Mastering MICROEJ SDK Development Tools

Accelerate your product development using MICROEJ SDK Tools

MICROEJ®

DISCLAIMER

All rights reserved. Information, technical data and tutorials contained in this document are proprietary under copyright law of MicroEJ S.A. Without written permission from MicroEJ S.A., copying or sending parts of the document or the entire document by any means to third parties is not permitted. Granted authorizations for using parts of the document or the entire document do not mean MicroEJ S.A. gives public full access rights.

The information contained herein is not warranted to be error-free.

MicroEJ® and all relative logos are trademarks or registered trademarks of MicroEJ S.A. in France and other Countries.

Other trademarks are proprietary of their respective owners.

Java™ is Sun Microsystems' trademark for a technology for developing application software and deploying it in cross-platform, networked environments. When it is used in this site without adding the "™" symbol, it includes implementations of the technology by companies other than Sun. Java™, all Java-based marks and all related logos are trademarks or registered trademarks of Sun Microsystems Inc, in the United States and other Countries.

OVERVIEW

- Goal:
    - Provide an overview of the development tools provided to developers to debug an application
    - Illustrate the use of the development tools
- Development tools categories:
    - Runtime & Post-Mortem Debugging Tools
    - Memory Inspection Tools (debug memory corruption, leaks)
    - Static Analysis Tools
    - GUI Application Debugging Tools (bottlenecks identification, rendering issues)
- The following icons are used in the next slides:
    - 🖥️ : tool only working on Simulator
    - ▣ : tool only working on the Device
    - No icon means that the tool is working on the Device and on the Simulator.

DEVELOPMENT TOOLS OVERVIEW

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

PREREQUISITES

Hardware required:

- NXP i.MX RT1170 Evaluation Kit (EVKB) + micro-USB cable + RK055HDMIPI4MA0 display panel
- More information about the Evaluation Kit: NXP i.MX RT1170 User Manual

Environment Setup:

- Follow the NXP i.MX RT1170 Evaluation Kit Getting Started to setup your environment and run a demo application on the Virtual Device and on the i.MX RT1170 Evaluation Kit.
- Note: the next slides are using **IntelliJ IDEA** with **MicroEJ plugin for IntelliJ IDEA 1.1.0**. This training supports all other available IDEs (Android Studio, VS Code, …)

This training requires the Getting Started to be completed until the
Run an Application on the i.MX RT1170 Evaluation Kit section (included).

⚠ The path to the NXP i.MX RT1170 VEE Port sources should be as short as possible and contain **no whitespace** or **non-ASCII character.**

ENVIRONMENT SETUP (1/4)

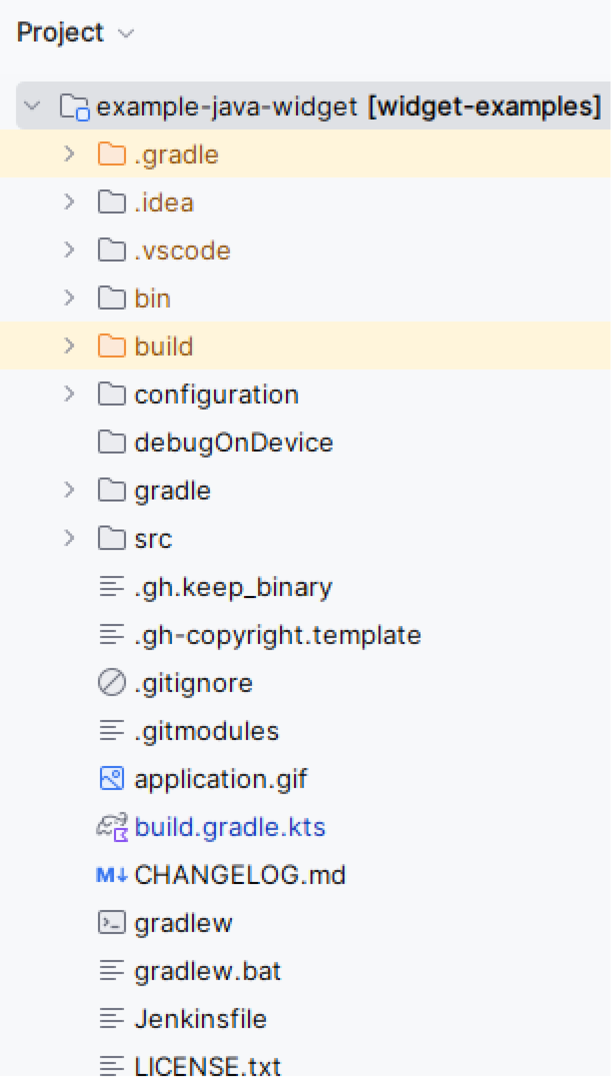
GET TRAINING RESOURCES

- Download and extract the training resources provided with this training, it should contain:
  - **example-java-widget/**
  - **slide-container/**
  - **fill_array_heap_corruption.c**
  - **jdwp-server-1.0.4.jar**

ENVIRONMENT SETUP (2/4)

OPEN THE EXAMPLE-JAVA-WIDGET PROJECT

- Open IntelliJ IDEA.
- If no project is opened in your IDE yet, click on **Open.** Otherwise, go to **File > Open.**
- Browse to the VEE Port sources folder: **example-java-widget**
- Click on **OK**.
- The project sources appear in the Projects view.



```
Project
example-java-widget [widget-examples]
  .gradle
  .idea
  .vscode
  bin
  build
  configuration
  debugOnDevice
  gradle
  src
  .gh.keep_binary
  .gh-copyright.template
  .gitignore
  .gitmodules
  application.gif
  build.gradle.kts
  CHANGELOG.md
  gradlew
  gradlew.bat
  Jenkinsfile
  LICENSE.txt
```

ENVIRONMENT SETUP (3/4)

VEE PORT SELECTION

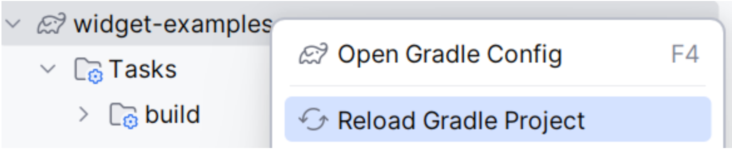
- Get the path to the **NXP i.MX RT1170 VEE Port** (e.g. C:\workspaces\training\nxpvee-mimxrt1170-evk\nxpvee-mimxrt1170-evk)
- Add the path to the VEE Port in the **settings.gradle.kts** file of the **example-java-widget** project:

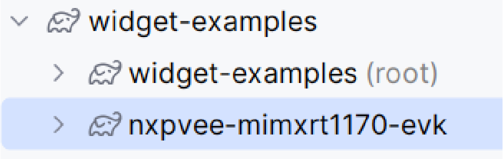
```
rootProject.name = “widget-examples"
includeBuild("C:\\workspaces\\training\\nxpvee-mimxrt1170-evk\\nxpvee-mimxrt1170-evk")
```

- In the **build.gradle.kts** file of the **example-java-widget** project, add the dependency to the VEE Port:

```
dependencies {
   implementation("ej.api:edc:1.3.5")

   //Uncomment the microejVee dependency to set the VEE Port or Kernel to use
   microejVee("com.nxp.vee.mimxrt1170:vee-port:3.0.0")
}
```

- Reload the Gradle project to see the **NXP i.MXRT1170 VEE Port** project in the IDE:



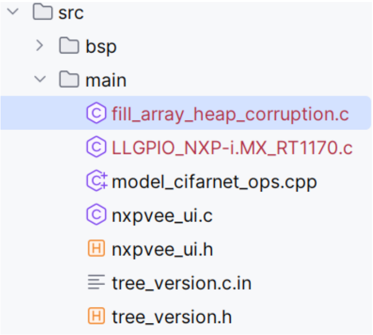


ENVIRONMENT SETUP (4/4)

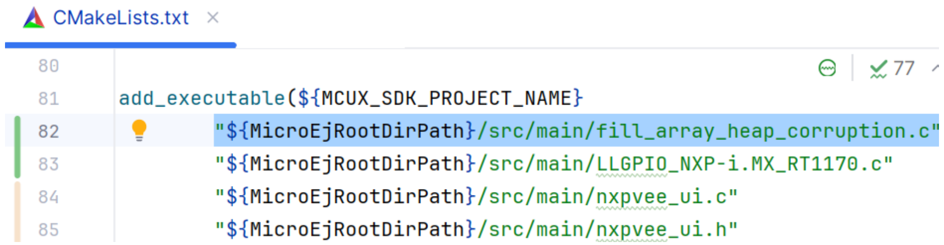
ADD CUSTOM NATIVE FUNCTION

A native function needs to be implemented in the BSP to run the sample on the device:

- Copy/Paste **fill_array_heap_corruption.c** in the BSP project (e.g. copy it in *nxpvee-mimxrt1170-evk /bsp/vee/src/main*).
- Add it to **CMakeLists.txt** (*nxpvee-mimxrt1170-evk/bsp/vee/scripts/armgcc/CMakeLists.txt*):



```
src
  bsp
  main
    fill_array_heap_corruption.c
    LLGPIO_NXP-i.MX_RT1170.c
    model_cifarnet_ops.cpp
    nxpvee_ui.c
    nxpvee_ui.h
    tree_version.c.in
    tree_version.h
```

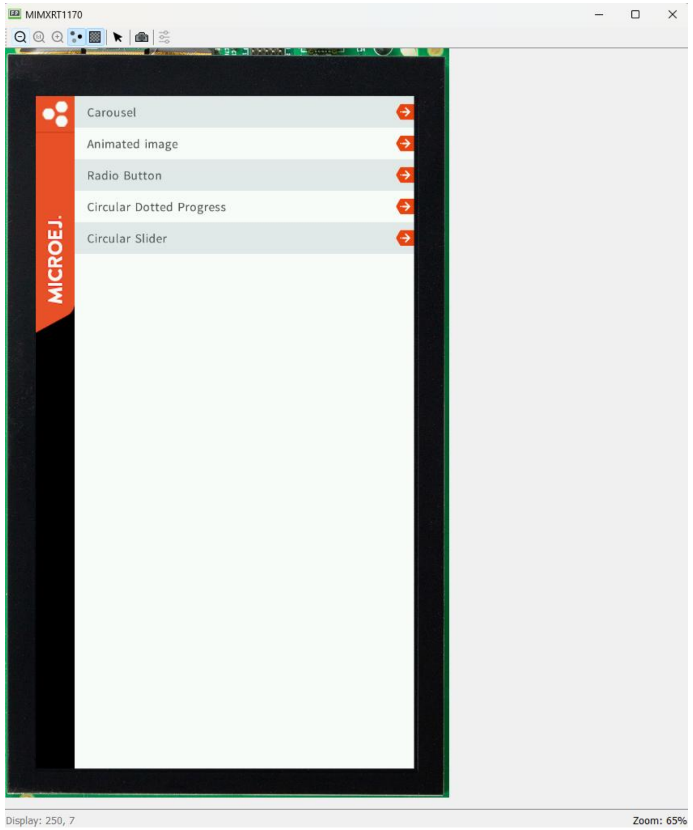


```

add_executable(${MCUX_SDK_PROJECT_NAME}
        "${MicroEjRootDirPath}/src/main/fill_array_heap_corruption.c"
        "${MicroEjRootDirPath}/src/main/LLGPIO_NXP-i.MX_RT1170.c"
        "${MicroEjRootDirPath}/src/main/nxpvee_ui.c"
        "${MicroEjRootDirPath}/src/main/nxpvee_ui.h"
```

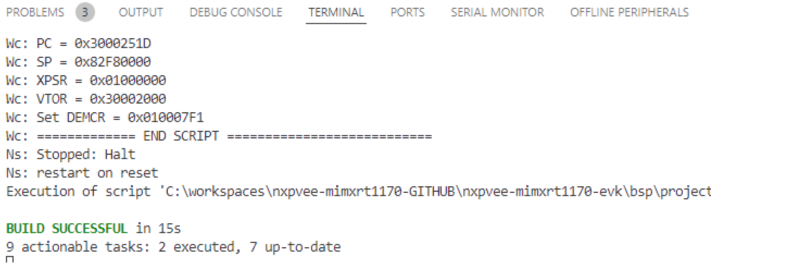
CHECK THE ENVIRONMENT SETUP

Run the sample on the Simulator and on the Device to make sure that the setup is correct:

- Go to the Gradle tasks view
- Run the **runOnSimulator** task:



- Run the **runOnDevice** task. Make sure the hardware is properly setup (cf. Prerequisites)



```
Wc: PC = 0x3000251D
Wc: SP = 0x82F80000
Wc: XPSR = 0x01000000
Wc: VTOR = 0x30002000
Wc: Set DEMCR = 0x010007F1
Wc: ============= END SCRIPT ===========================
Ns: Stopped: Halt
Ns: restart on reset
Execution of script 'C:\workspaces\nxpvee-mimxrt1170-GITHUB\nxpvee-mimxrt1170-evk\bsp\project

BUILD SUCCESSFUL in 15s
9 actionable tasks: 2 executed, 7 up-to-date
```

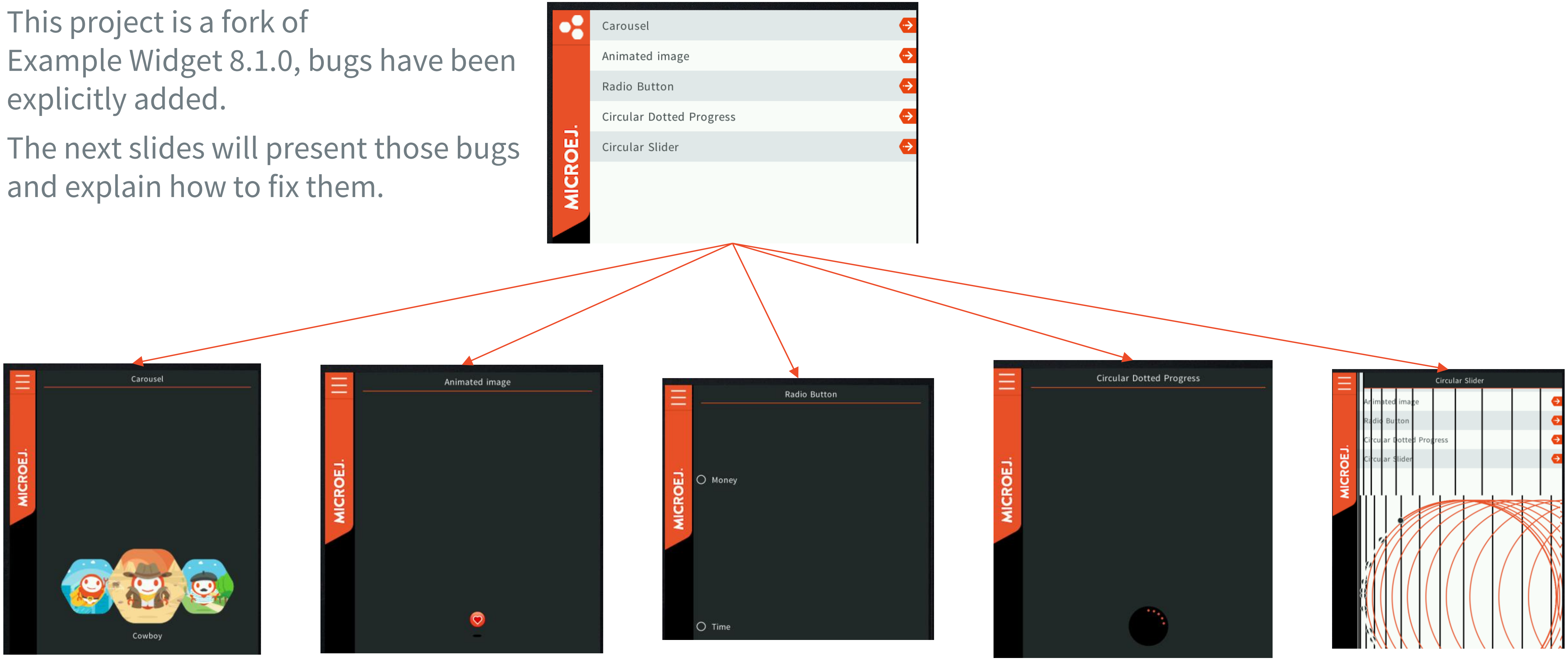
Example-Java-Widget Overview

Introduction to the Example-Java-Widget project

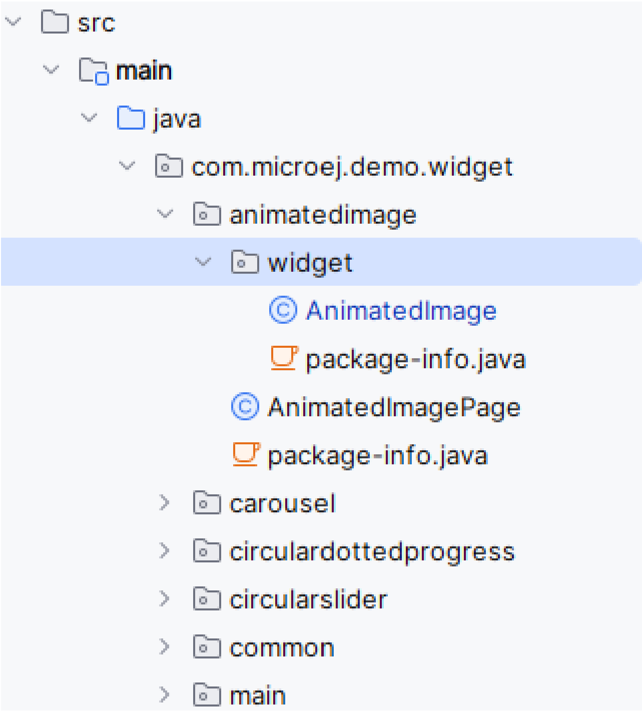
GUI OVERVIEW

This project is a fork of Example Widget 8.1.0, bugs have been explicitly added.

The next slides will present those bugs and explain how to fix them.



