug purpose.

CORE ENGINE MEMORY INTEGRITY CHECK (2/3)

REPRODUCE THE ISSUE

- Add C code provided in **fill_array_heap_corruption.c** in the BSP project
- Build and run the **com.microej.demo.widget** on the device
- Enter the **Radio Button page**, click on one of the buttons
- The GUI should freeze, the Heap is corrupted
- In debug mode, the execution is stuck in a while loop:.



CORE ENGINE MEMORY INTEGRITY CHECK (3/3)

ROOT CAUSE ANALYSIS

- The **fillArrayDo** native function writes outside the array memory area:

```
void fillArrayDo(uint8_t * array, jint length){
  *(array--=2)=1; // Write outside of the array
}

void Java_com_microej_demo_widget_radiobutton_widget_RadioButton_fillArray(uint8_t * array, jint length){

  int32_t crcBefore = LLMJVM_checkIntegrity();
  fillArrayDo(array, length);
  int32_t crcAfter = LLMJVM_checkIntegrity();
  if(crcBefore != crcAfter){
        // Corrupted MicroJVM virtual machine internal structures
        while(1);
  }
}
```

FIX

- Fix the implementation of **fillArrayDo**.

KEY TAKEWAYS

- Heap Dumper:
  - Generates heap dumps (.heap file) on System.gc() execution

- Heap Analyzer features:
  - Compare: compares two heap dumps, showing which objects were created, or garbage collected, or have changed values
    → useful for memory leaks detection
  - Heap Viewer: shows which instances are in the heap, when they were created, and attempts to identify problematic areas
    → useful for memory optimization

- Core Engine Memory Integrity Check: detect memory corruptions in native functions.

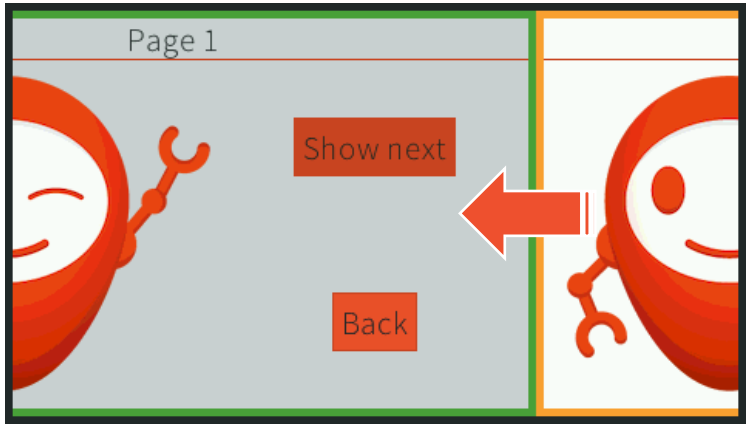
- Heap Usage Monitoring Tool: estimate the heap requirements of an application.

Debugging a GUI Application

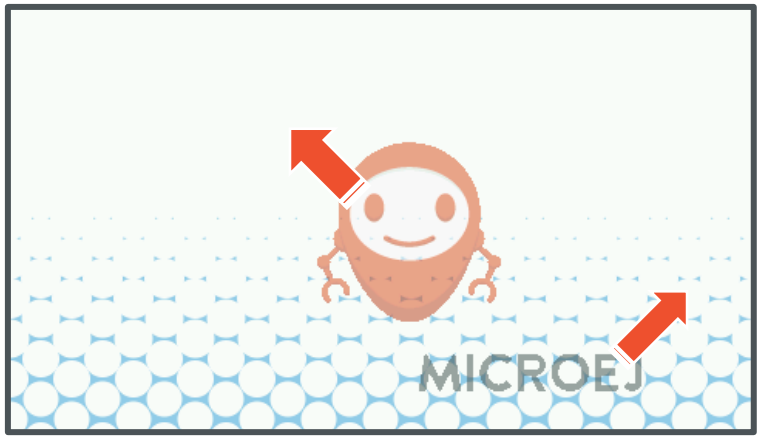
Identifying & Debugging Performance Bottlenecks

IDENTIFYING & DEBUGGING BOTTLENECKS

- Tools:
  - Flush Visualizer
  - SystemView
- Example:
  - Identify performance bottlenecks that prevents smooth animations



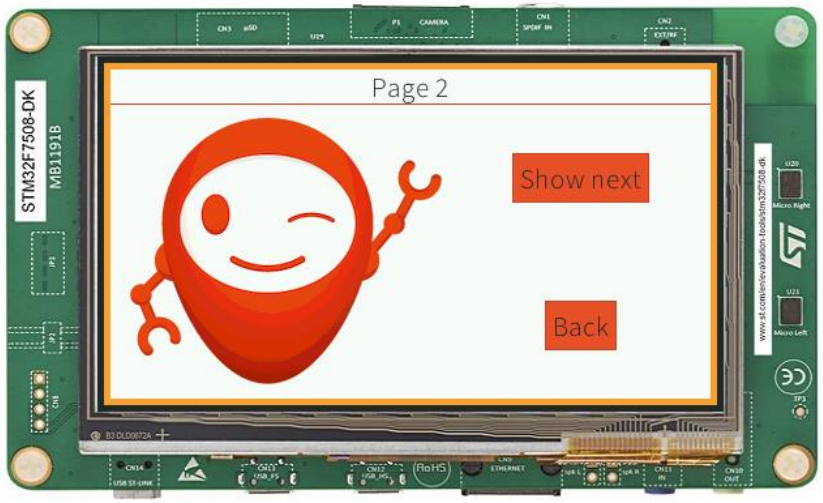
Sliding animation between 2 pages



Animated images

FLUSH VISUALIZER (1/2)

- The Flush Visualizer shows the pixel surface drawn between two MicroUI frame buffer flushes.
- A perfect application has 100% of its display area drawn. **A total area drawn between 100% to 200%** is the norm in practice because widgets often overlap.

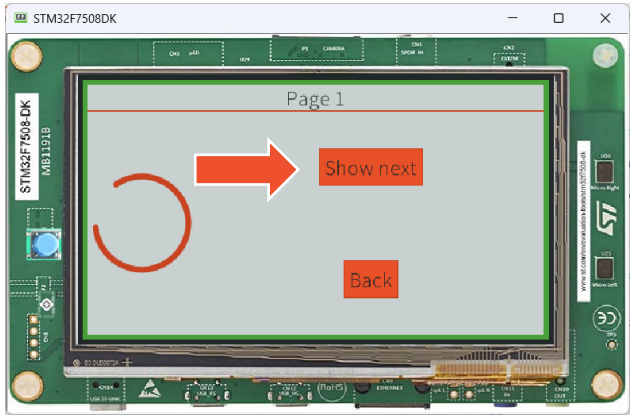



```
fillRectangle x1=0 y1=0 x2=479 y2=271 area=99
fillRectangle x1=5 y1=5 x2=474 y2=266 area=93
fillRectangle x1=5 y1=5 x2=9 y2=266 area=0
fillRectangle x1=5 y1=5 x2=474 y2=9 area=1
fillRectangle x1=470 y1=5 x2=474 y2=266 area=0
fillRectangle x1=5 y1=262 x2=474 y2=266 area=1
fillRectangle x1=10 y1=36 x2=469 y2=36 area=0
fillRectangle x1=318 y1=74 x2=422 y2=111 area=2
fillRectangle x1=318 y1=74 x2=318 y2=111 area=0
fillRectangle x1=318 y1=74 x2=422 y2=74 area=0
fillRectangle x1=422 y1=74 x2=422 y2=111 area=0
fillRectangle x1=318 y1=111 x2=422 y2=111 area=0
fillRectangle x1=343 y1=187 x2=397 y2=224 area=1
fillRectangle x1=343 y1=187 x2=343 y2=224 area=0
fillRectangle x1=343 y1=187 x2=397 y2=187 area=0
fillRectangle x1=397 y1=187 x2=397 y2=224 area=0
fillRectangle x1=343 y1=224 x2=397 y2=224 area=0
drawImage regionX=0 regionY=0 width=260 height=235 x=10 y=37 alpha=255 area=44
fillRectangle x1=0 y1=0 x2=479 y2=271 area=99
fillRectangle x1=5 y1=5 x2=474 y2=266 area=93
fillRectangle x1=5 y1=5 x2=9 y2=266 area=0
fillRectangle x1=5 y1=5 x2=474 y2=9 area=1
fillRectangle x1=470 y1=5 x2=474 y2=266 area=0
fillRectangle x1=5 y1=262 x2=474 y2=266 area=1
fillRectangle x1=10 y1=36 x2=469 y2=36 area=0
fillRectangle x1=318 y1=74 x2=422 y2=111 area=2
fillRectangle x1=318 y1=74 x2=318 y2=111 area=0
fillRectangle x1=318 y1=74 x2=422 y2=74 area=0
fillRectangle x1=422 y1=74 x2=422 y2=111 area=0
fillRectangle x1=318 y1=111 x2=422 y2=111 area=0
fillRectangle x1=343 y1=187 x2=397 y2=224 area=1
fillRectangle x1=343 y1=187 x2=343 y2=224 area=0
fillRectangle x1=343 y1=187 x2=397 y2=187 area=0
fillRectangle x1=397 y1=187 x2=397 y2=224 area=0
fillRectangle x1=343 y1=224 x2=397 y2=224 area=0
drawImage regionX=0 regionY=0 width=260 height=235 x=10 y=37 alpha=255 area=44
32# Area drawn: 493%
```

FLUSH VISUALIZER (2/3)

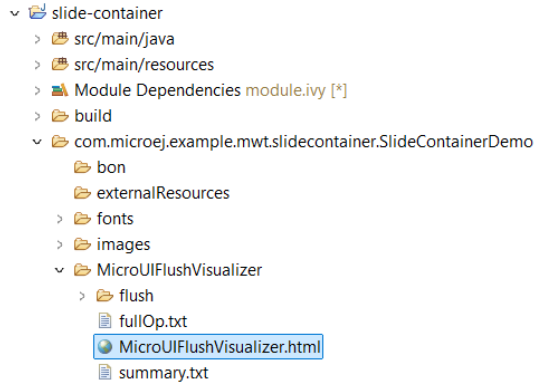
REPRODUCE THE ISSUE

- Enable the Flush Visualizer in the FrontPanel project of the VEE Port
- Run the **slide-container** example of **ExampleJava-MWT** repository on Simulator
- Click on the **Show next** button, the Flush Visualizer displays a message in the Console:




```
slide-container SIM_ [MicroEJ Application] C:\Program Files\Eclipse Adoptium\jdk-11.0.23.9-hotspot\bin\javaw.exe  (Jul 31, 2024, 5:54:06 PM) [pid: 23700]
=============== [ Initialization Stage ] ===============
=============== [ Converting fonts ] ===============
WARNING: option 'com.microej.runtime.kf.waitstop.delay' is not defined (automatically set to '2000')
=============== [ Converting images ] ===============
=============== [ Launching on Simulator ] ===============
Warning: Area drawn 202% during flush 1
Warning: Area drawn 404% during flush 281
```

- Flush Visualizer report is accessible in the project output folder:



FLUSH VISUALIZER (3/3)

ROOT CAUSE ANALYSIS

- Drawings are done twice once the animation is over:

```
public void tick(int value, boolean finished) {
    // Move the 2 pages
    updatePosition(value, leftChild, rightChild);
    if (finished) {
        // Refresh on the newly visible child.
        restore();
    }
}
```

updatePosition()

```
fillRectangle x1=0 y1=0 x2=479 y2=271 area=99
fillRectangle x1=5 y1=5 x2=474 y2=266 area=93
fillRectangle x1=5 y1=5 x2=9 y2=266 area=0
fillRectangle x1=5 y1=5 x2=474 y2=9 area=1
fillRectangle x1=470 y1=5 x2=474 y2=266 area=0
fillRectangle x1=5 y1=262 x2=474 y2=266 area=1
fillRectangle x1=10 y1=36 x2=469 y2=36 area=0
fillRectangle x1=318 y1=74 x2=422 y2=111 area=2
fillRectangle x1=318 y1=74 x2=318 y2=111 area=0
fillRectangle x1=318 y1=74 x2=422 y2=74 area=0
fillRectangle x1=422 y1=74 x2=422 y2=111 area=0
fillRectangle x1=318 y1=111 x2=422 y2=111 area=0
fillRectangle x1=343 y1=187 x2=397 y2=224 area=1
fillRectangle x1=343 y1=187 x2=343 y2=224 area=0
fillRectangle x1=343 y1=187 x2=397 y2=187 area=0
fillRectangle x1=397 y1=187 x2=397 y2=224 area=0
fillRectangle x1=343 y1=224 x2=397 y2=224 area=0
drawImage regionX=0 regionY=0 width=260 height=235 x=10 y=37 alpha=255 area=44
```

restore()

```
fillRectangle x1=0 y1=0 x2=479 y2=271 area=99
fillRectangle x1=5 y1=5 x2=474 y2=266 area=93
fillRectangle x1=5 y1=5 x2=9 y2=266 area=0
fillRectangle x1=5 y1=5 x2=474 y2=9 area=1
fillRectangle x1=470 y1=5 x2=474 y2=266 area=0
fillRectangle x1=5 y1=262 x2=474 y2=266 area=1
fillRectangle x1=10 y1=36 x2=469 y2=36 area=0
fillRectangle x1=318 y1=74 x2=422 y2=111 area=2
fillRectangle x1=318 y1=74 x2=318 y2=111 area=0
fillRectangle x1=318 y1=74 x2=422 y2=74 area=0
fillRectangle x1=422 y1=74 x2=422 y2=111 area=0
fillRectangle x1=318 y1=111 x2=422 y2=111 area=0
fillRectangle x1=343 y1=187 x2=397 y2=224 area=1
fillRectangle x1=343 y1=187 x2=343 y2=224 area=0
fillRectangle x1=343 y1=187 x2=397 y2=187 area=0
fillRectangle x1=397 y1=187 x2=397 y2=224 area=0
fillRectangle x1=343 y1=224 x2=397 y2=224 area=0
drawImage regionX=0 regionY=0 width=260 height=235 x=10 y=37 alpha=255 area=44
32# Area drawn: 493%
```