PROJECT STRUCTURE



```
src
  main
    java
      com.microej.demo.widget
        animatedimage
          widget
            AnimatedImage
            package-info.java
          AnimatedImagePage
          package-info.java
        carousel
        circulardottedprogress
        circularslider
        common
        main
```

The application is composed of the following packages:

- Animated image: provides a page with an animated image widget producing a GUI freeze
- Carousel: typical page without bugs
- Circular Dotted Progress: provides a page that produces a memory leak
- Circular Slider: provides a page having a rendering issue
- Common: provides a navigation framework and common resources used by the application
- Main: main page of the application
- Radio Button: provides a page generating a heap memory corruption

Each package contains:

- A Page class, describing the layout of the page
- A Widget package containing the widgets of the page

Debug the BSP C Code

Start a Debug Session in VS Code

OVERVIEW

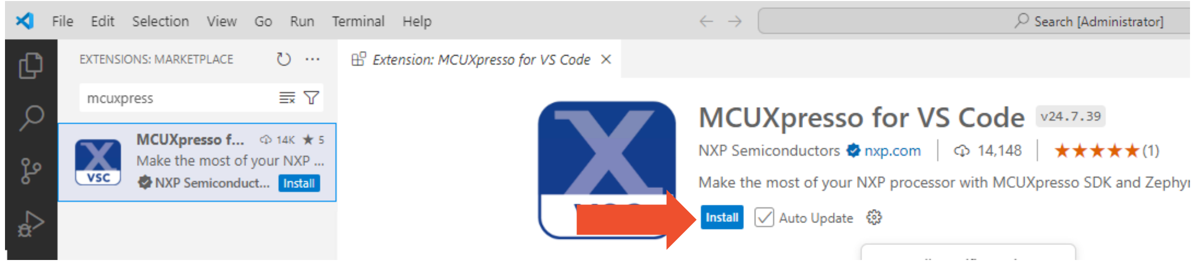
This section describes how to debug C code running on the NXP i.MXRT1170 using Visual Studio Code and GDB.

This section is not MicroEJ specific.

A GDB debugger will be required in the next slides.
You can skip those slides if you already have a GDB client working to debug the NXP i.MXRT1170.

Requirements:

- Install Visual Studio Code (https://code.visualstudio.com/)
- In VS Code, install the MCUXpresso extension for VS Code:



MCUXpresso for VS Code v24.7.39

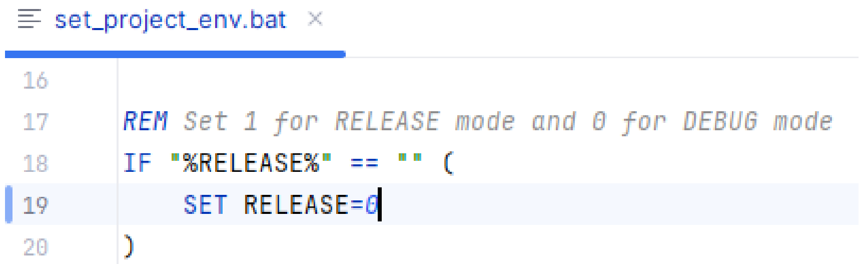
NXP Semiconductors nxp.com | 14,148 | ★★★★★ (1)

Make the most of your NXP processor with MCUXpresso SDK and Zephyr

☑ Auto Update

ENABLE THE DEBUG MODE

- Enable the DEBUG mode by setting **RELEASE=0** in **nxpvee-mimxrt1170-evk\bsp\vee\scripts\set_project_env.bat:**



```

REM Set 1 for RELEASE mode and 0 for DEBUG mode
IF "%RELEASE%" == "" (
    SET RELEASE=0
)
```

BSP DEBUGGING IN VS CODE (1/2)

SELECT THE DEBUG BUILD MODE

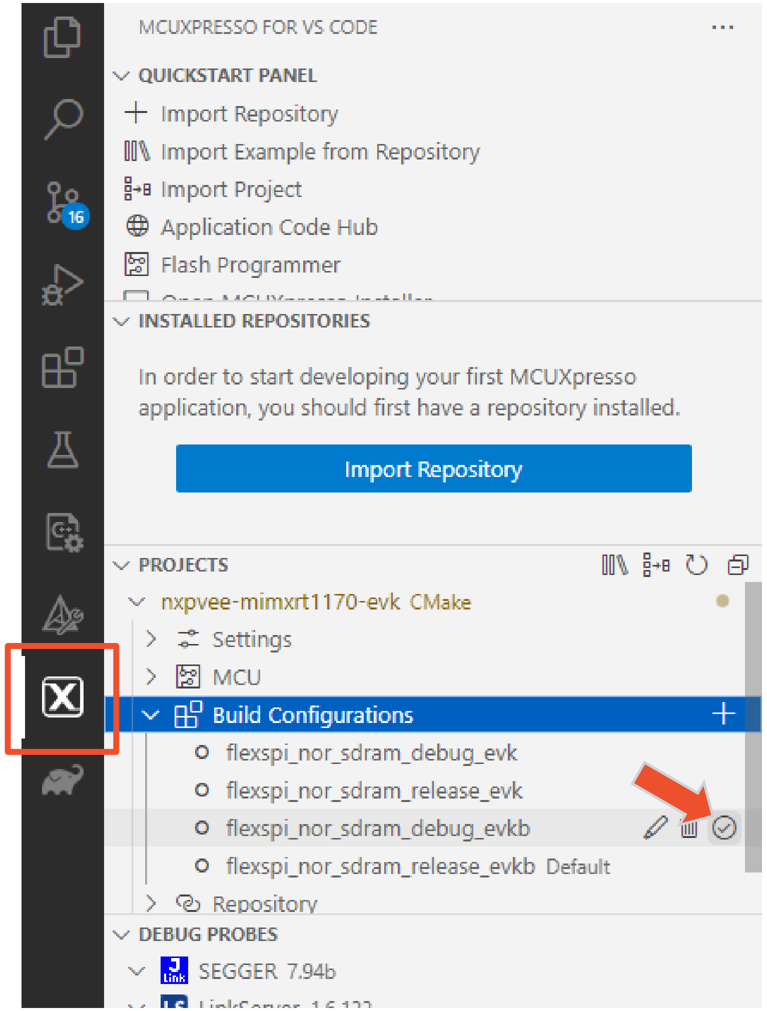
VS Code allows to build, flash and debug embedded projects.

Open the **NXP i.MX RT1170 VEE Port** project in VS Code:

- In VS Code, go to **File > Open Folder…**
- Browse to the VEE Port sources folder: **nxpvee-mimxrt1170-evk**
- Click on **OK**.

Once the project is opened:

- Open the **MCUXpresso** plugin view.
- Open the **Build Configurations** section in the **PROJECTS** view.
- Set **flexspi_nor_sdram_debug_evkb** as the default build configuration.



MCUXPRESSO FOR VS CODE

QUICKSTART PANEL
- Import Repository
- Import Example from Repository
- Import Project
- Application Code Hub
- Flash Programmer

INSTALLED REPOSITORIES

In order to start developing your first MCUXpresso application, you should first have a repository installed.

Import Repository

PROJECTS
- nxpvee-mimxrt1170-evk CMake
  - Settings
  - MCU
  - Build Configurations
    - flexspi_nor_sdram_debug_evk
    - flexspi_nor_sdram_release_evk
    - flexspi_nor_sdram_debug_evkb
    - flexspi_nor_sdram_release_evkb Default
  - Repository

DEBUG PROBES
- SEGGER 7.94b

BSP DEBUGGING IN VS CODE (2/2)

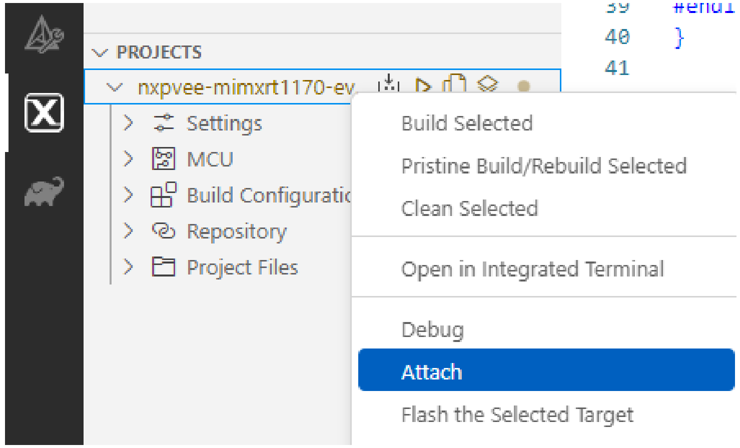
Make sure that the **example-java-widget** project is running on the device.

Open the **NXP i.MX RT1170 VEE Port** project in VS Code:

- In VS Code, go to **File > Open Folder…**
- Browse to the VEE Port sources folder: **nxpvee-mimxrt1170-evk**
- Click on **OK**.

Once the project is opened:

- Open the **MCUXpresso** plugin view.
- Right-click on the project.
- Select **Debug** (as the application is already running on the device).

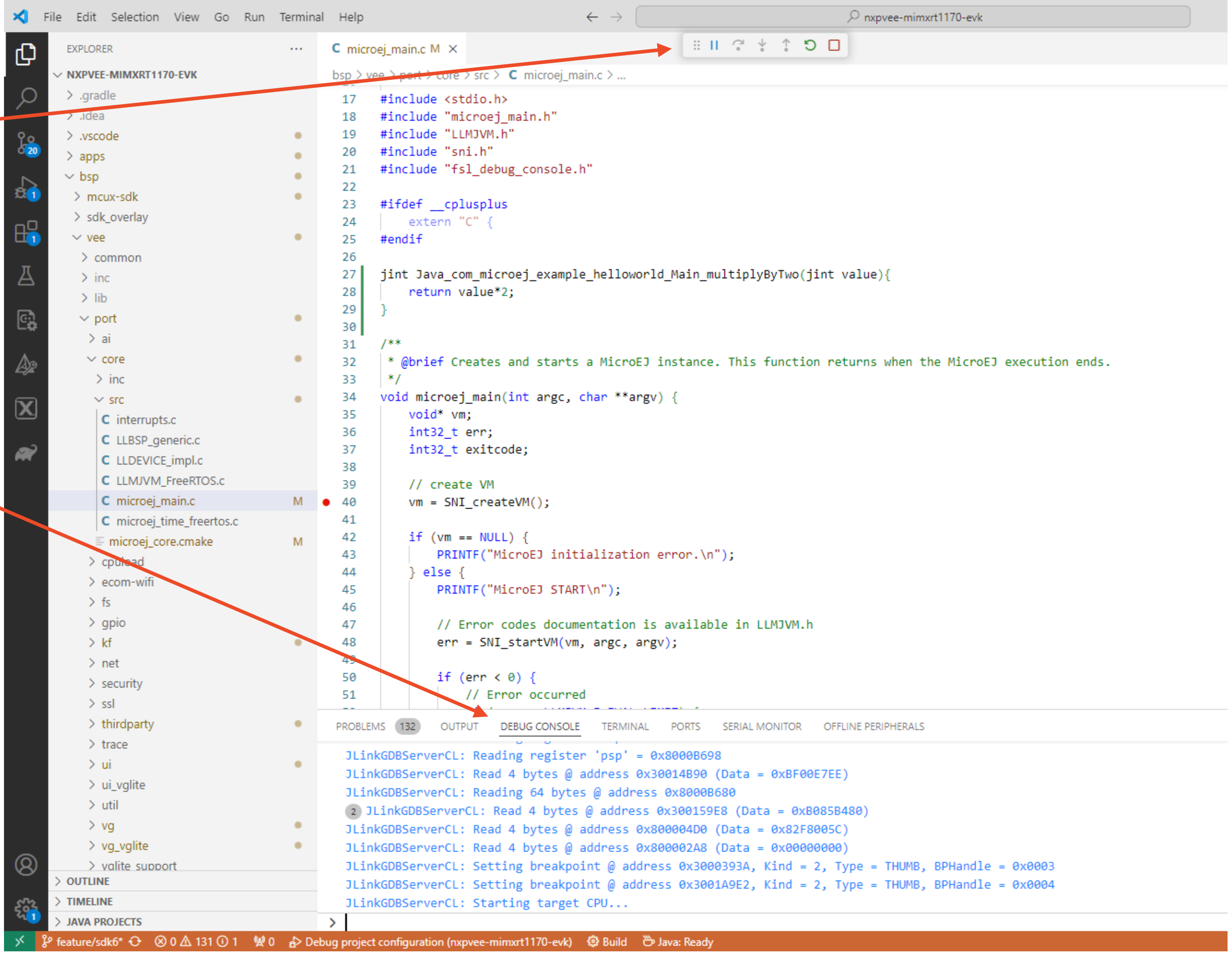


DEBUG VIEW IN VS CODE

The debug view opens and the application runs on the device:

A control bar is available to start/stop/pause the debugging

The Debug Console allows to type GDB commands Refer to VS Code documentation for more information.



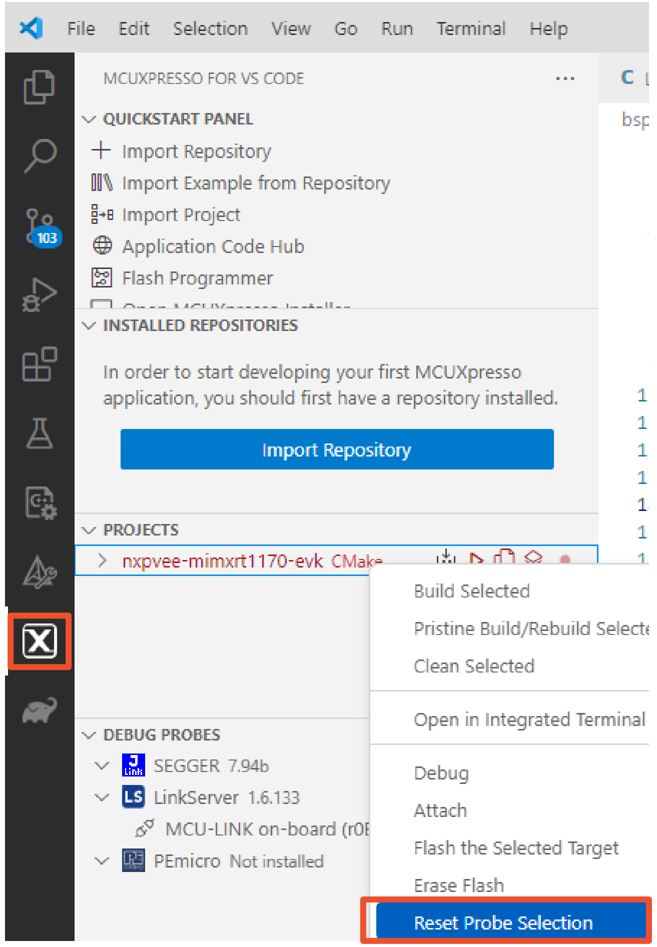
```c
#include <stdio.h>
#include "microej_main.h"
#include "LLMJVM.h"
#include "sni.h"
#include "fsl_debug_console.h"

#ifdef __cplusplus
    extern "C" {
#endif

jint Java_com_microej_example_helloworld_Main_multiplyByTwo(jint value){
    return value*2;
}

/**
 * @brief Creates and starts a MicroEJ instance. This function returns when the MicroEJ execution ends.
 */
void microej_main(int argc, char **argv) {
    void* vm;
    int32_t err;
    int32_t exitcode;

    // create VM
    vm = SNI_createVM();

    if (vm == NULL) {
        PRINTF("MicroEJ initialization error.\n");
    } else {
        PRINTF("MicroEJ START\n");

        // Error codes documentation is available in LLMJVM.h
        err = SNI_startVM(vm, argc, argv);

        if (err < 0) {
            // Error occurred
```

```
JLinkGDBServerCL: Reading register 'psp' = 0x8000B698
JLinkGDBServerCL: Read 4 bytes @ address 0x30014B90 (Data = 0xBF00E7EE)
JLinkGDBServerCL: Reading 64 bytes @ address 0x8000B680
2 JLinkGDBServerCL: Read 4 bytes @ address 0x300159E8 (Data = 0xB085B480)
JLinkGDBServerCL: Read 4 bytes @ address 0x800004D0 (Data = 0x82F8005C)
JLinkGDBServerCL: Read 4 bytes @ address 0x800002A8 (Data = 0x00000000)
JLinkGDBServerCL: Setting breakpoint @ address 0x3000393A, Kind = 2, Type = THUMB, BPHandle = 0x0003
JLinkGDBServerCL: Setting breakpoint @ address 0x3001A9E2, Kind = 2, Type = THUMB, BPHandle = 0x0004
JLinkGDBServerCL: Starting target CPU...
>
```

TROUBLESHOOTING

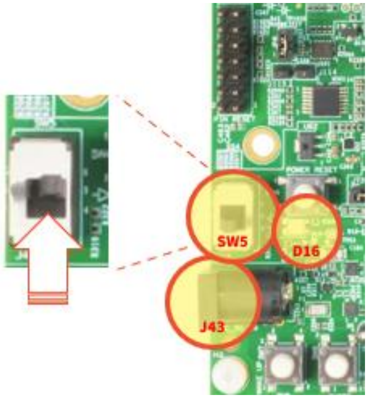
In case of connection issue to the target, reset the debug probe selection via the MCUXpresso plugin:

- Select the MCUXpresso plugin in the left banner.
- Right-click on the project name and select **Reset Probe Selection.**
- Start the debug again.



In order to start developing your first MCUXpresso application, you should first have a repository installed.