FIX

- Run the **updatePosition()** code only when the animation is running:

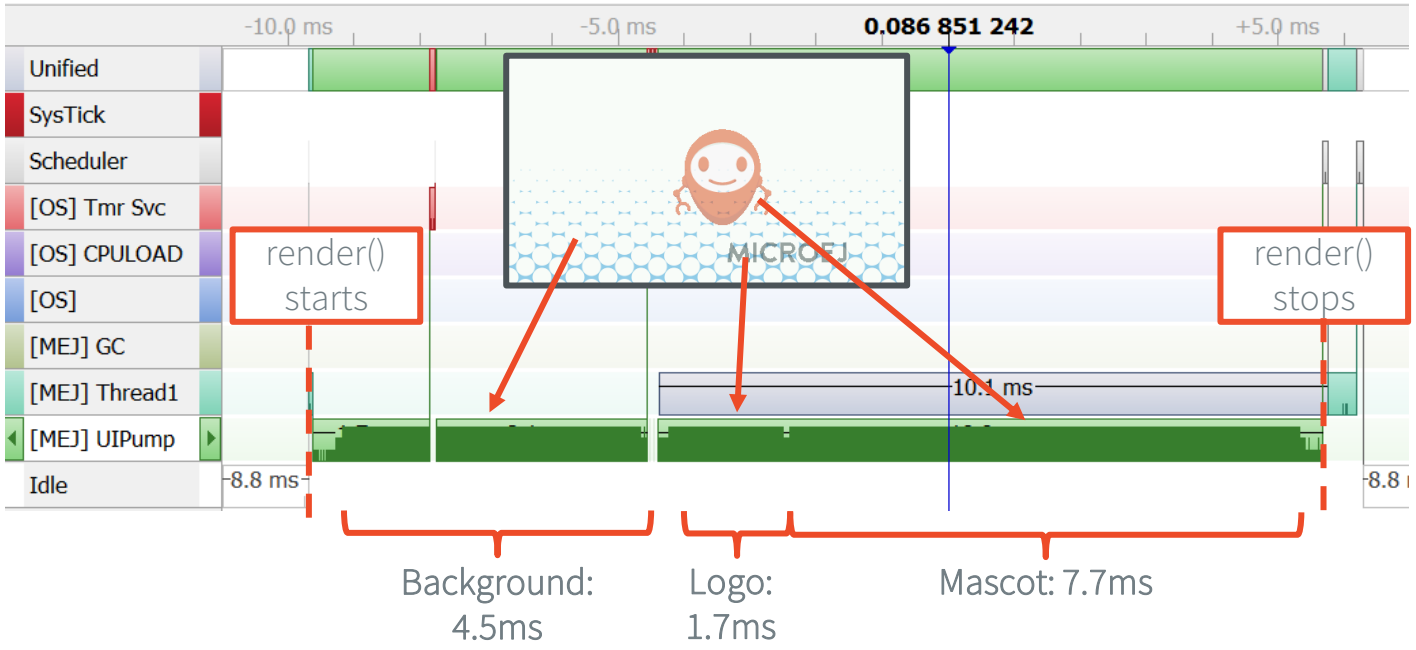
```
public void tick(int value, boolean finished) {
    if (finished) {
        // Refresh on the newly visible child.
        restore();
    } else {
        // Move the 2 pages
        updatePosition(value, leftChild, rightChild);
    }
}
```

```
fillRectangle x1=0 y1=0 x2=479 y2=271 area=99
fillRectangle x1=5 y1=5 x2=474 y2=266 area=93
fillRectangle x1=5 y1=5 x2=9 y2=266 area=0
fillRectangle x1=5 y1=5 x2=474 y2=9 area=1
fillRectangle x1=470 y1=5 x2=474 y2=266 area=0
fillRectangle x1=5 y1=262 x2=474 y2=266 area=1
fillRectangle x1=10 y1=36 x2=469 y2=36 area=0
fillRectangle x1=318 y1=74 x2=422 y2=111 area=2
fillRectangle x1=318 y1=74 x2=318 y2=111 area=0
fillRectangle x1=318 y1=74 x2=422 y2=74 area=0
fillRectangle x1=422 y1=74 x2=422 y2=111 area=0
fillRectangle x1=318 y1=111 x2=422 y2=111 area=0
fillRectangle x1=343 y1=187 x2=397 y2=224 area=1
fillRectangle x1=343 y1=187 x2=343 y2=224 area=0
fillRectangle x1=343 y1=187 x2=397 y2=187 area=0
fillRectangle x1=397 y1=187 x2=397 y2=224 area=0
fillRectangle x1=343 y1=224 x2=397 y2=224 area=0
drawImage regionX=0 regionY=0 width=260 height=235 x=10 y=37 alpha=255 area=44
29# Area drawn: 246%
```

Next step: investigate the 2 first fillRectangle() redrawing ~200% of the screen

SYSTEMVIEW (1/2)

- The SystemView tool can be used to trace the application execution and identify performance bottlenecks.
- Use case: measure the rendering time of images:




```
public void render(GraphicsContext gc) {
    drawTracer.recordEvent(MY_EVT_ID);
    int length = images.size();
    for(int i = -1; ++i<length;) {
        images.get(i).paint(gc);
    }
    drawTracer.recordEventEnd(MY_EVT_ID);
}
```