If the issue persists, unplug/plug the USB cable and turn OFF/ON the device:



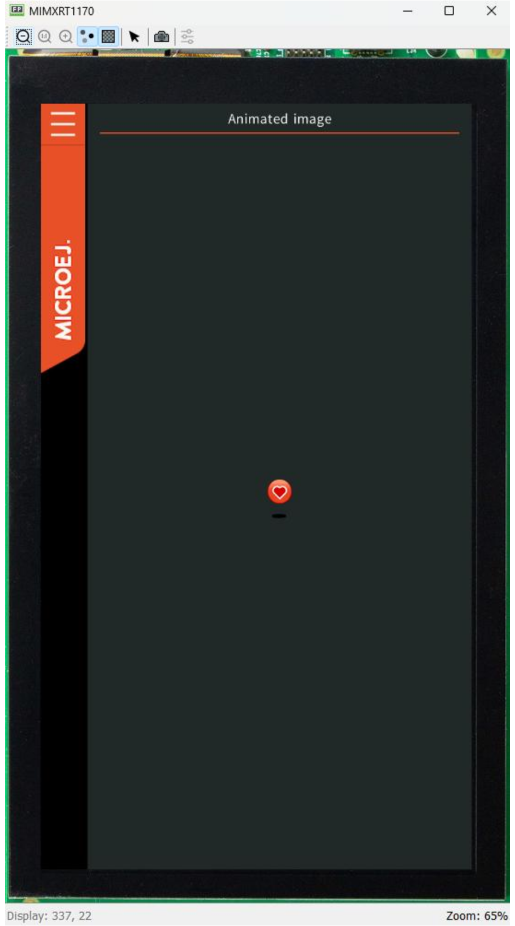
Runtime & Post-Mortem Debugging Tools

Debug the Application code

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



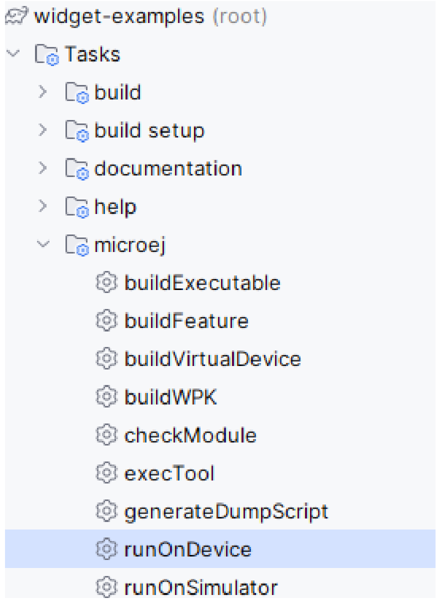
GUI freeze when entering a page

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

DEBUG A DEADLOCK IN AN APPLICATION

REPRODUCE THE ISSUE

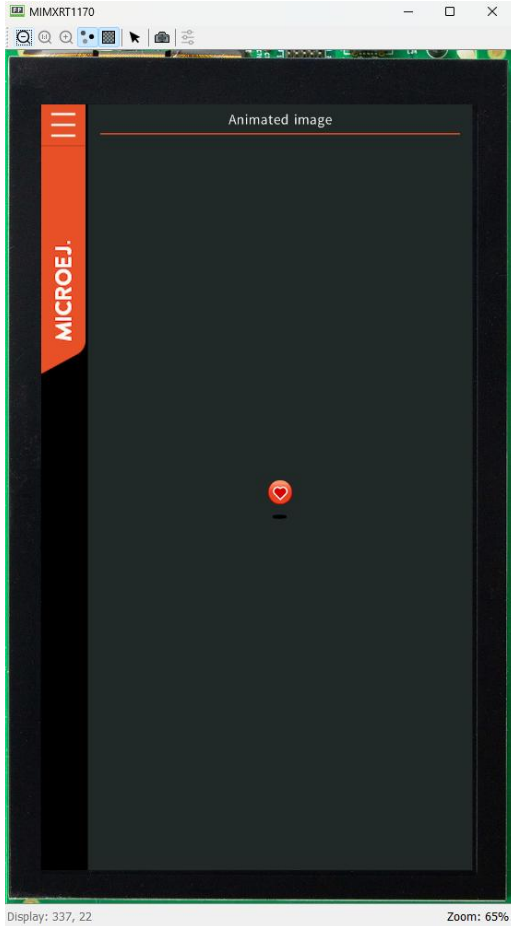
- Run the **example-java-widget** project the on Device:



```
widget-examples (root)
  Tasks
    build
    build setup
    documentation
    help
    microej
      buildExecutable
      buildFeature
      buildVirtualDevice
      buildWPK
      checkModule
      execTool
      generateDumpScript
      runOnDevice
      runOnSimulator
```

- Enter the **Animated Image** page.

The GUI should freeze after the screen transition:



CORE ENGINE VM DUMP (1/4)

- 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.

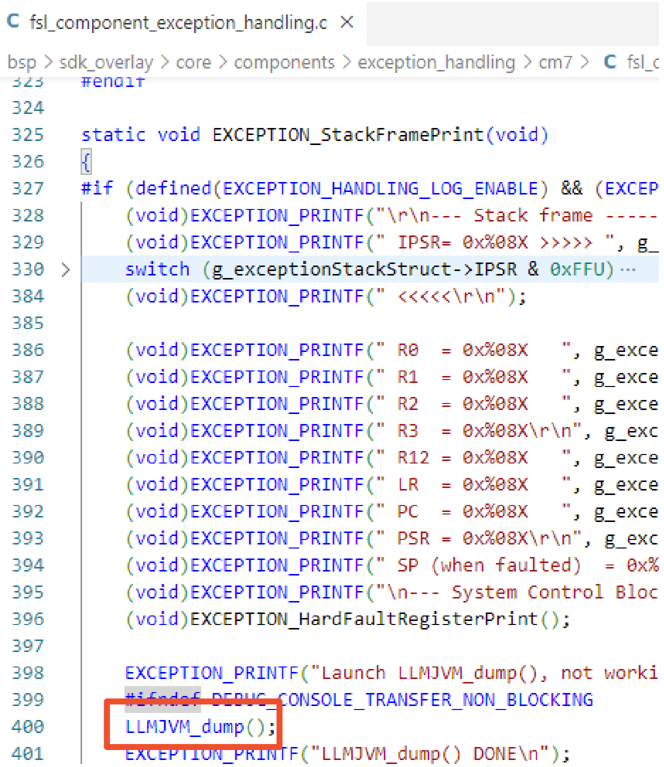
- What?
  - Prints the state of the MicroEJ Core Engine to the standard output stream.
  - For each Java thread, the Java stack trace, the name, the state and the priority are printed.

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

CORE ENGINE VM DUMP (2/4)

HOW-TO?

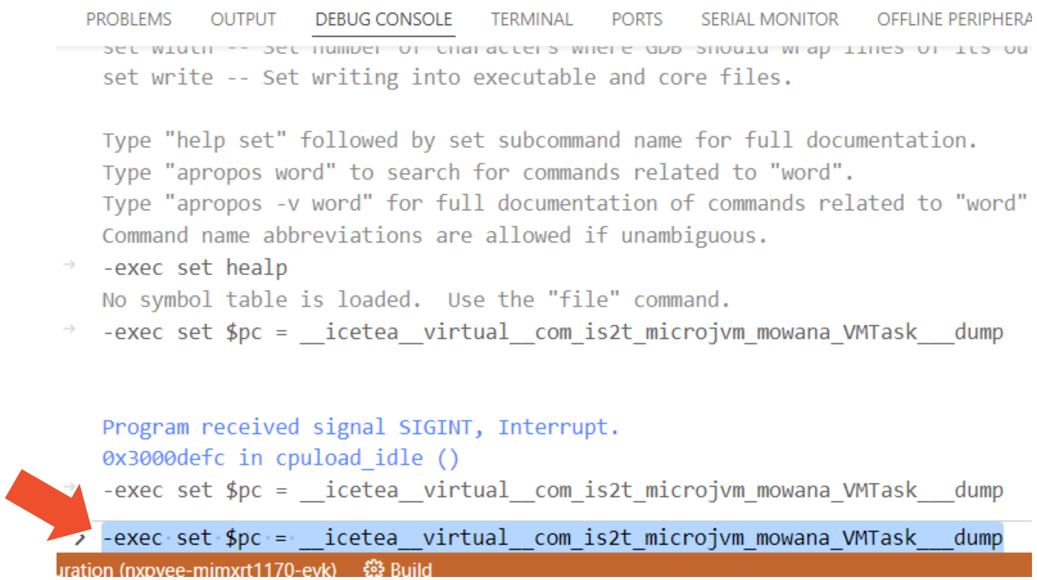
- Example of LLMJVM Dump triggered from a fault handler:



fsl_component_exception_handling.c
bsp > sdk_overlay > core > components > exception_handling > cm7 > fsl_c

```
#endif

static void EXCEPTION_StackFramePrint(void)
{
#if (defined(EXCEPTION_HANDLING_LOG_ENABLE) && (EXCEP
    (void)EXCEPTION_PRINTF("\r\n--- Stack frame -----
    (void)EXCEPTION_PRINTF(" IPSR= 0x%08X >>>>> ", g_
    switch (g_exceptionStackStruct->IPSR & 0xFFU) ···
    (void)EXCEPTION_PRINTF(" <<<<<\r\n");

    (void)EXCEPTION_PRINTF(" R0  = 0x%08X   ", g_exce
    (void)EXCEPTION_PRINTF(" R1  = 0x%08X   ", g_exce
    (void)EXCEPTION_PRINTF(" R2  = 0x%08X   ", g_exce
    (void)EXCEPTION_PRINTF(" R3  = 0x%08X\r\n", g_exc
    (void)EXCEPTION_PRINTF(" R12 = 0x%08X   ", g_exce
    (void)EXCEPTION_PRINTF(" LR  = 0x%08X   ", g_exce
    (void)EXCEPTION_PRINTF(" PC  = 0x%08X   ", g_exce
    (void)EXCEPTION_PRINTF(" PSR = 0x%08X\r\n", g_exc
    (void)EXCEPTION_PRINTF(" SP (when faulted)  = 0x%
    (void)EXCEPTION_PRINTF("\n--- System Control Bloc
    (void)EXCEPTION_HardFaultRegisterPrint();

    EXCEPTION_PRINTF("Launch LLMJVM_dump(), not worki
    #ifndef DEBUG_CONSOLE_TRANSFER_NON_BLOCKING
    LLMJVM_dump();
    EXCEPTION_PRINTF("LLMJVM_dump() DONE\n");
```

- Trigger the LLMJVM Dump from the debugger (see next slide):

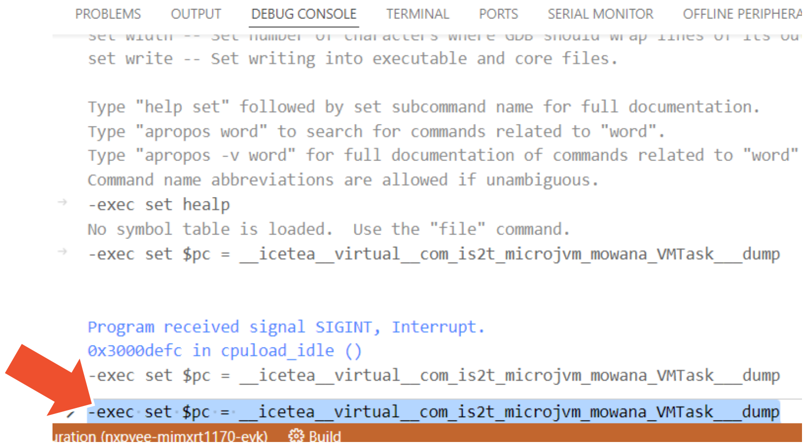


```
set width -- Set number of characters where GDB should wrap lines of its ou
set write -- Set writing into executable and core files.

Type "help set" followed by set subcommand name for full documentation.
Type "apropos word" to search for commands related to "word".
Type "apropos -v word" for full documentation of commands related to "word"
Command name abbreviations are allowed if unambiguous.
-exec set healp
No symbol table is loaded.  Use the "file" command.
-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump


Program received signal SIGINT, Interrupt.
0x3000defc in cpuload_idle ()
-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump
-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump
```

CORE ENGINE VM DUMP (3/4)

TRIGGER THE LLMJVM DUMP FROM THE DEBUGGER (VS CODE / GDB)

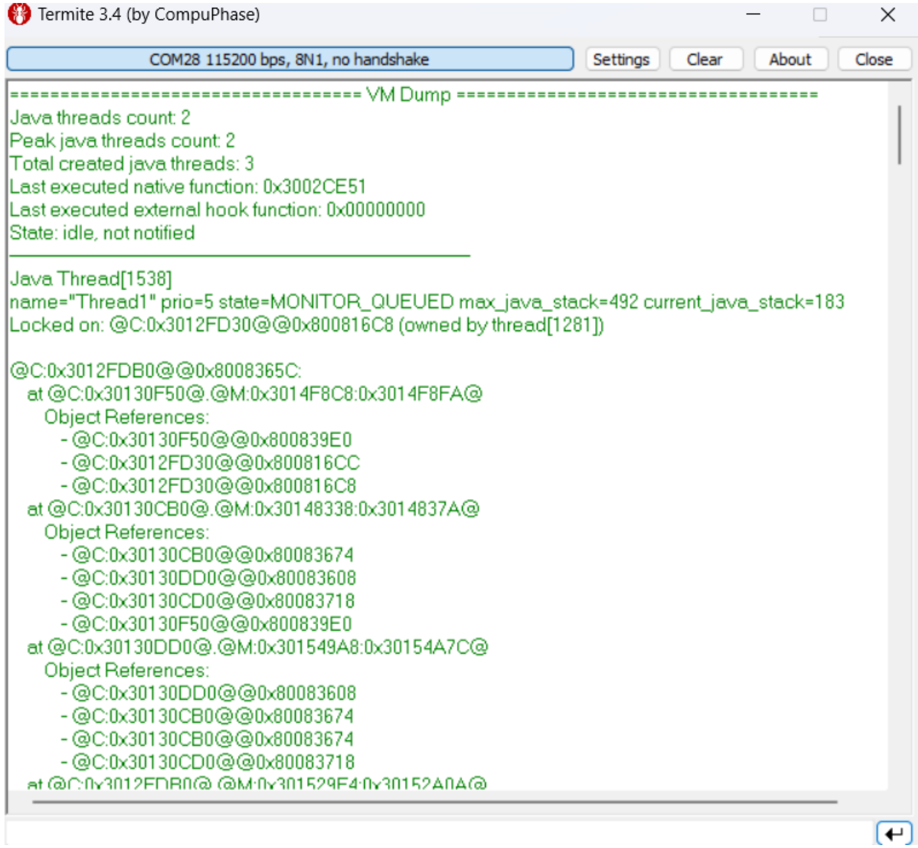
- Start a serial terminal to get the application execution traces
- Start the debug session in VS Code
- Click the “Play” button to start the application
- Enter the **Animated Image** page
- Click the “Pause” button
- Run the following command in the **Debug Console**:



```
set width -- Set number of characters where GDB should wrap lines of its ou
set write -- Set writing into executable and core files.

Type "help set" followed by set subcommand name for full documentation.
Type "apropos word" to search for commands related to "word".
Type "apropos -v word" for full documentation of commands related to "word"
Command name abbreviations are allowed if unambiguous.
-exec set healp
No symbol table is loaded.  Use the "file" command.
-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump


Program received signal SIGINT, Interrupt.
0x3000defc in cpuload_idle ()
-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump

-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump
```

```
-exec set $pc = __icetea__virtual__com_is2t_microjvm_mowana_VMTask___dump
```

- Click the “Play” button, the VM dump can be see in the serial terminal



Termite 3.4 (by CompuPhase)
COM28 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
================================== VM Dump ==================================
Java threads count: 2
Peak java threads count: 2
Total created java threads: 3
Last executed native function: 0x3002CE51
Last executed external hook function: 0x00000000
State: idle, not notified
_________________________________________

Java Thread[1538]
name="Thread1" prio=5 state=MONITOR_QUEUED max_java_stack=492 current_java_stack=183
Locked on: @C:0x3012FD30@@0x800816C8 (owned by thread[1281])

@C:0x3012FDB0@@0x8008365C:
  at @C:0x30130F50@.@M:0x3014F8C8:0x3014F8FA@
    Object References:
      - @C:0x30130F50@@0x800839E0
      - @C:0x3012FD30@@0x800816CC
      - @C:0x3012FD30@@0x800816C8
  at @C:0x30130CB0@.@M:0x30148338:0x3014837A@
    Object References:
      - @C:0x30130CB0@@0x80083674
      - @C:0x30130DD0@@0x80083608
      - @C:0x30130CD0@@0x80083718
      - @C:0x30130F50@@0x800839E0
  at @C:0x30130DD0@.@M:0x301549A8:0x30154A7C@
    Object References:
      - @C:0x30130DD0@@0x80083608
      - @C:0x30130CB0@@0x80083674
      - @C:0x30130CB0@@0x80083674
      - @C:0x30130CD0@@0x80083718
  at @C:0x3012FDB0@.@M:0x301529F4:0x30152A0A@
```

CORE ENGINE VM DUMP (4/4)

EXAMPLE OF DUMP

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