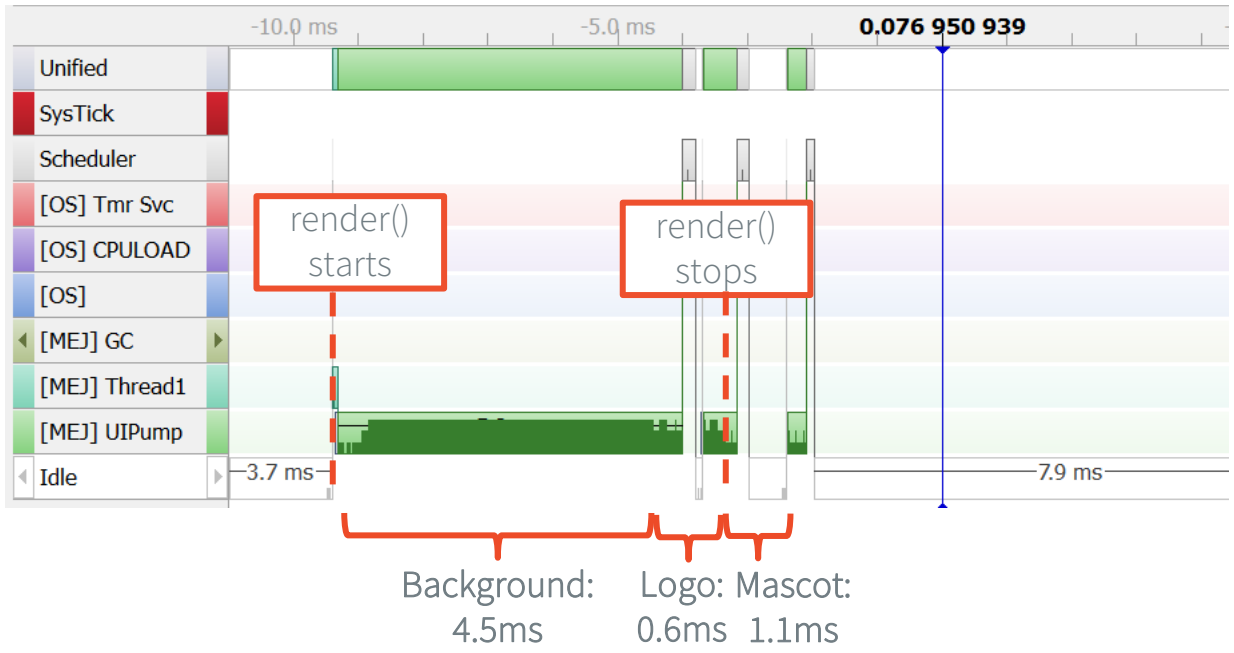
Custom trace event to track the execution of the render() method

- Rendering duration for the 3 images: 13.9ms

- All the images are drawn using the CPU (UIPump thread) → Hardware accelerator can be used to offload the CPU (e.g. SMT32 DMA2D accelerator)

SYSTEMVIEW (2/2)

- Implement the **UI_DRAWING_drawImage** function in the BSP to use the DMA2D accelerator to perform the image drawing:



drawing_dma2d.c

```
UI_DRAWING_drawImage(MICROUI_GraphicsContext *, MICROUI_Image *, jint, jint, jint, jint, jint
DRAWING_Status UI_DRAWING_drawImage(MICROUI_GraphicsContext* gc, MICROUI_Image* ima
        DRAWING_Status ret;
        DRAWING_DMA2D_blending_t dma2d_blending_data;
        if (_drawing_dma2d_is_image_compatible_with_dma2d(gc, image, x_src, y_src,
                LLUI_DISPLAY_setDrawingLimits(dma2d_blending_data.x_dest, dma2d_ble
                _drawing_dma2d_blending_configure(&dma2d_blending_data);
                _drawing_dma2d_blending_start(&dma2d_blending_data);
                ret = DRAWING_RUNNING;
        }
        else {
                UI_DRAWING_SOFT_drawImage(gc, image, x_src, y_src, width, height, x
                ret = DRAWING_DONE;
        }
        return ret;
}
```

Implementation of UI_DRAWING_drawImage using DMA2D

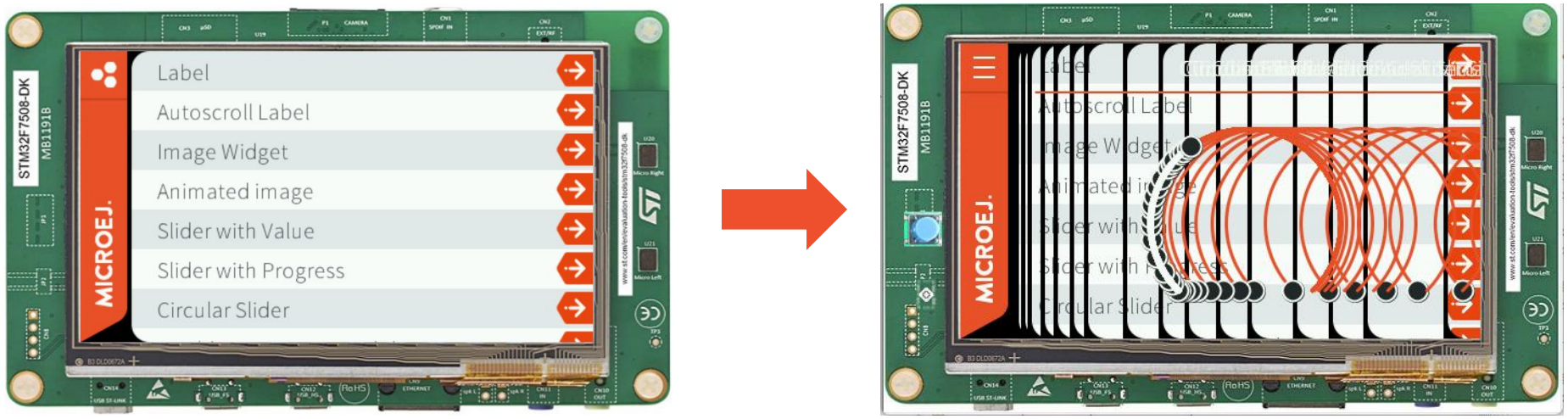
- Rendering duration for the 3 images : 6.2ms (45% faster compared to the software implementation)
  - Blending / Drawing of **Logo** and **Mascot** images is accelerated thanks to DMA2D
- CPU is offloaded during DMA2D operations (Idle state) → other VEE or RTOS threads can run at this time

Debugging Rendering Issues

IDENTIFY GUI RENDERING ISSUES

- Tools:
  - Widget Debug Utilities
  - MWT Debug Utilities

- Example:
  - Debug the rendering issue of a page:



Rendering issue when entering an application page

WIDGET DEBUG UTILITIES (1/3)

The Widget Library provides several Debug Utilities to investigate and troubleshoot GUI applications:

- Print the hierarchy of widgets and styles
- Print the path to a widget
- Count the number of widgets or containers
- Count the maximum depth of a hierarchy
- Print the bounds of a widget
- Print the bounds of all the widgets in a hierarchy

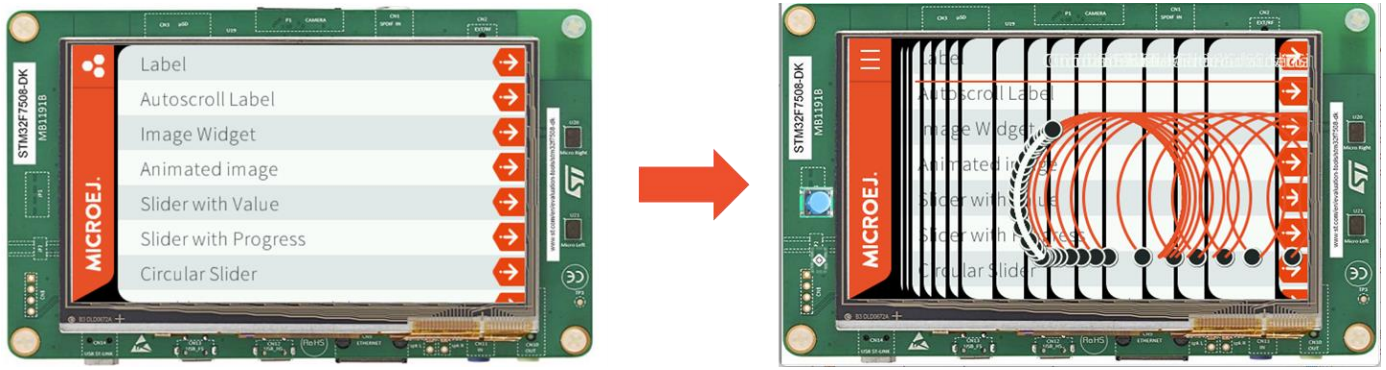
WIDGET DEBUG UTILITIES (2/3)

REPRODUCE THE ISSUE

- In the **com.microej.demo.widget** project, edit the **PageHelper** class of as follow:

```
81    public static void addCommonStyle(CascadingStylesheet stylesheet) {
82        Selector titleButton = new ClassSelector(TITLE_BUTTON_CLASSSELECTOR);
83
84        EditableStyle style = stylesheet.getDefaultStyle();
85        style.setColor(DemoColors.DEFAULT_FOREGROUND);
86        style.setBackground(NoBackground.NO_BACKGROUND); // Uncomment to reproduce the rendering issue
87        // style.setBackground(new RectangularBackground(DemoColors.DEFAULT_BACKGROUND)); // Comment to reproduce the
88        // rendering issue
89        style.setFont(Fonts.getSourceSansPro19px300());
```

- Run on Simulator
- Enter the **Circular Slider Page** to see the rendering issue:



WIDGET DEBUG UTILITIES (3/3)

ROOT CAUSE ANALYSIS

- The background is not redrawn when the page shows up
- Print the style hierarchy of the Desktop to get more information:

CircularSlider.java

```
103     protected void onShown() {
104         HierarchyInspector.printHierarchyStyle(getDesktop().getWidget());
105         super.onShown();
106     }
```

```
SimpleDock{x=0,y=0,w=480,h=272} (color=white, background=NoBackground, font=Font[SourceS
+--SimpleDock{x=0,y=0,w=44,h=272} (color=white, background=NoBackground, font=Font[Sourc
|   +--ImageButton{x=0,y=0,w=44,h=46} (color=white, background=RectangularBackground, bor
|   +--ImageWidget{x=0,y=46,w=44,h=226} (color=white, background=RectangularBackground, f
+--OverlapContainer{x=44,y=0,w=436,h=272} (color=white, background=NoBackground, font=Fo
|   +--SimpleDock{x=44,y=0,w=436,h=272} (color=white, background=NoBackground, border=Fle
|   |   +--Label{x=54,y=0,w=420,h=54} (color=white, background=NoBackground, border=Flexib
|   |   +--SimpleDock{x=54,y=54,w=420,h=198} (color=white, background=NoBackground, font=F
|   |   |   +--CircularSlider{x=54,y=54,w=420,h=198} (color=white, background=NoBackground,
|   +--ImageWidget{x=44,y=0,w=20,h=16} (dimension=OptimalDimension[XY], background=NoBack
|   +--ImageWidget{x=44,y=256,w=20,h=16} (dimension=OptimalDimension[XY], background=NoBa
```

→ There are only transparent backgrounds used in the widget hierarchy

FIX

- Check the default StyleSheet configuration:

```
public static void addCommonStyle(CascadingStylesheet stylesheet) {
    Selector titleButton = new ClassSelector(TITLE_BUTTON_CLASSSELECTOR);

    EditableStyle style = stylesheet.getDefaultStyle();
    style.setColor(DemoColors.DEFAULT_FOREGROUND);
    style.setBackground(NoBackground.NO_BACKGROUND);
```

→ The default style is providing a transparent background.

- The CircularSlider page is not setting the background neither:

*CircularSliderPage.java

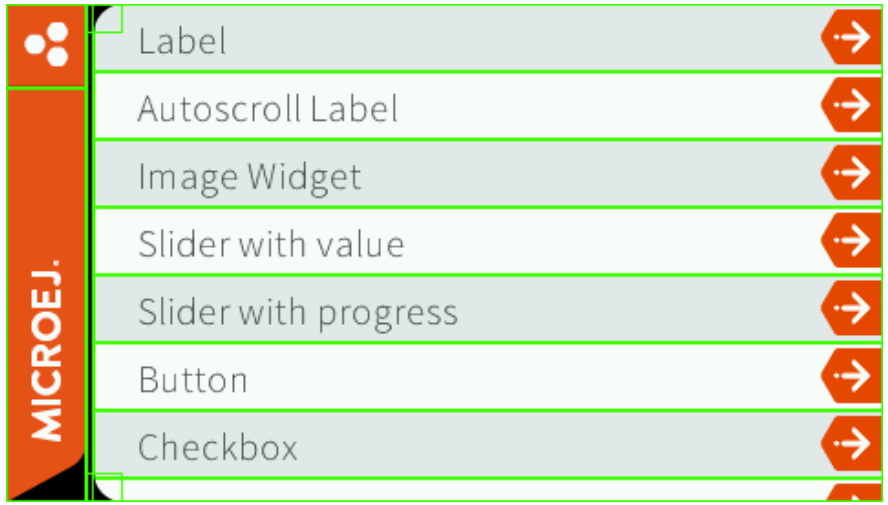
```
38      @Override
39      public void populateStylesheet(CascadingStylesheet stylesheet) {
40          EditableStyle sliderStyle = stylesheet.getSelectorStyle(new TypeSelector(CircularSlider.class));
41          sliderStyle.setFont(Fonts.getSourceSansPro16px700());
42          sliderStyle.setExtraInt(CircularSlider.THICKNESS_ID, THICKNESS);
43          sliderStyle.setExtraInt(CircularSlider.SLIDER_COLOR_ID, DemoColors.DEFAULT_BACKGROUND);
44          sliderStyle.setExtraInt(CircularSlider.GUIDE_THICKNESS_ID, GUIDE_THICKNESS);
45          sliderStyle.setExtraInt(CircularSlider.GUIDE_COLOR_ID, BAR_COLOR);
46          sliderStyle.setExtraInt(CircularSlider.SLIDER_DIAMETER_ID, SLIDER_SIZE);
47          sliderStyle.setExtraInt(CircularSlider.SLIDER_THICKNESS_ID, SLIDER_THICKNESS);
48      }
```