```
================================== 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
```

SIMULATOR & DEVICE DEBUGGER

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 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.

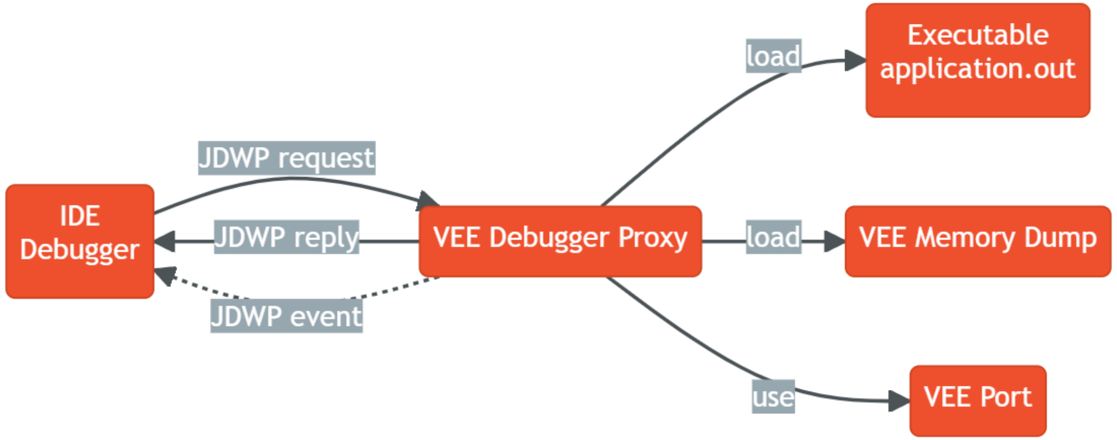
Debug on Device

Debug the Application Code
on the Device

VEE DEBUGGER PROXY PRINCIPLE

The VEE Debugger Proxy is an implementation of the Java Debug Wire protocol (JDWP) for debugging Applications executed by MICROEJ VEE.

- VEE Debugger Proxy principle:

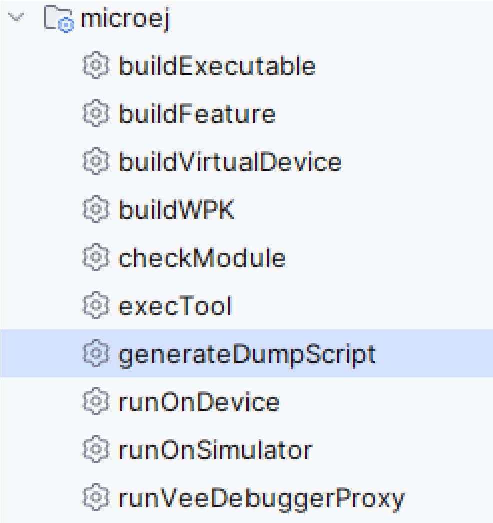


- Available since Architecture 8.1

- 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. On the MicroEJ Simulator, run a Remote Java Application Debugging session

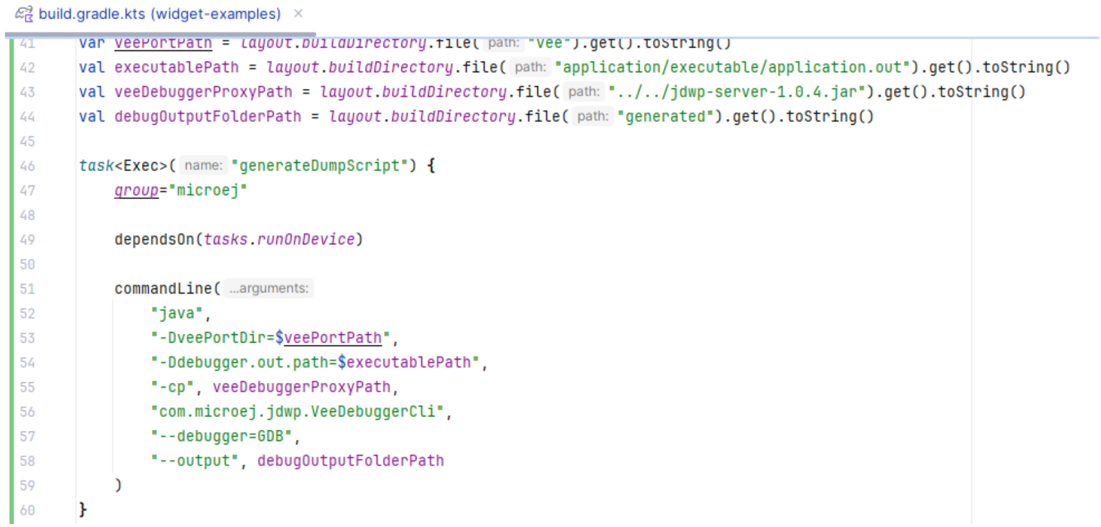
GENERATE THE VEE MEMORY DUMP SCRIPT (1/2)

- The VEE Debugger Proxy tool **jdwp-server-1.0.4** is required to generate the VEE memory dump script.
  The tool is provided in the training package.
- The **example-java-widget** project provides the **generateDumpScript** task that allows the user to generate the VEE memory dump script:



```
microej
    buildExecutable
    buildFeature
    buildVirtualDevice
    buildWPK
    checkModule
    execTool
    generateDumpScript
    runOnDevice
    runOnSimulator
    runVeeDebuggerProxy
```

This task is declared in the **build.gradle.kts**, it is based on the command line provided in the VEE Debugger Proxy documentation:



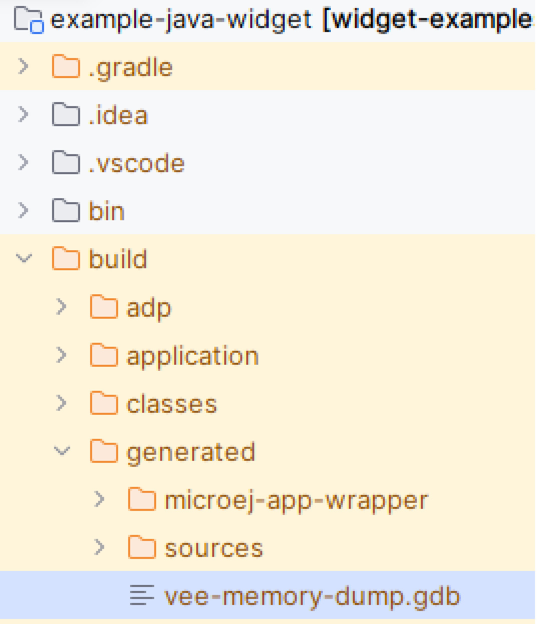
```
var veePortPath = layout.buildDirectory.file( path: "vee").get().toString()
val executablePath = layout.buildDirectory.file( path: "application/executable/application.out").get().toString()
val veeDebuggerProxyPath = layout.buildDirectory.file( path: "../../jdwp-server-1.0.4.jar").get().toString()
val debugOutputFolderPath = layout.buildDirectory.file( path: "generated").get().toString()

task<Exec>( name: "generateDumpScript") {
    group="microej"

    dependsOn(tasks.runOnDevice)

    commandLine( ...arguments:
        "java",
        "-DveePortDir=$veePortPath",
        "-Ddebugger.out.path=$executablePath",
        "-cp", veeDebuggerProxyPath,
        "com.microej.jdwp.VeeDebuggerCli",
        "--debugger=GDB",
        "--output", debugOutputFolderPath
    )
}
```

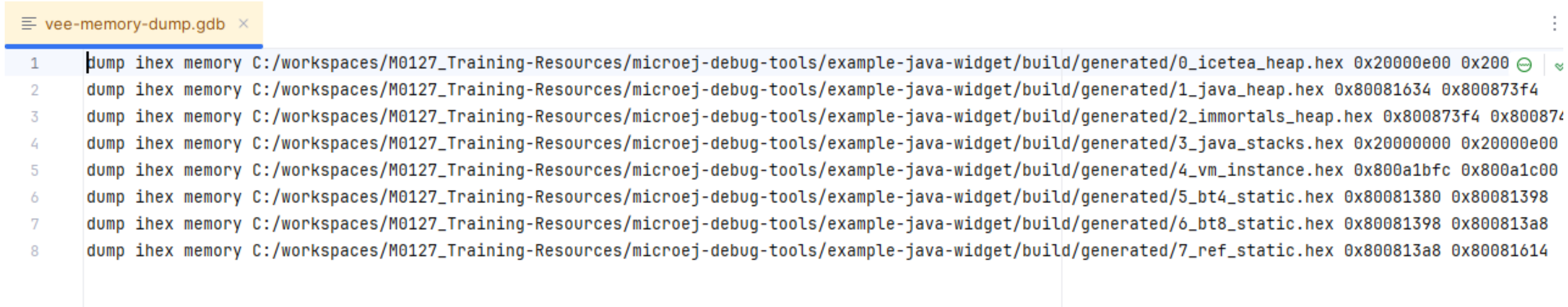
Note that this task is configured to generate a **GDB** dump script.

GENERATE THE VEE MEMORY DUMP SCRIPT (2/2)

Run the **generateDumpScript** task:

- The application is built and flashed on the device,
- The dump script is generated in the build/generated folder:

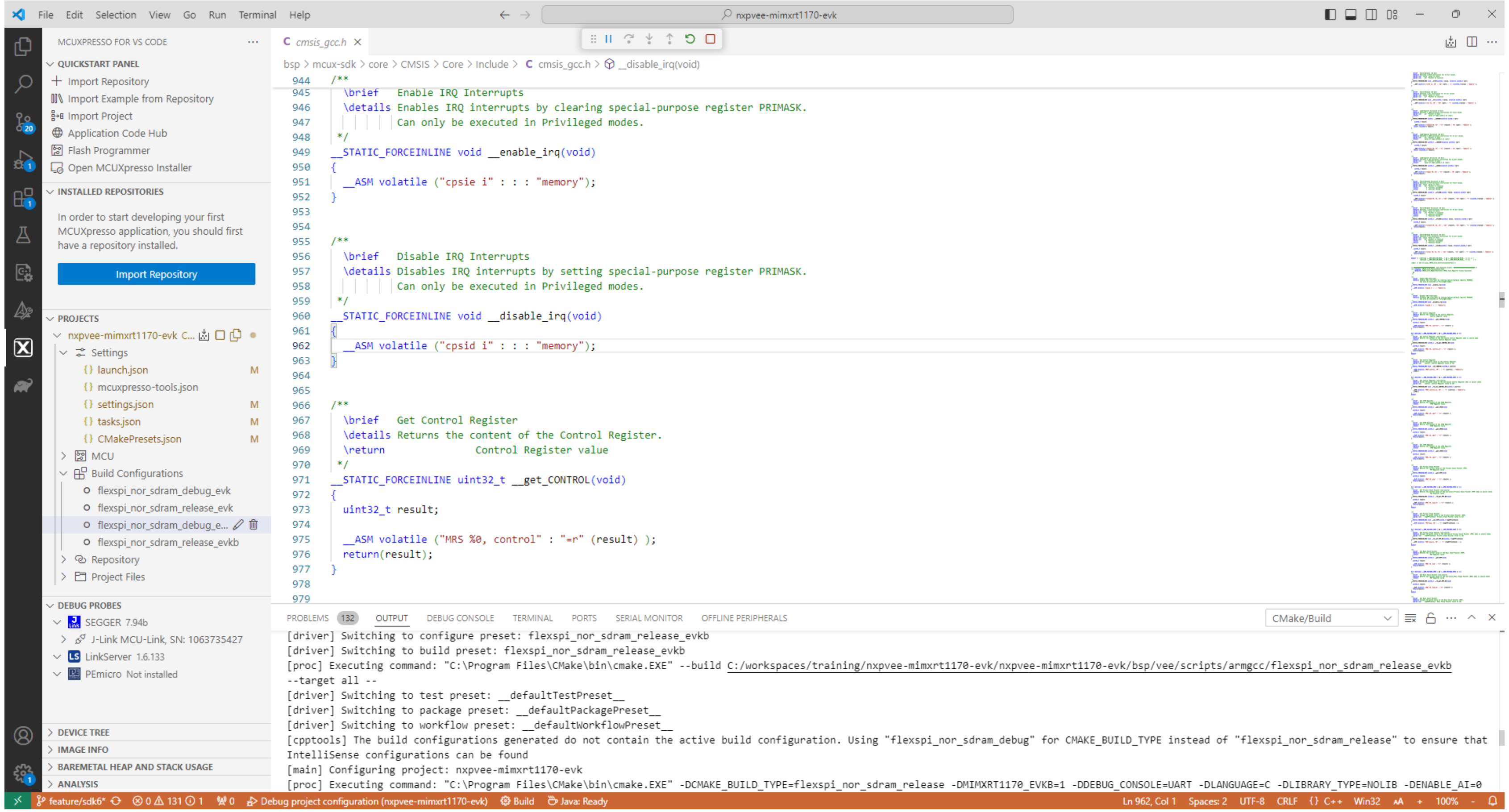




```
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/0_icetea_heap.hex 0x20000e00 0x200
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/1_java_heap.hex 0x80081634 0x800873f4
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/2_immortals_heap.hex 0x800873f4 0x80087
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/3_java_stacks.hex 0x20000000 0x20000e00
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/4_vm_instance.hex 0x800a1bfc 0x800a1c00
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/5_bt4_static.hex 0x80081380 0x80081398
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/6_bt8_static.hex 0x80081398 0x800813a8
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/7_ref_static.hex 0x800813a8 0x80081614
```

DUMP THE DEVICE MEMORY (1/3)

- Once the **vee-memory-dump.gdb** file is generated, open VS Code.
- Attach to the device in VS Code (cf. Debug the BSP C Code).

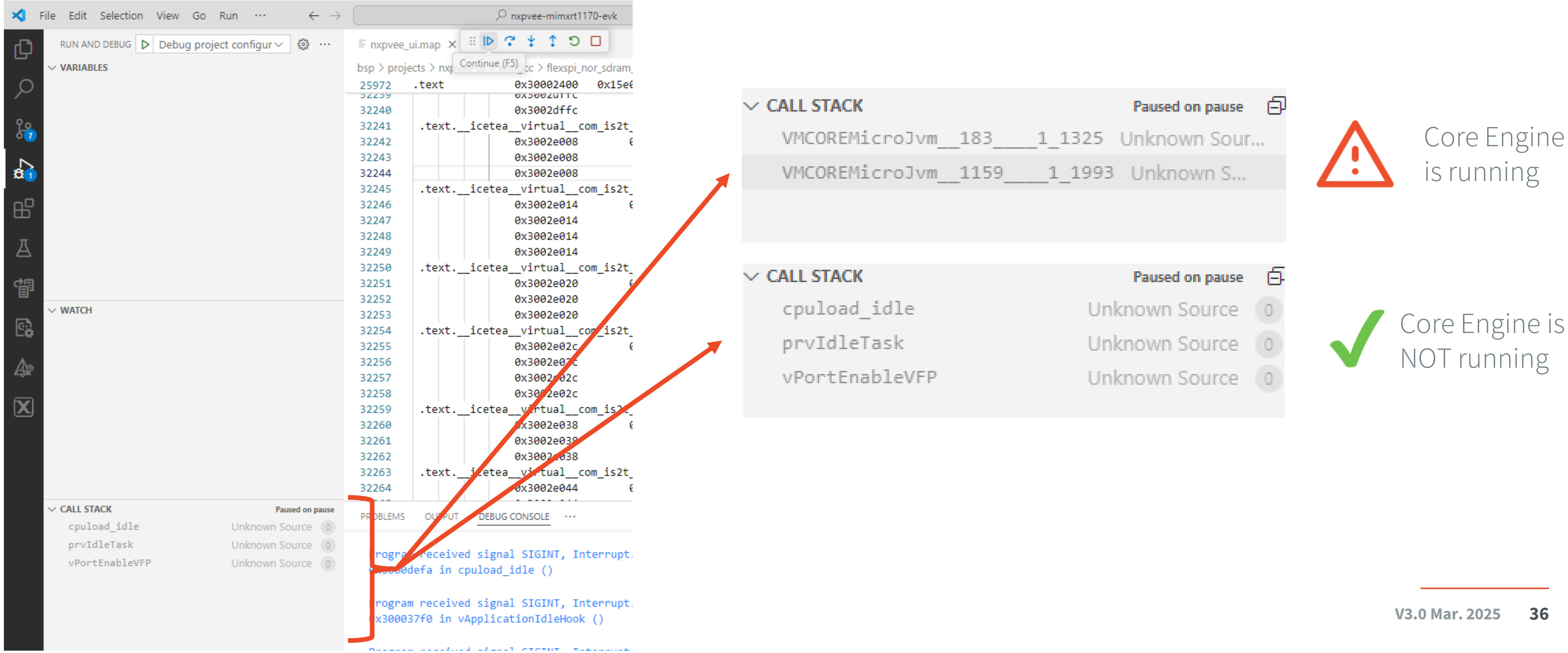

File Edit Selection View Go Run Terminal Help
nxpvee-mimxrt1170-evk
MCUXPRESSO FOR VS CODE
QUICKSTART PANEL
Import Repository
Import Example from Repository
Import Project
Application Code Hub
Flash Programmer
Open MCUXpresso Installer
INSTALLED REPOSITORIES
In order to start developing your first MCUXpresso application, you should first have a repository installed.
Import Repository
PROJECTS
nxpvee-mimxrt1170-evk C...
Settings
launch.json M
mcuxpresso-tools.json
settings.json M
tasks.json M
CMakePresets.json M
MCU
Build Configurations
flexspi_nor_sdram_debug_evk
flexspi_nor_sdram_release_evk
flexspi_nor_sdram_debug_e...
flexspi_nor_sdram_release_evkb
Repository
Project Files
DEBUG PROBES
SEGGER 7.94b
J-Link MCU-Link, SN: 1063735427
LinkServer 1.6.133
PEmicro Not installed
DEVICE TREE
IMAGE INFO
BAREMETAL HEAP AND STACK USAGE
ANALYSIS
cmsis_gcc.h
bsp > mcux-sdk > core > CMSIS > Core > Include > cmsis_gcc.h > __disable_irq(void)
PROBLEMS 132
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SERIAL MONITOR
OFFLINE PERIPHERALS
CMake/Build
feature/sdk6* 0 131 1 0 Debug project configuration (nxpvee-mimxrt1170-evk) Build Java: Ready
Ln 962, Col 1 Spaces: 2 UTF-8 CRLF {} C++ Win32 100%

```c
/**
  \brief   Enable IRQ Interrupts
  \details Enables IRQ interrupts by clearing special-purpose register PRIMASK.
           Can only be executed in Privileged modes.
 */
__STATIC_FORCEINLINE void __enable_irq(void)
{
  __ASM volatile ("cpsie i" : : : "memory");
}


/**
  \brief   Disable IRQ Interrupts
  \details Disables IRQ interrupts by setting special-purpose register PRIMASK.
           Can only be executed in Privileged modes.
 */
__STATIC_FORCEINLINE void __disable_irq(void)
{
  __ASM volatile ("cpsid i" : : : "memory");
}


/**
  \brief   Get Control Register
  \details Returns the content of the Control Register.
  \return               Control Register value
 */
__STATIC_FORCEINLINE uint32_t __get_CONTROL(void)
{
  uint32_t result;