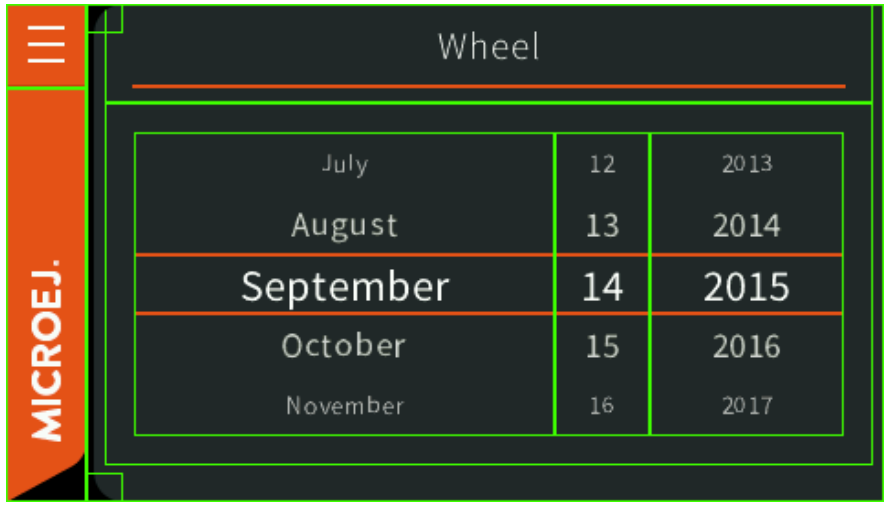
- Fix proposals:
  - Set an opaque background in the default StyleSheet (if possible)
  - Set the background in the StyleSheet of the CircularSlider page (at least on the top level widget of the CircularSlider page → SimpleDock)

MWT DEBUG UTILITIES (1/3)

HIGHLIGHTING THE BOUNDS OF THE WIDGETS

- When designing a UI, it can be pretty convenient to highlight the bounds of each widget. Here are some cases where it helps:
  - Verify if the layout fits the expected design
  - Set the outlines (margin, padding, border)
  - Check the alignment of the widget content inside its bounds
- Example with the Home page and the Wheel page:





MWT DEBUG UTILITIES (2/3)

MONITORING THE RENDER OPERATIONS

- It may not be obvious what/how exactly the UI is rendered, especially if:
  - A widget is re-rendered from a distant part of the application code
  - A specific RenderPolicy is used (e.g. OverlapRenderPolicy)
- The Widget library provides a default monitor implementation that prints the operations on the standard output.
- The logs produced also contain information about what is rendered (widget and area) and what code requested the rendering.
- Example with the RadioButton page (application logs after click):

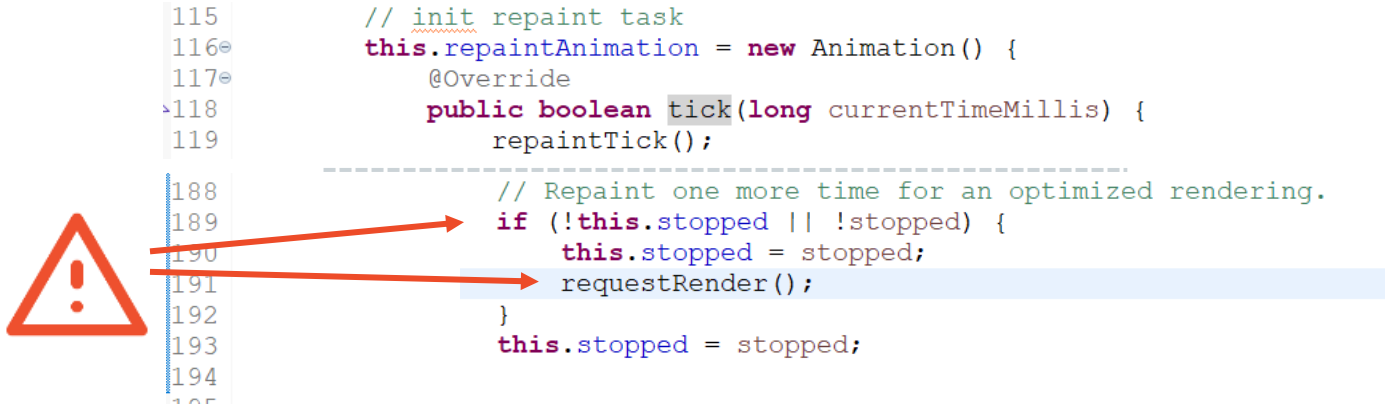
```
rendermonitor@ INFO: Render requested on com.common.PageHelper$2 > SimpleDock > OverlapContainer > SimpleDock > List > RadioButton at {0,0 87x25} of {221,116 87x25} by
com.microej.demo.widget.radiobutton.widget.RadioButtonGroup.setChecked(RadioButtonGroup.java:47)
rendermonitor@ INFO: Render requested on com.common.PageHelper$2 > SimpleDock > OverlapContainer > SimpleDock > List > RadioButton at {0,0 87x25} of {221,166 87x25} by
com.microej.demo.widget.radiobutton.widget.RadioButtonGroup.setChecked(RadioButtonGroup.java:50)
rendermonitor@ INFO: Render executed on  com.common.PageHelper$2 > SimpleDock > OverlapContainer > SimpleDock > List > RadioButton at {-221,-116 87x25} of {221,116 87x25}
rendermonitor@ INFO: Render executed on  com.common.PageHelper$2 > SimpleDock > OverlapContainer > SimpleDock > List > RadioButton at {-221,-141 87x25} of {221,141 87x25}
```

MWT DEBUG UTILITIES (3/3)

MONITORING THE ANIMATORS

- Since an animator ticks its animations as often as possible, the animator may take **100% CPU usage** if none of its animations requests a render.
- MWT notifies when **none of the animations has requested a render** during an animator tick:

```
animatormonitor WARNING: None of the animations has requested a render during the
animator tick. Animations list:
[com.microej.demo.widget.carousel.widget.Carousel$1@2d6d4]
```



- requestRender() is only executed when the widget is moving, or if the user is manipulating it.
  The tick() method loops indefinitely if there is no animation to do.