  __ASM volatile ("MRS %0, control" : "=r" (result) );
  return(result);
}

```

```
[driver] Switching to configure preset: flexspi_nor_sdram_release_evkb
[driver] Switching to build preset: flexspi_nor_sdram_release_evkb
[proc] Executing command: "C:\Program Files\CMake\bin\cmake.EXE" --build C:/workspaces/training/nxpvee-mimxrt1170-evk/nxpvee-mimxrt1170-evk/bsp/vee/scripts/armgcc/flexspi_nor_sdram_release_evkb
--target all --
[driver] Switching to test preset: __defaultTestPreset__
[driver] Switching to package preset: __defaultPackagePreset__
[driver] Switching to workflow preset: __defaultWorkflowPreset__
[cpptools] The build configurations generated do not contain the active build configuration. Using "flexspi_nor_sdram_debug" for CMAKE_BUILD_TYPE instead of "flexspi_nor_sdram_release" to ensure that
IntelliSense configurations can be found
[main] Configuring project: nxpvee-mimxrt1170-evk
[proc] Executing command: "C:\Program Files\CMake\bin\cmake.EXE" -DCMAKE_BUILD_TYPE=flexspi_nor_sdram_release -DMIMXRT1170_EVKB=1 -DDEBUG_CONSOLE=UART -DLANGUAGE=C -DLIBRARY_TYPE=NOLIB -DENABLE_AI=0
```

DUMP THE DEVICE MEMORY (2/3)

- To make sure that the Core engine is not running when the dump is performed, it is recommended to create a breakpoint at a specific safe point (Core Engine hooks or native function).
- Otherwise, make sure that the Core engine is not running when **pausing** the debugger (see Call Stack section in VS Code):



```
.text                   0x30002400   0x15e
                        0x3002dffc
 .text.__icetea__virtual__com_is2t_
                        0x3002e008
                        0x3002e008
                        0x3002e008
 .text.__icetea__virtual__com_is2t_
                        0x3002e014
                        0x3002e014
                        0x3002e014
                        0x3002e014
 .text.__icetea__virtual__com_is2t_
                        0x3002e020
                        0x3002e020
                        0x3002e020
 .text.__icetea__virtual__com_is2t_
                        0x3002e02c
                        0x3002e02c
                        0x3002e02c
                        0x3002e02c
 .text.__icetea__virtual__com_is2t_
                        0x3002e038
                        0x3002e038
                        0x3002e038
 .text.__icetea__virtual__com_is2t_
                        0x3002e044
```

```
Program received signal SIGINT, Interrupt.
0x30000defa in cpuload_idle ()

Program received signal SIGINT, Interrupt.
0x300037f0 in vApplicationIdleHook ()
```

CALL STACK — Paused on pause

VMCOREMicroJvm__183____1_1325	Unknown Sour...
VMCOREMicroJvm__1159____1_1993	Unknown S...

Core Engine is running

CALL STACK — Paused on pause

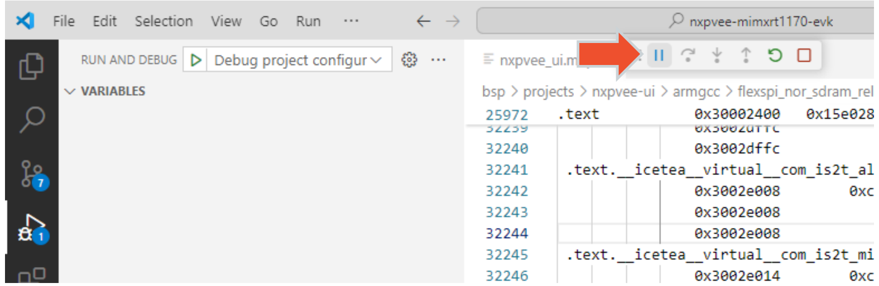
cpuload_idle	Unknown Source	0
prvIdleTask	Unknown Source	0
vPortEnableVFP	Unknown Source	0

✓ Core Engine is NOT running

DUMP THE DEVICE MEMORY (3/3)

In VS Code, run the **vee-memory-dump.gdb** script file to dump the memory:

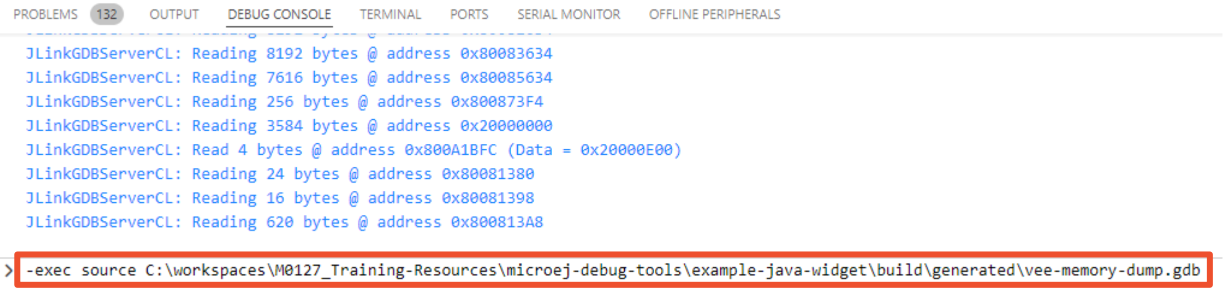
1. Pause the debugger:



```
.text                0x30002400    0x15e028
                     0x3002dffc
 .text.__icetea__virtual__com_is2t_al
                     0x3002e008        0xc
                     0x3002e008
                     0x3002e008
 .text.__icetea__virtual__com_is2t_mi
                     0x3002e014        0xc
```

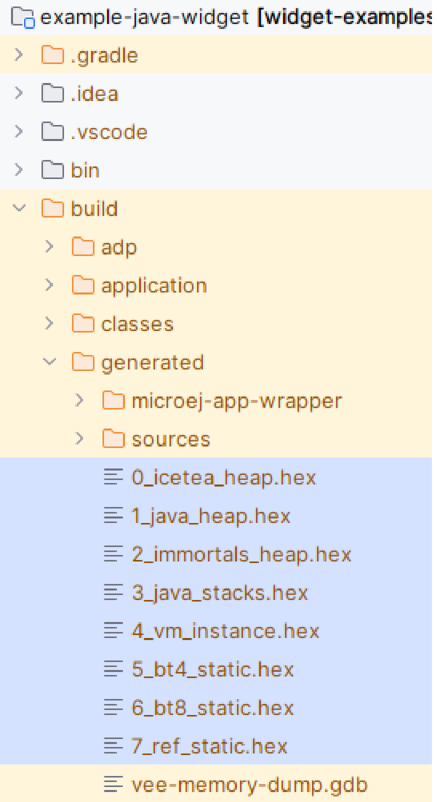
2. Run the following command in the **Debug Console** view:

```
-exec source C:\[YOUR_PATH]\vee-memory-dump.gdb
```



```
JLinkGDBServerCL: Reading 8192 bytes @ address 0x80083634
JLinkGDBServerCL: Reading 7616 bytes @ address 0x80085634
JLinkGDBServerCL: Reading 256 bytes @ address 0x800873F4
JLinkGDBServerCL: Reading 3584 bytes @ address 0x20000000
JLinkGDBServerCL: Read 4 bytes @ address 0x800A1BFC (Data = 0x20000E00)
JLinkGDBServerCL: Reading 24 bytes @ address 0x80081380
JLinkGDBServerCL: Reading 16 bytes @ address 0x80081398
JLinkGDBServerCL: Reading 620 bytes @ address 0x800813A8

> -exec source C:\workspaces\M0127_Training-Resources\microej-debug-tools\example-java-widget\build\generated\vee-memory-dump.gdb
```

3. Heap dumps are generated in the output folder:

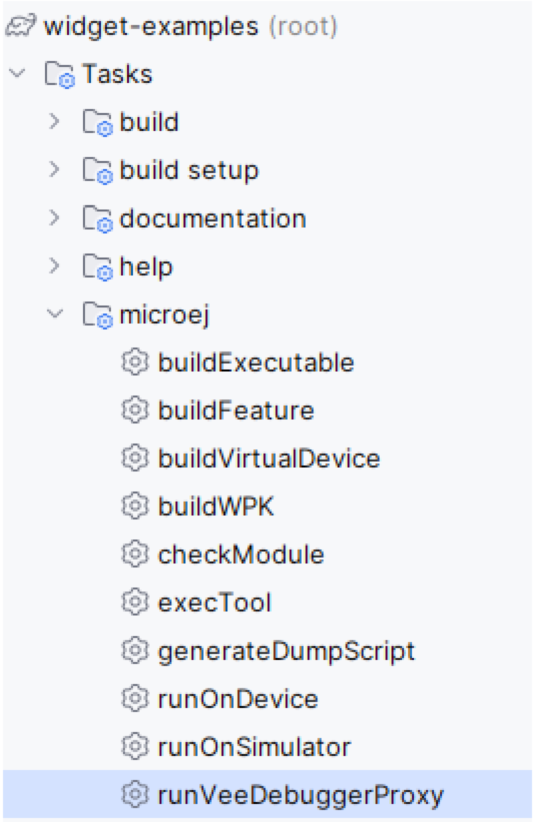


```
example-java-widget [widget-examples
  > .gradle
  > .idea
  > .vscode
  > bin
  v build
      > adp
      > application
      > classes
      v generated
          > microej-app-wrapper
          > sources
            0_icetea_heap.hex
            1_java_heap.hex
            2_immortals_heap.hex
            3_java_stacks.hex
            4_vm_instance.hex
            5_bt4_static.hex
            6_bt8_static.hex
            7_ref_static.hex
            vee-memory-dump.gdb
```

Note: the output folder is specified when generating the **vee-memory-dump.gdb** script.

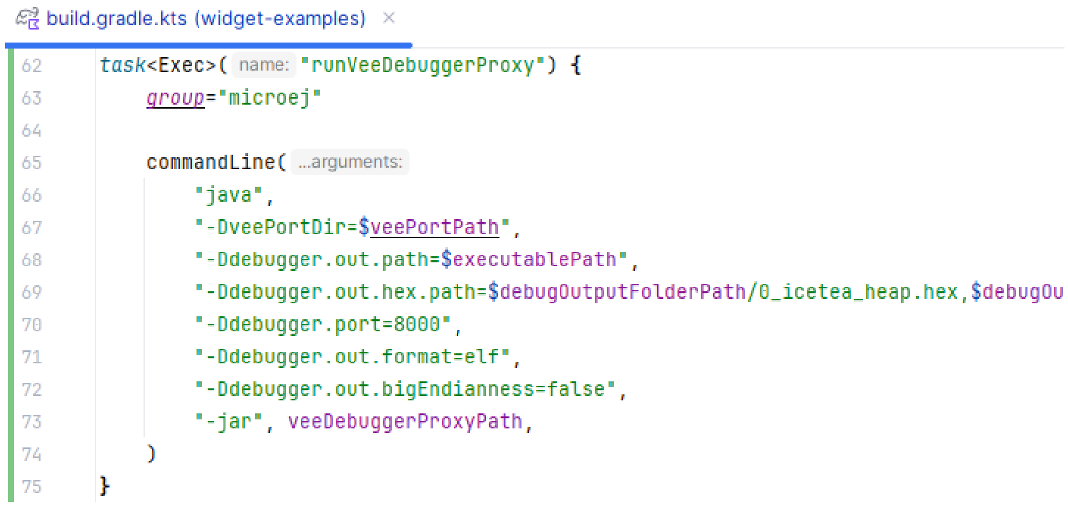
RUN THE VEE DEBUGGER PROXY (1/2)

The **example-java-widget** project provides the **generateDumpScript** task that allows the user to run the VEE debugger proxy:



- widget-examples (root)
  - Tasks
    - build
    - build setup
    - documentation
    - help
    - microej
      - buildExecutable
      - buildFeature
      - buildVirtualDevice
      - buildWPK
      - checkModule
      - execTool
      - generateDumpScript
      - runOnDevice
      - runOnSimulator
      - runVeeDebuggerProxy

This task is declared in the **build.gradle.kts**, it is based on the command line provided in the VEE Debugger Proxy documentation:



build.gradle.kts (widget-examples)

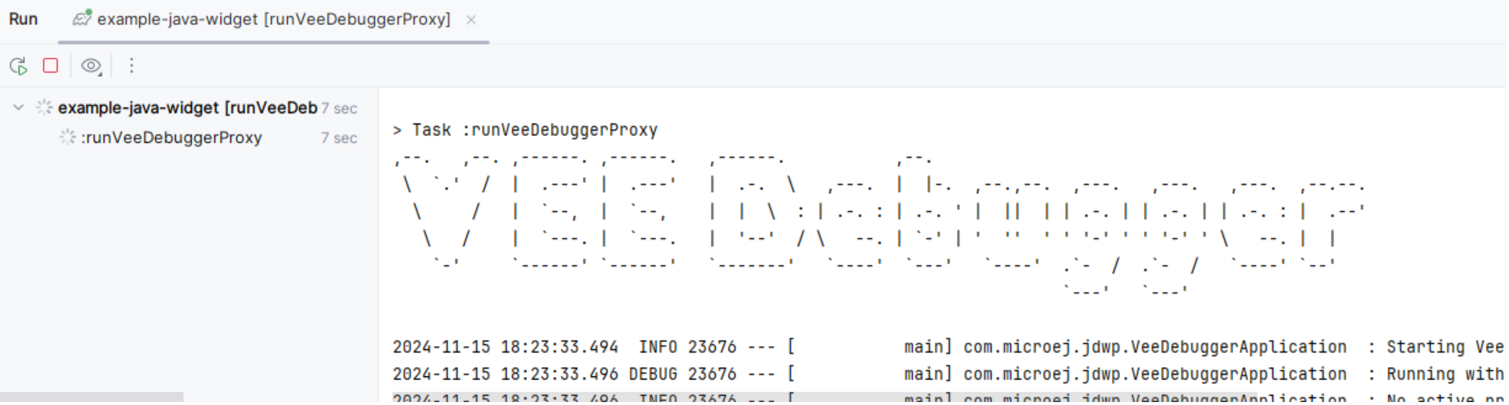
```
task<Exec>( name: "runVeeDebuggerProxy") {
    group="microej"

    commandLine( ...arguments:
        "java",
        "-DveePortDir=$veePortPath",
        "-Ddebugger.out.path=$executablePath",
        "-Ddebugger.out.hex.path=$debugOutputFolderPath/0_icetea_heap.hex,$debugOu
        "-Ddebugger.port=8000",
        "-Ddebugger.out.format=elf",
        "-Ddebugger.out.bigEndianness=false",
        "-jar", veeDebuggerProxyPath,
    )
}
```

The tool takes the dumped .hex files as input.

RUN THE VEE DEBUGGER PROXY (2/2)

Run the **runVeeDebuggerProxy** task:

- The tool is launched in the console:

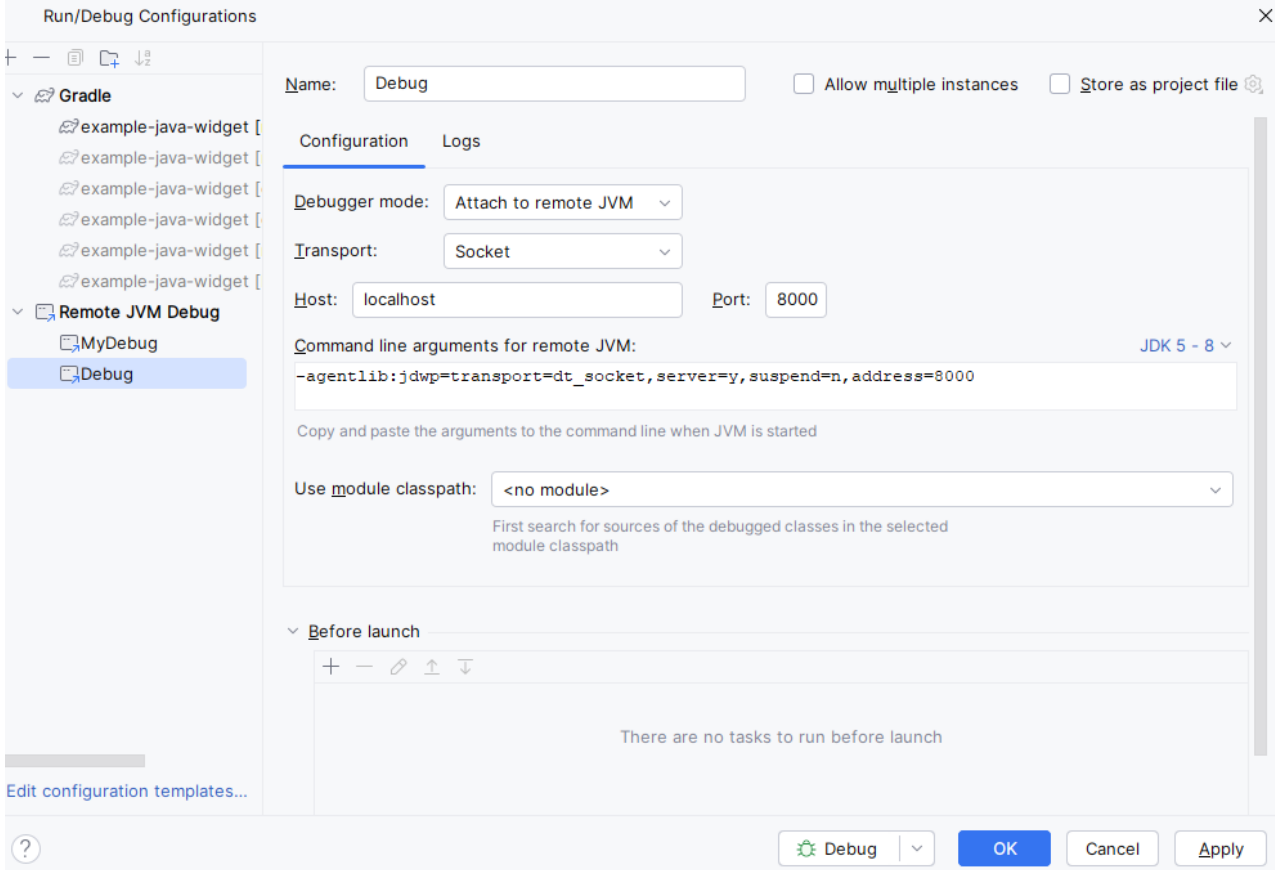

Run
example-java-widget [runVeeDebuggerProxy]
example-java-widget [runVeeDeb 7 sec
:runVeeDebuggerProxy 7 sec

```
> Task :runVeeDebuggerProxy
,--.   ,--. ,------. ,------.   ,------.           ,--.
 \  `.'  /  |  .---' |  .---'   |  .-.  \   ,---.  |  |-.  ,--.,--.  ,---.   ,---.   ,---.  ,--.--.
  \     /   |  `--,  |  `--,    |  |  \  : | .-. : | .-. ' |  ||  | | .-. | | .-. | | .-. : |  .--'
   \   /    |  `---. |  `---.   |  '--'  / \   --. | `-' | '  ''  ' ' '-' ' ' '-' ' \   --. |  |
    `-'     `------' `------'   `-------'   `----'  `---'   `----'  .`-  /  .`-  /   `----' `--'
                                                                    `---'   `---'

2024-11-15 18:23:33.494  INFO 23676 --- [           main] com.microej.jdwp.VeeDebuggerApplication  : Starting Vee
2024-11-15 18:23:33.496 DEBUG 23676 --- [           main] com.microej.jdwp.VeeDebuggerApplication  : Running with
2024-11-15 18:23:33.496  INFO 23676 --- [           main] com.microej.jdwp.VeeDebuggerApplication  : No active pr
```

RUN A REMOTE JAVA APPLICATION DEBUG SESSION

In IntelliJ IDEA:

- Click on **Run > Edit Configurations….**
- Click on **+** button (Add New Configuration).
- Select **Remote JVM Debug**.
- Click on the **New launch configuration** button.
- Give a name to the launcher in the **Name** field.
- Set the debug **host** to **localhost** and **port** to **8000**.
- Click on the **Debug** button.



Name: Debug

□ Allow multiple instances

□ Store as project file

Debugger mode: Attach to remote JVM

Transport: Socket

Host: localhost

Port: 8000

Command line arguments for remote JVM:

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000
```

Copy and paste the arguments to the command line when JVM is started

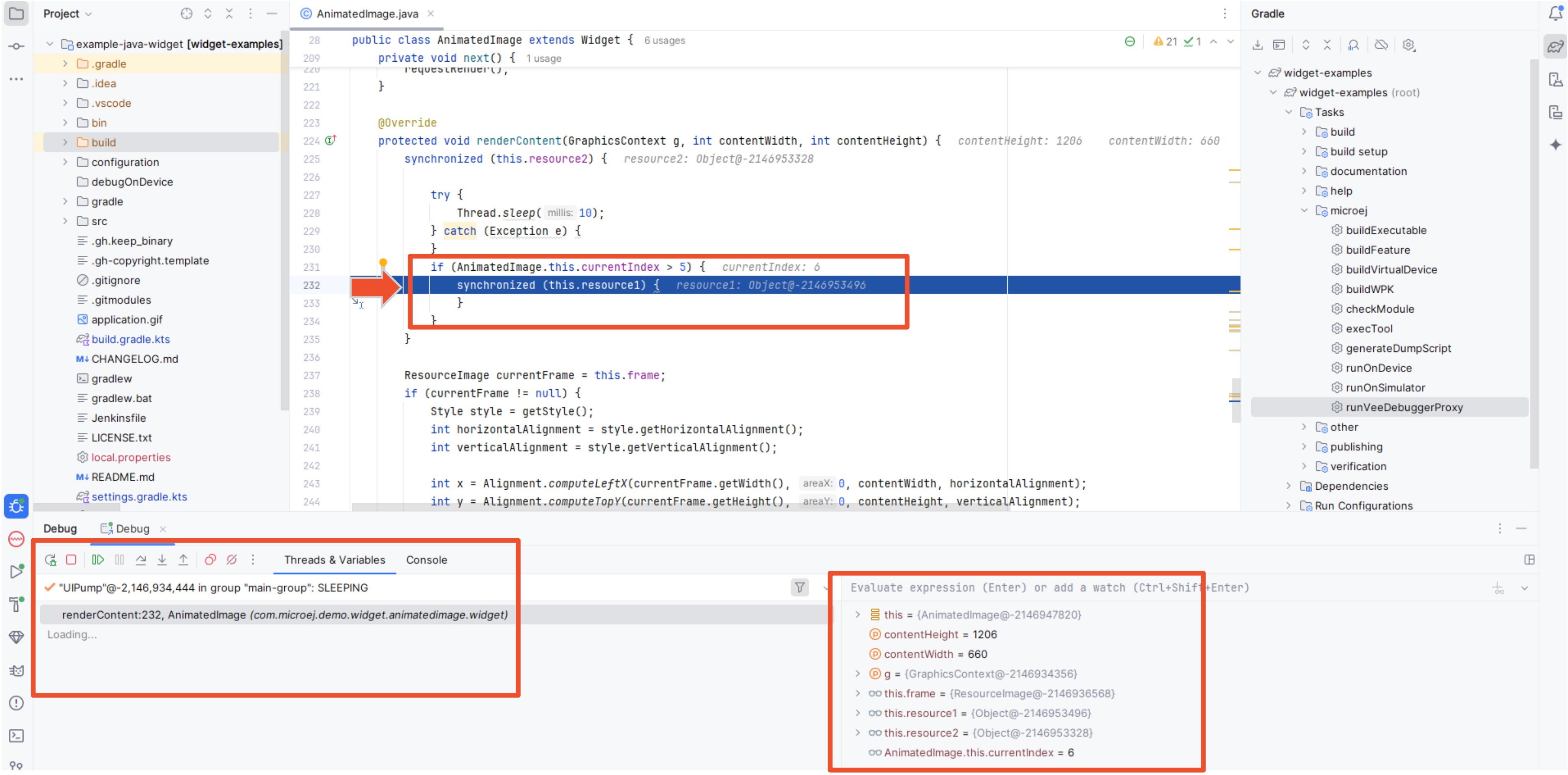
Use module classpath: <no module>

First search for sources of the debugged classes in the selected module classpath

There are no tasks to run before launch

GET THE POST-MORTEM DEBUGGING STATE (1/2)

The following debug state can be seen when pausing the debugger **and the threads**:



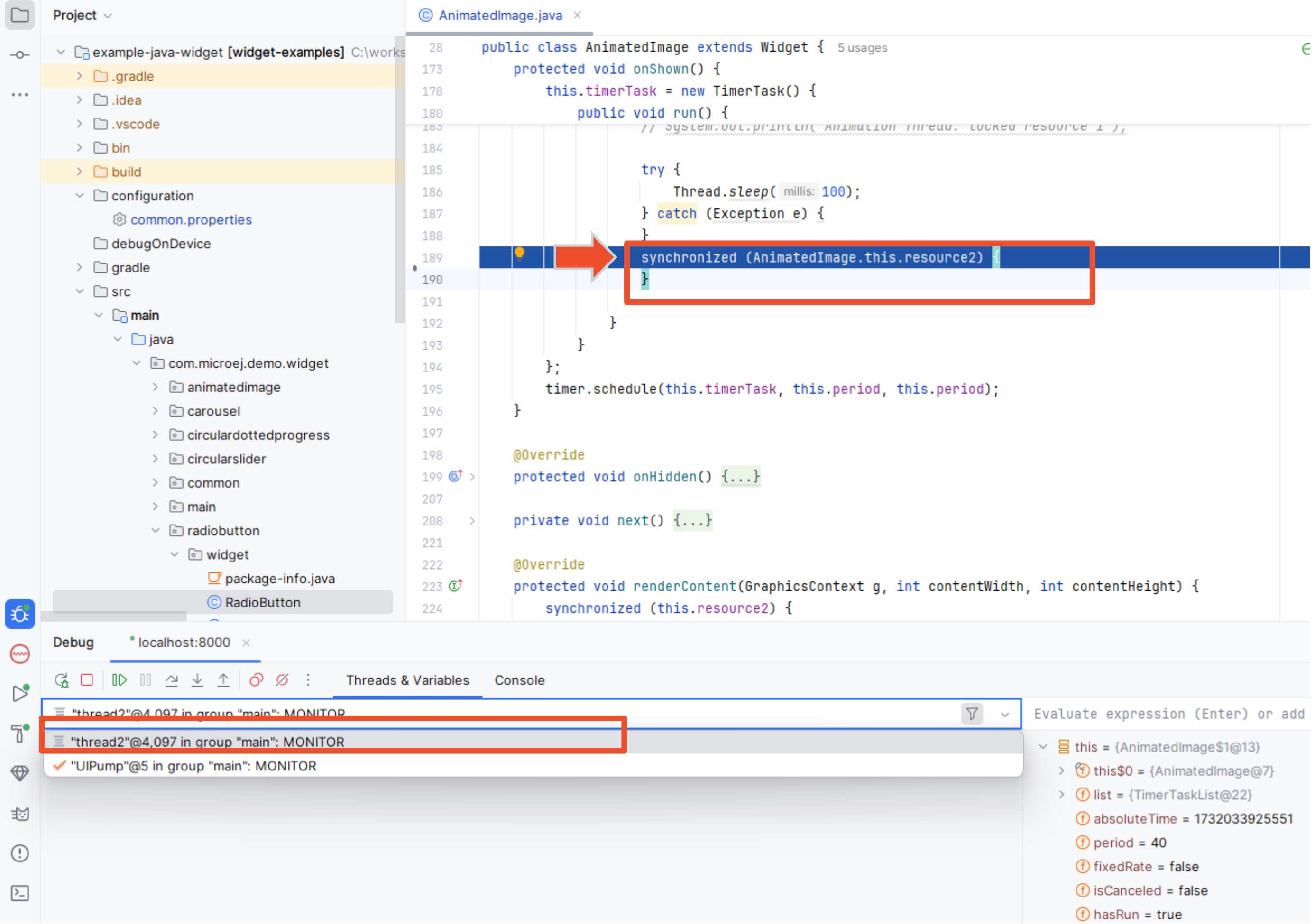
```
public class AnimatedImage extends Widget {  6 usages
    private void next() {  1 usage
        requestRender();
    }

    @Override
    protected void renderContent(GraphicsContext g, int contentWidth, int contentHeight) {   contentHeight: 1206   contentWidth: 660
        synchronized (this.resource2) {   resource2: Object@-2146953328

            try {
                Thread.sleep( millis: 10);
            } catch (Exception e) {
            }
            if (AnimatedImage.this.currentIndex > 5) {   currentIndex: 6
                synchronized (this.resource1) {   resource1: Object@-2146953496
                }
            }
        }

        ResourceImage currentFrame = this.frame;
        if (currentFrame != null) {
            Style style = getStyle();
            int horizontalAlignment = style.getHorizontalAlignment();
            int verticalAlignment = style.getVerticalAlignment();

            int x = Alignment.computeLeftX(currentFrame.getWidth(),  areaX: 0, contentWidth, horizontalAlignment);
            int y = Alignment.computeTopY(currentFrame.getHeight(),  areaY: 0, contentHeight, verticalAlignment);
```

```
"UIPump"@-2,146,934,444 in group "main-group": SLEEPING
    renderContent:232, AnimatedImage (com.microej.demo.widget.animatedimage.widget)
Loading...
```

```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
> this = {AnimatedImage@-2146947820}
  contentHeight = 1206
  contentWidth = 660
> g = {GraphicsContext@-2146934356}
> this.frame = {ResourceImage@-2146936568}
> this.resource1 = {Object@-2146953496}
> this.resource2 = {Object@-2146953328}
  AnimatedImage.this.currentIndex = 6
```

GET THE POST-MORTEM DEBUGGING STATE (2/2)

The 2nd Thread state can also be seen:



```java
public class AnimatedImage extends Widget {
    protected void onShown() {
        this.timerTask = new TimerTask() {
            public void run() {
                    // System.out.println("Animation Thread: locked resource 1");

                    try {
                        Thread.sleep( millis: 100);
                    } catch (Exception e) {
                    }
                    synchronized (AnimatedImage.this.resource2) {
                    }

                }
            }
        };
        timer.schedule(this.timerTask, this.period, this.period);
    }

    @Override
    protected void onHidden() {...}

    private void next() {...}

    @Override
    protected void renderContent(GraphicsContext g, int contentWidth, int contentHeight) {
        synchronized (this.resource2) {
```

```
"thread2"@4,097 in group "main": MONITOR
"thread2"@4,097 in group "main": MONITOR
✓ "UIPump"@5 in group "main": MONITOR
```

```
Evaluate expression (Enter) or add
this = {AnimatedImage$1@13}
  this$0 = {AnimatedImage@7}
  list = {TimerTaskList@22}
  absoluteTime = 1732033925551
  period = 40
  fixedRate = false
  isCanceled = false
  hasRun = true
```

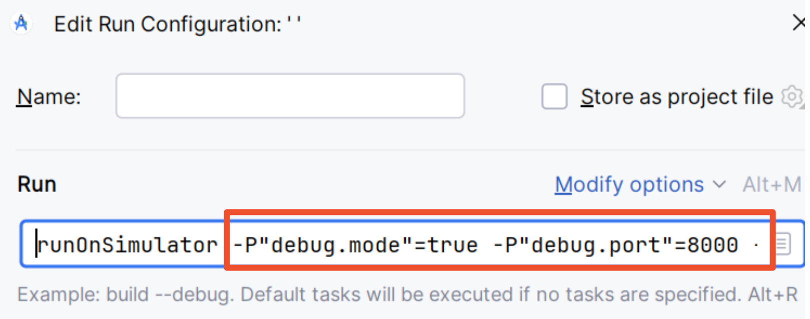
Debug on Simulator

Debug the Application Code
on the Simulator

DEBUG ON THE SIMULATOR (1/2)

- Execute the **runOnSimulator** Gradle task with the following options:

```
-P"debug.mode"=true -P"debug.port"=8000
```



Edit Run Configuration: ''

Name:

□ Store as project file

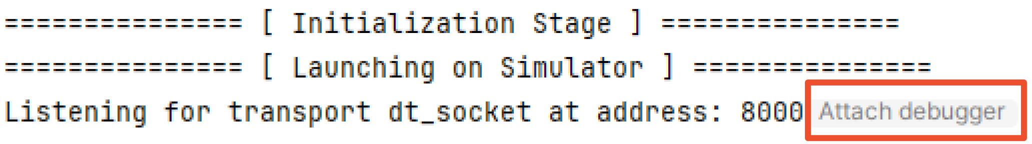
Run

Modify options Alt+M

```
runOnSimulator -P"debug.mode"=true -P"debug.port"=8000
```

Example: build --debug. Default tasks will be executed if no tasks are specified. Alt+R

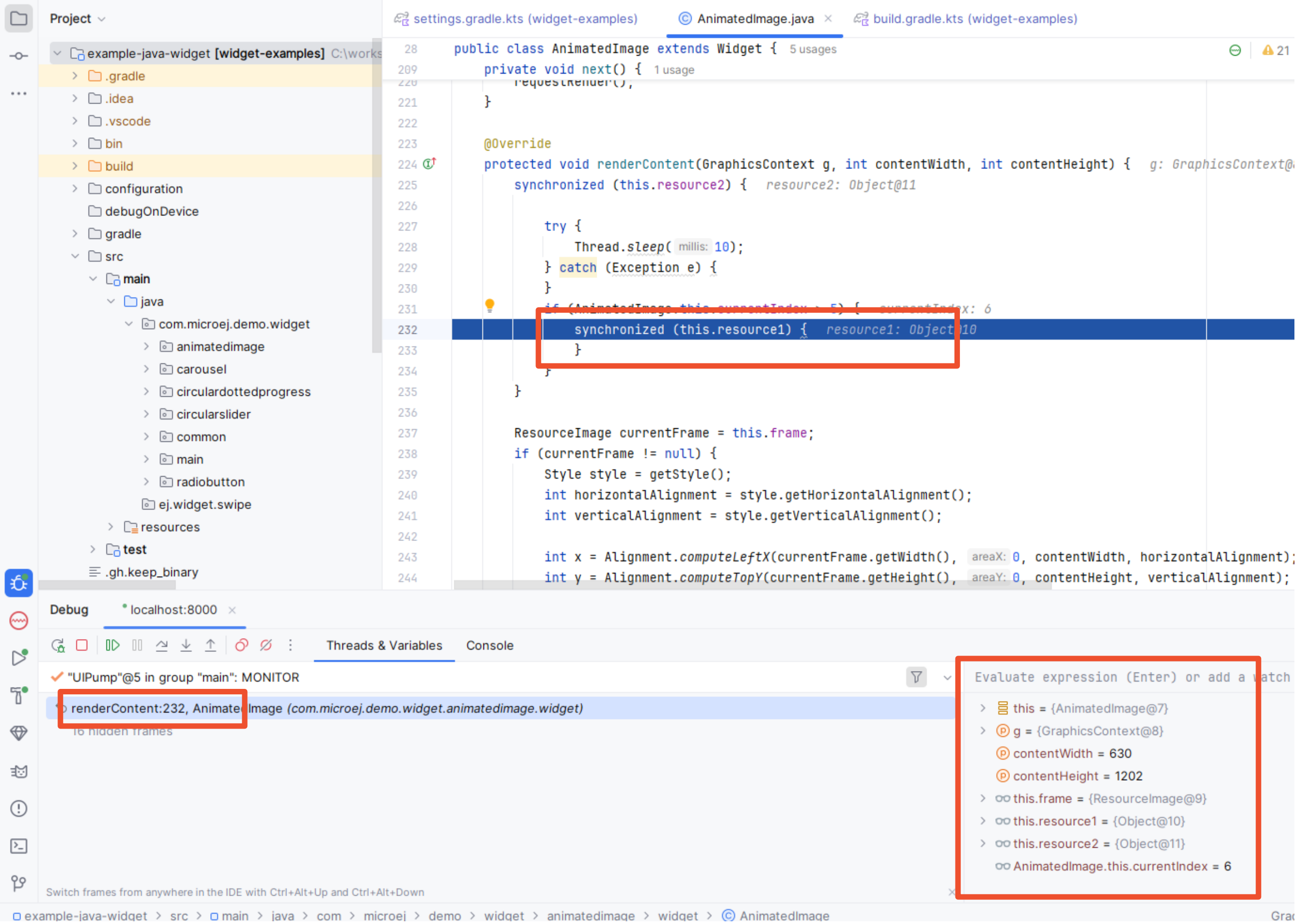
- The console opens with the following message, click on Attach debugger to start the debug session:



```
=============== [ Initialization Stage ] ===============
=============== [ Launching on Simulator ] ===============
Listening for transport dt_socket at address: 8000 Attach debugger
```