  → Stop the animation when not required to save CPU time

KEY TAKEWAYS

- SystemView: live analysis of an application with a cross view between RTOS & VEE threads
  → bottlenecks analysis & profiling

- Flush Visualizer: show the pixel surface drawn between two MicroUI frame buffer flushes
  → avoid useless redraws, improve performances

- MWT & Widget Debug utilities: detect issues with the widget hierarchy
  → debug rendering issues

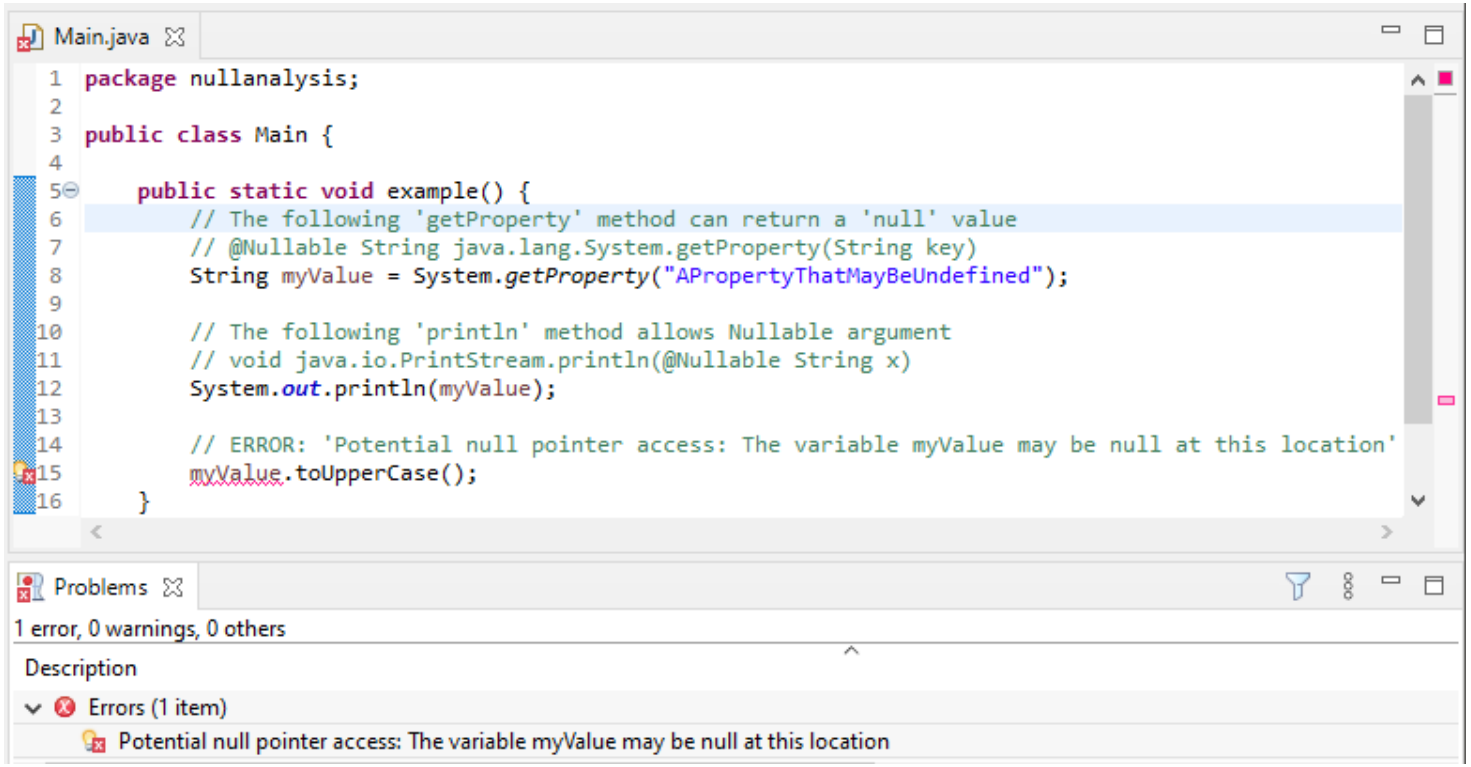
Static Analysis Tools

STATIC ANALYSIS TOOLS (1/3)

NULL ANALYSIS

Static analysis tools are helpful allies to prevent several classes of bugs.

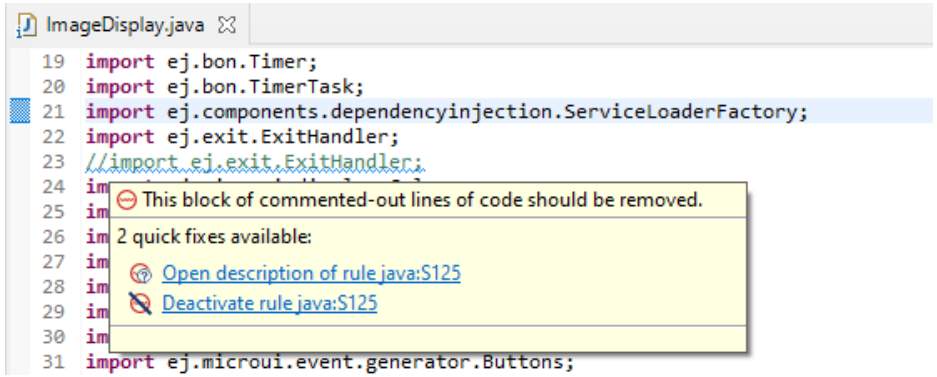
- Use the Null Analysis tool to detect and prevent NullPointerException, one of the most common causes of runtime failure of Java programs.



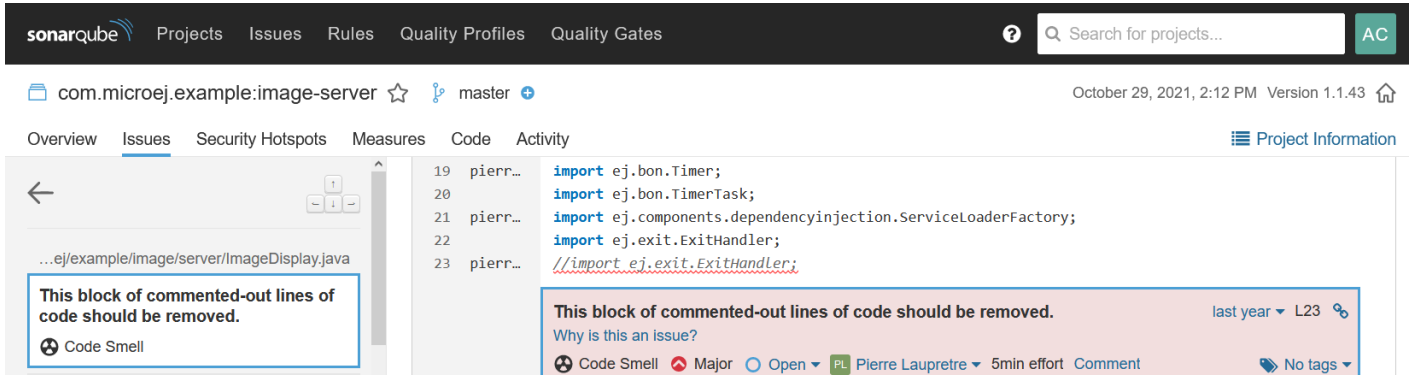
STATIC ANALYSIS TOOLS (2/3)