DEBUG ON THE SIMULATOR (2/2)

- Pause the Debugger once the freeze occurs.
- The same state that the debug on device view can be seen



```
public class AnimatedImage extends Widget {
    private void next() {
            requestRender();
    }

    @Override
    protected void renderContent(GraphicsContext g, int contentWidth, int contentHeight) {   g: GraphicsContext@
        synchronized (this.resource2) {   resource2: Object@11

            try {
                Thread.sleep( millis: 10);
            } catch (Exception e) {
            }
            if (AnimatedImage.this.currentIndex > 5) {   currentIndex: 6
                synchronized (this.resource1) {   resource1: Object@10
                }
            }
        }

        ResourceImage currentFrame = this.frame;
        if (currentFrame != null) {
            Style style = getStyle();
            int horizontalAlignment = style.getHorizontalAlignment();
            int verticalAlignment = style.getVerticalAlignment();

            int x = Alignment.computeLeftX(currentFrame.getWidth(),  areaX: 0, contentWidth, horizontalAlignment);
            int y = Alignment.computeTopY(currentFrame.getHeight(),  areaY: 0, contentHeight, verticalAlignment);
```

"UIPump"@5 in group "main": MONITOR

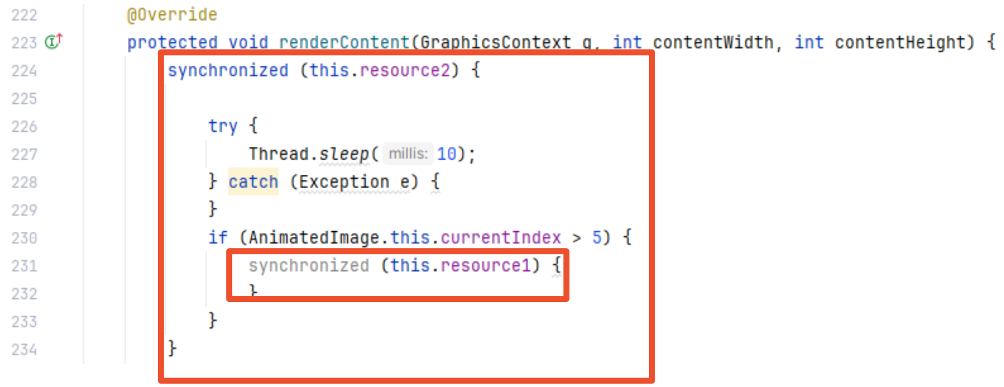
renderContent:232, AnimatedImage *(com.microej.demo.widget.animatedimage.widget)*

```
Evaluate expression (Enter) or add a watch
this = {AnimatedImage@7}
g = {GraphicsContext@8}
contentWidth = 630
contentHeight = 1202
this.frame = {ResourceImage@9}
this.resource1 = {Object@10}
this.resource2 = {Object@11}
AnimatedImage.this.currentIndex = 6
```

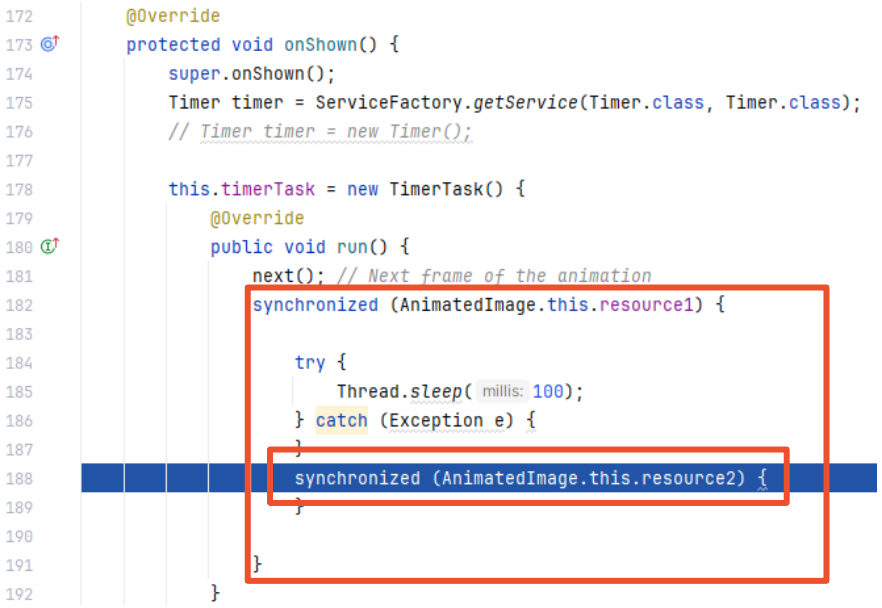
ISSUE ANALYSIS & FIX

The interlock is caused by the synchronization of **resource1** and **resource2** objects on 2 different Threads:

- The **renderContent** method is called in the **UIPump** thread context
    - Note: this method is called at every animation frame of the animated image
- The **onShown** method creates a TimerTask that is executed in the context of the **Timer** thread
    - Note: this method is called once, when the Animated Page is shown



```
@Override
protected void renderContent(GraphicsContext g, int contentWidth, int contentHeight) {
    synchronized (this.resource2) {

        try {
            Thread.sleep( millis: 10);
        } catch (Exception e) {
        }
        if (AnimatedImage.this.currentIndex > 5) {
            synchronized (this.resource1) {
            }
        }
    }
```



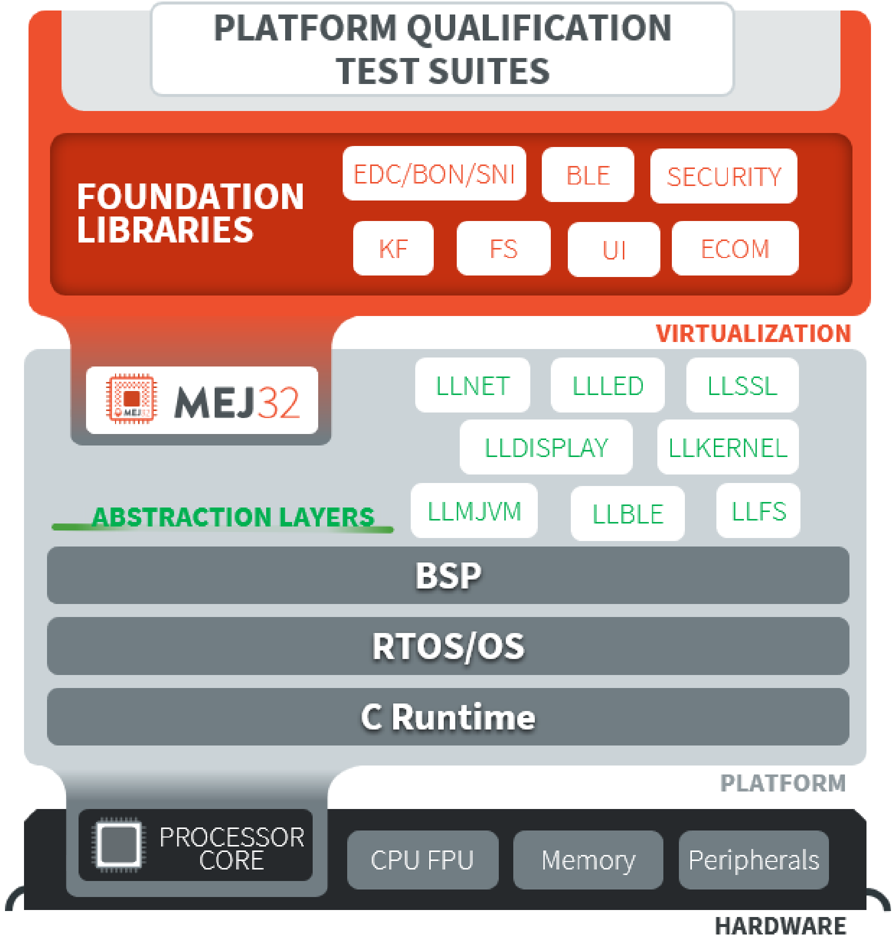
```
@Override
protected void onShown() {
    super.onShown();
    Timer timer = ServiceFactory.getService(Timer.class, Timer.class);
    // Timer timer = new Timer();

    this.timerTask = new TimerTask() {
        @Override
        public void run() {
            next(); // Next frame of the animation
            synchronized (AnimatedImage.this.resource1) {

                try {
                    Thread.sleep( millis: 100);
                } catch (Exception e) {
                }
                synchronized (AnimatedImage.this.resource2) {
                }

            }
        }
```

Fix: this code has been written for training purpose, remove it to unlock the application.

Port Qualification Tool

PORT QUALIFICATION TOOL (1/2)

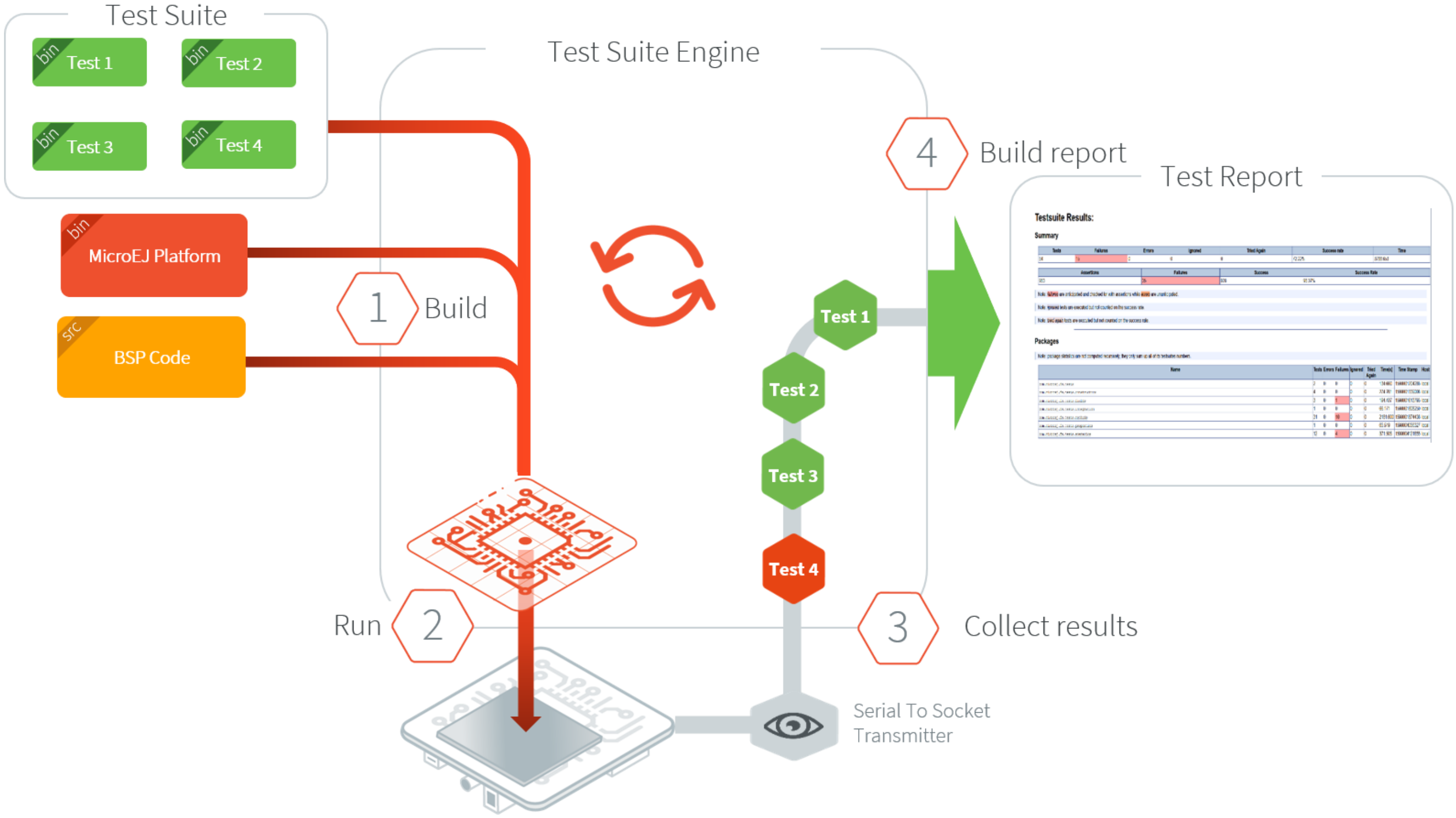
- The Port Qualification Tool (PQT) project provides the tools required to validate each component of a MicroEJ VEE Port.
- After porting or adding a Foundation Library to a MicroEJ VEE Port, it is necessary to validate its integration.
- For each Low Level API, an Abstraction Layer implementation is 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 Testsuite 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.)

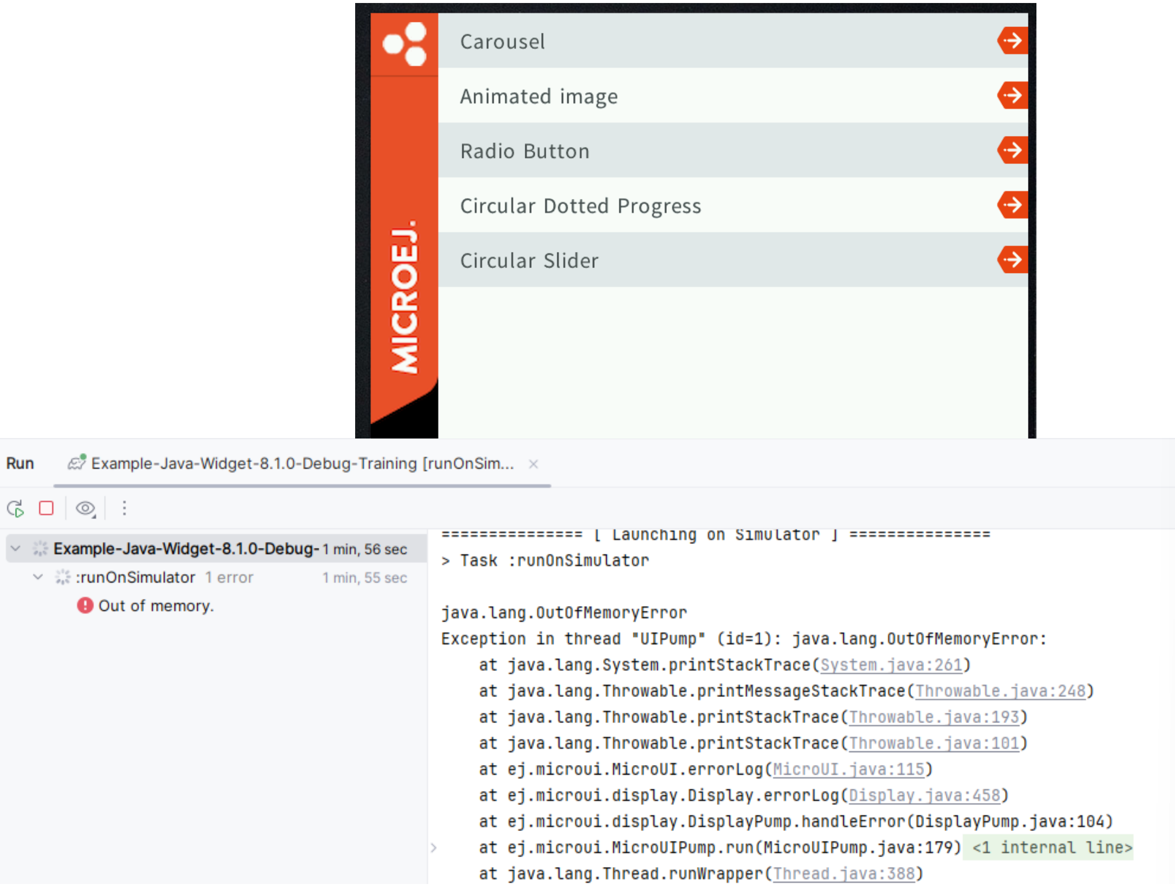
4. Debugger (on device & simulator): analysis of an applicative issue

Memory Inspection Tools

MEMORY INSPECTION TOOLS

- Tools:
  - ○ Memory Map Analyzer*
  - ○ 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


Carousel
Animated image
Radio Button
Circular Dotted Progress
Circular Slider
MICROEJ.
Run
Example-Java-Widget-8.1.0-Debug-Training [runOnSim...
Example-Java-Widget-8.1.0-Debug- 1 min, 56 sec
:runOnSimulator 1 error 1 min, 55 sec

Out of memory.

```
=============== [ Launching on Simulator ] ===============
> Task :runOnSimulator

java.lang.OutOfMemoryError
Exception in thread "UIPump" (id=1): java.lang.OutOfMemoryError:
    at java.lang.System.printStackTrace(System.java:261)
    at java.lang.Throwable.printMessageStackTrace(Throwable.java:248)
    at java.lang.Throwable.printStackTrace(Throwable.java:193)
    at java.lang.Throwable.printStackTrace(Throwable.java:101)
    at ej.microui.MicroUI.errorLog(MicroUI.java:115)
    at ej.microui.display.Display.errorLog(Display.java:458)
    at ej.microui.display.DisplayPump.handleError(DisplayPump.java:104)
    at ej.microui.MicroUIPump.run(MicroUIPump.java:179) <1 internal line>
    at java.lang.Thread.runWrapper(Thread.java:388)
```