SONARQUBE

- SonarQube™ is an open source platform for continuous inspection of code quality. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, potential bugs, comments, and architecture.
- SonarQube can be integrated with CI tools to monitor code quality during the project life.
- To set it up on your MicroEJ application project, please refer to this documentation. (configures the set of rules relevant to the context of MicroEJ Application development)



SonarQube code analysis performed inside MICROEJ SDK

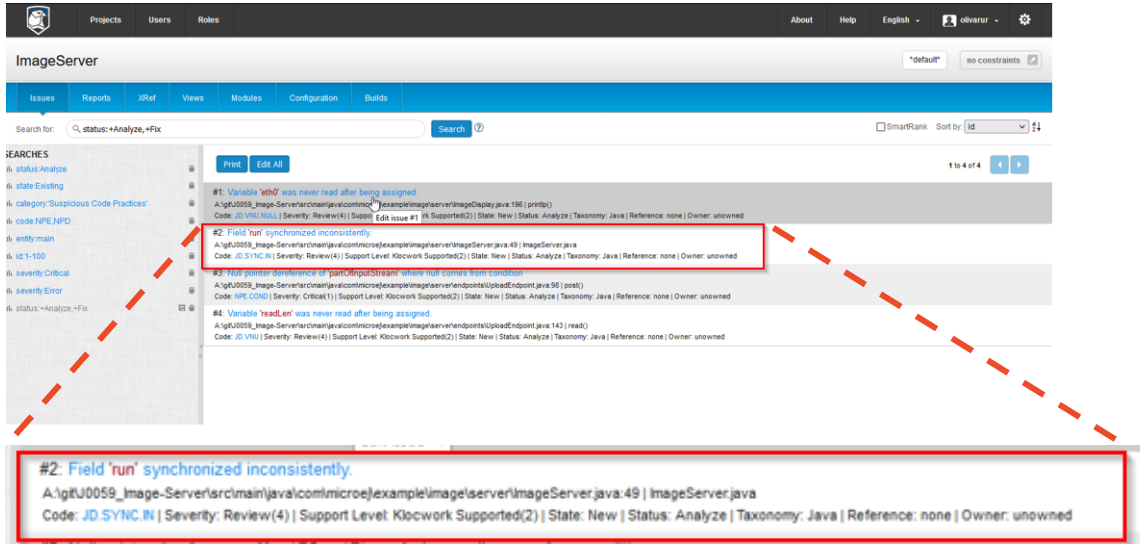


SonarQube code analysis performed on SonarQube server

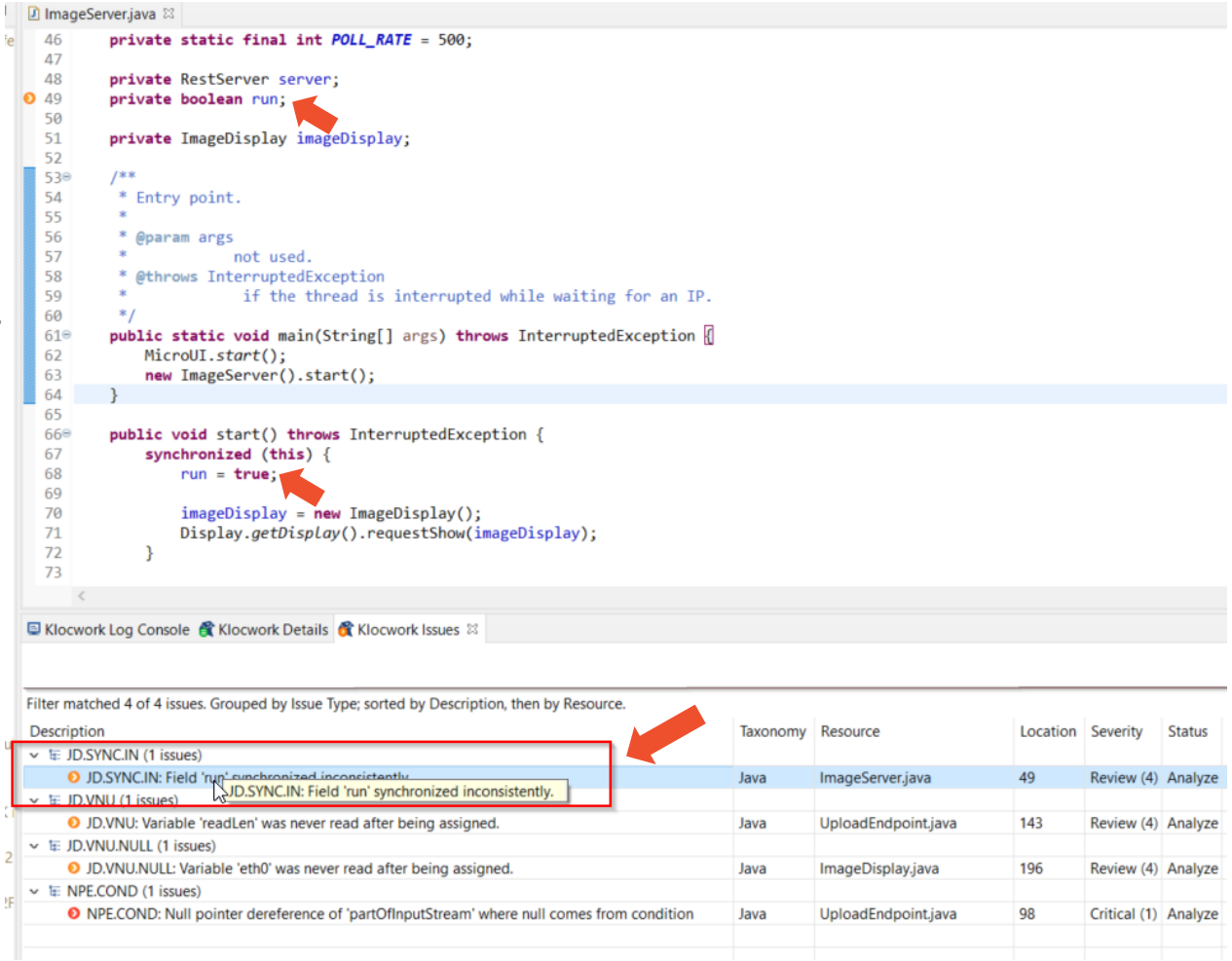
STATIC ANALYSIS TOOLS (3/3)

KLOCWORK

- Klocwork is another code analysis platform that can be integrated to MICROEJ SDK. Documentation can be found here.
- Klocwork can be integrated with CI tools to monitor code quality during the project life.



Klocwork code analysis performed on Klocwork server



Klocwork code analysis performed inside MICROEJ SDK

THANK YOU

for your attention !

MICROEJ®