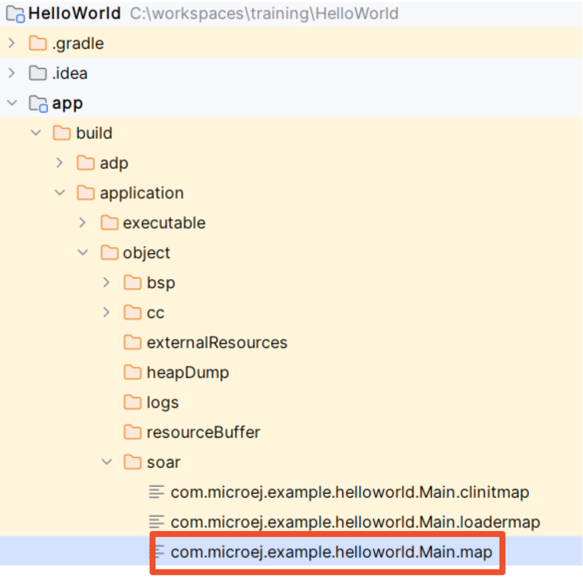
Out Of Memory exception in a GUI application

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

MEMORY MAP ANALYZER

PRINCIPLE

When the Executable of an Application is built, a Memory Map file is generated:



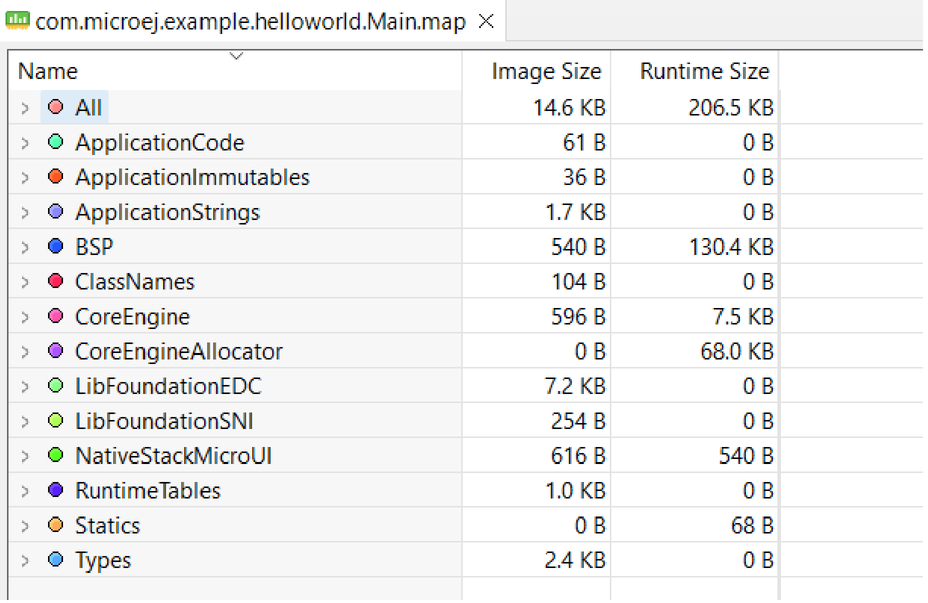
```
HelloWorld  C:\workspaces\training\HelloWorld
> .gradle
> .idea
v app
  v build
    > adp
    v application
      > executable
      v object
        > bsp
        > cc
          externalResources
          heapDump
          logs
          resourceBuffer
        v soar
            com.microej.example.helloworld.Main.clinitmap
            com.microej.example.helloworld.Main.loadermap
            com.microej.example.helloworld.Main.map
```

This file can be visualized with the Memory Map Analyzer, an Eclipse IDE plugin. It displays the memory consumption of different features in the RAM and ROM.

USAGE

.map files can be opened using the Memory Map Analyzer plugin. Make sure the Eclipse IDE is installed with the required plugin, then launch it.

- In Eclipse IDE, click on **File > Open File…** to open the .map file:



com.microej.example.helloworld.Main.map

Name	Image Size	Runtime Size
All	14.6 KB	206.5 KB
ApplicationCode	61 B	0 B
ApplicationImmutables	36 B	0 B
ApplicationStrings	1.7 KB	0 B
BSP	540 B	130.4 KB
ClassNames	104 B	0 B
CoreEngine	596 B	7.5 KB
CoreEngineAllocator	0 B	68.0 KB
LibFoundationEDC	7.2 KB	0 B
LibFoundationSNI	254 B	0 B
NativeStackMicroUI	616 B	540 B
RuntimeTables	1.0 KB	0 B
Statics	0 B	68 B
Types	2.4 KB	0 B

Note: it does not include the memory usage of the BSP project.

HEAP DUMPER & HEAP ANALYZER (1/8)

- 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 (2/8)

USAGE

Heap Dumper is a tool that allows to get a snapshot of the heap of an Application running on the Simulator or on a device.

To run the Heap Dumper on Simulator (IntelliJ IDEA / Android Studio):

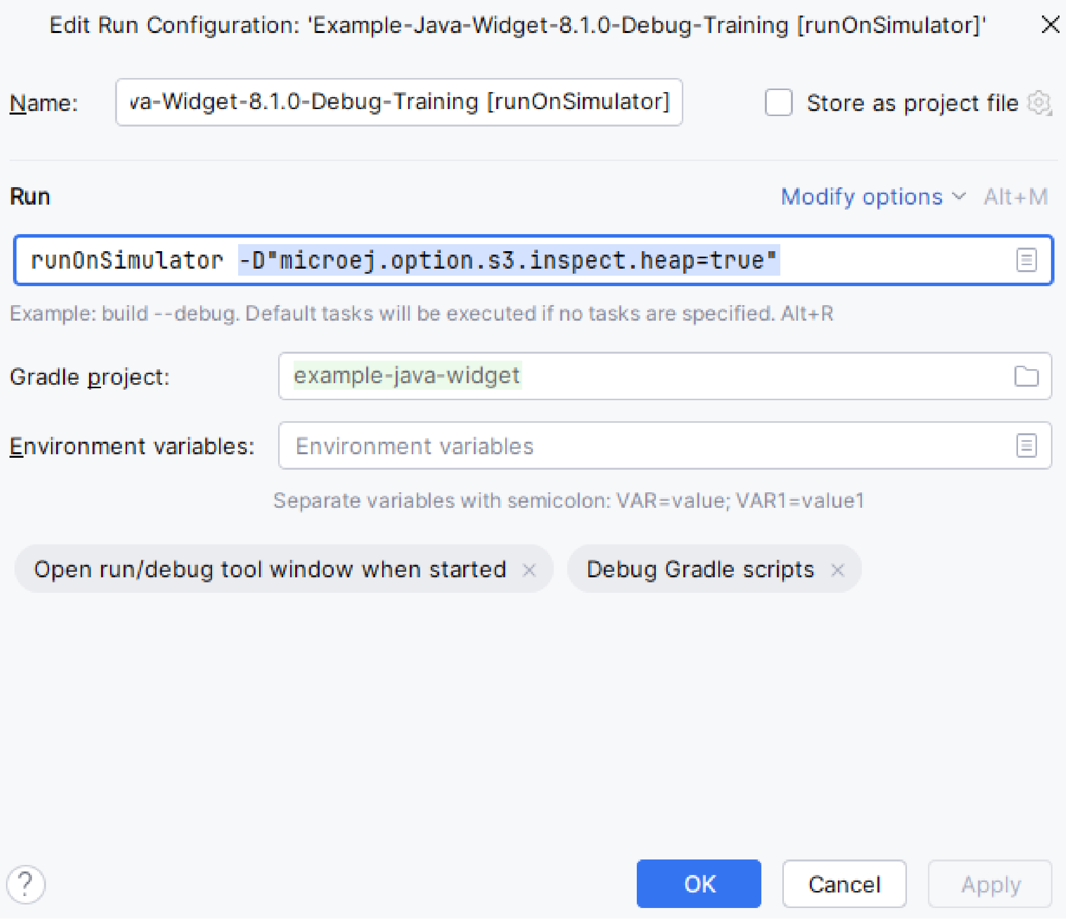
- Right-Click on the **runOnSimulator** task.
- Click on **Modify Run Configuration...**
- Add the following option:

```
-D"microej.option.s3.inspect.heap=true"
```

- Click on **Run.**

Or use this command line:

```
.\gradlew.bat runOnSimulator
-D"microej.option.s3.inspect.heap=true"
```



Edit Run Configuration: 'Example-Java-Widget-8.1.0-Debug-Training [runOnSimulator]'

Name: va-Widget-8.1.0-Debug-Training [runOnSimulator]

□ Store as project file

Run

```
runOnSimulator -D"microej.option.s3.inspect.heap=true"
```

Example: build --debug. Default tasks will be executed if no tasks are specified. Alt+R

Gradle project: example-java-widget

Environment variables: Environment variables

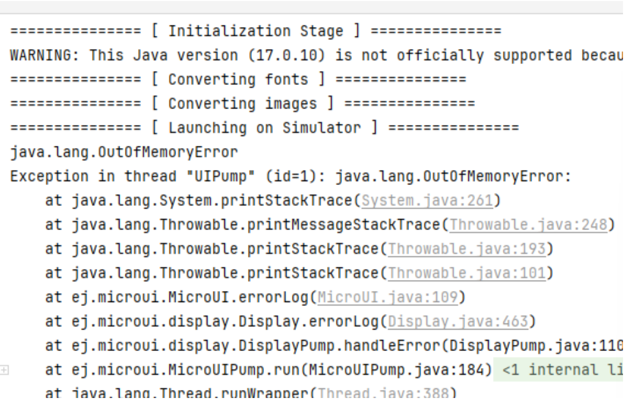
Separate variables with semicolon: VAR=value; VAR1=value1

Run Configuration options

HEAP DUMPER & HEAP ANALYZER (3/8)

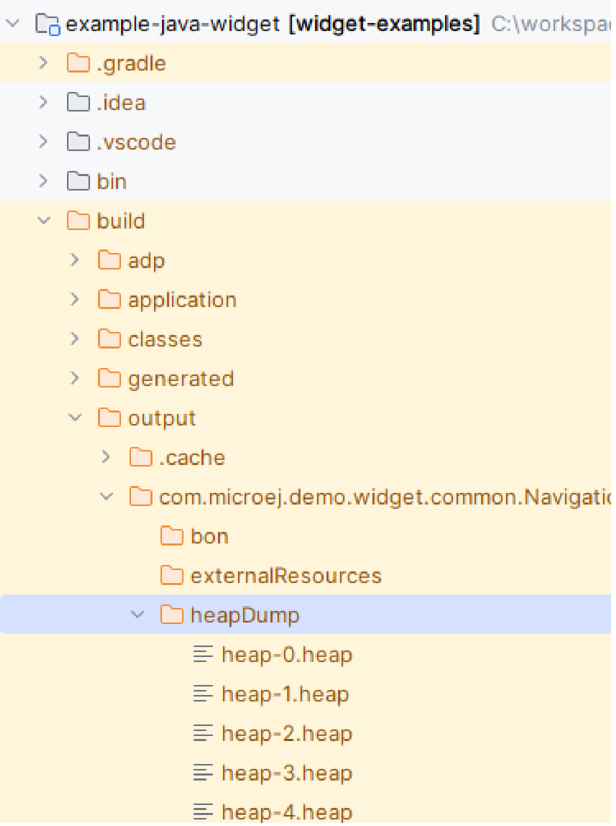
REPRODUCE THE ISSUE

- Once the Simulator is started, enter / leave the **Circular Dotted Progress page** ~10 times
- Get the error trace in the console:



```
=============== [ Initialization Stage ] ===============
WARNING: This Java version (17.0.10) is not officially supported becau
=============== [ Converting fonts ] ===============
=============== [ Converting images ] ===============
=============== [ Launching on Simulator ] ===============
java.lang.OutOfMemoryError
Exception in thread "UIPump" (id=1): java.lang.OutOfMemoryError:
    at java.lang.System.printStackTrace(System.java:261)
    at java.lang.Throwable.printMessageStackTrace(Throwable.java:248)
    at java.lang.Throwable.printStackTrace(Throwable.java:193)
    at java.lang.Throwable.printStackTrace(Throwable.java:101)
    at ej.microui.MicroUI.errorLog(MicroUI.java:109)
    at ej.microui.display.Display.errorLog(Display.java:463)
    at ej.microui.display.DisplayPump.handleError(DisplayPump.java:110
    at ej.microui.MicroUIPump.run(MicroUIPump.java:184) <1 internal li
    at java.lang.Thread.runWrapper(Thread.java:388)
```

- Close the Simulator.
- Heap Dumps are generated in the **build/output/<fqnMainClass>/heapDump/** folder of the project, where **<fqnMainClass>** is the Fully Qualified Name of the Application Main class.



```
example-java-widget [widget-examples] C:\workspa
  .gradle
  .idea
  .vscode
  bin
  build
    adp
    application
    classes
    generated
    output
      .cache
      com.microej.demo.widget.common.Navigati
        bon
        externalResources
        heapDump
          heap-0.heap
          heap-1.heap
          heap-2.heap
          heap-3.heap
          heap-4.heap
```

Generated Heap Dump files

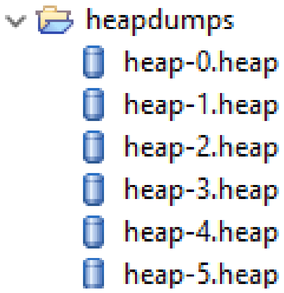
HEAP DUMPER & HEAP ANALYZER (4/8)

IMPORT THE HEAP DUMPS

Heap Dumps can be opened using the Heap Analyzer plug-in. Make sure the Eclipse IDE is installed with the required plugin, then launch it.

In **Eclipse IDE**, create a new empty Project:

- Go to **File > New > Project…**
- Select **General > Project.**
- Give it a name and click **Finish**.
- Copy paste the generated Heap Dumps into this project:



heapdumps
heap-0.heap
heap-1.heap
heap-2.heap
heap-3.heap
heap-4.heap
heap-5.heap

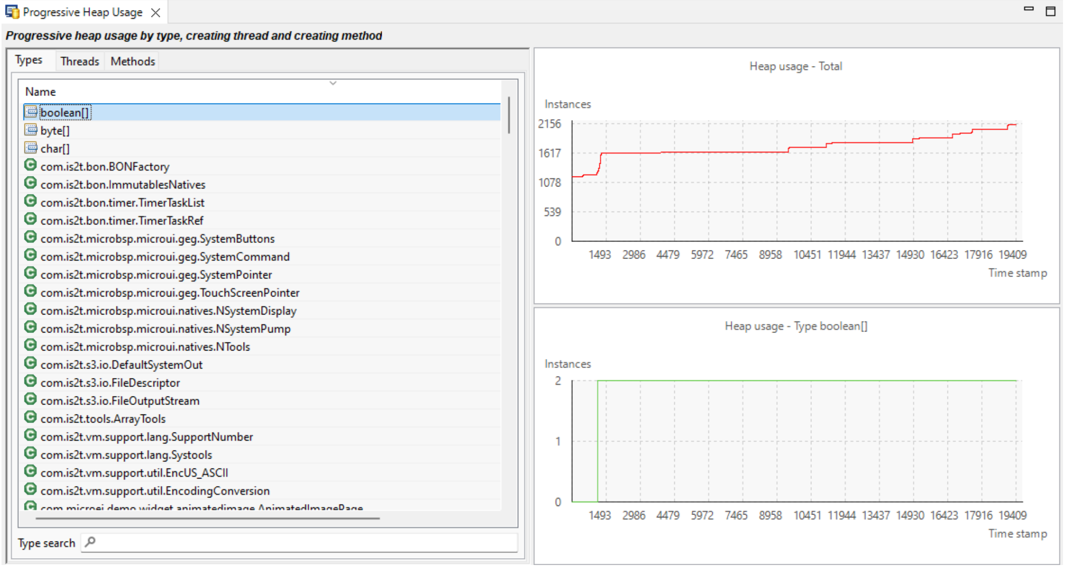
HEAP DUMPER & HEAP ANALYZER (5/8)

PROGRESSIVE HEAP USAGE ANALYSIS

The progressive heap usage tool allows to see the number of instances over time.
To use the tool:

- Select the last .heap file (e.g. **heap-5.heap**)
- Right-click on it and select **Heap Analyzer > Show progressive heap usage**

The following view opens:



Progressive Heap Usage

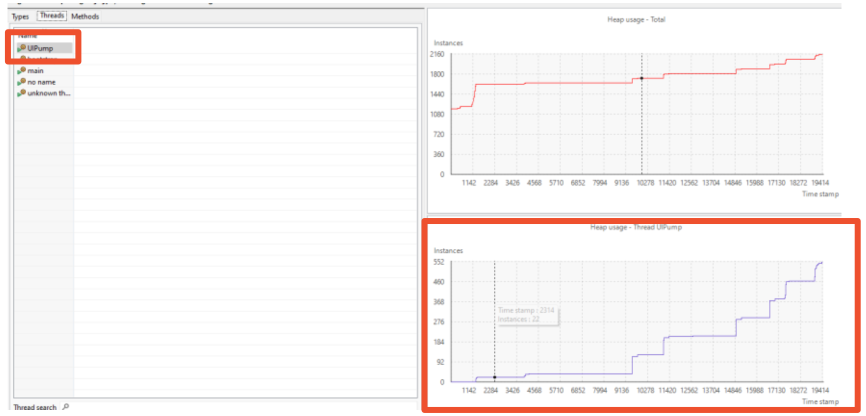
Progressive heap usage by type, creating thread and creating method

Types | Threads | Methods

Name
- boolean[]
- byte[]
- char[]
- com.is2t.bon.BONFactory
- com.is2t.bon.ImmutablesNatives
- com.is2t.bon.timer.TimerTaskList
- com.is2t.bon.timer.TimerTaskRef
- com.is2t.microbsp.microui.geg.SystemButtons
- com.is2t.microbsp.microui.geg.SystemCommand
- com.is2t.microbsp.microui.geg.SystemPointer
- com.is2t.microbsp.microui.geg.TouchScreenPointer
- com.is2t.microbsp.microui.natives.NSystemDisplay
- com.is2t.microbsp.microui.natives.NSystemPump
- com.is2t.microbsp.microui.natives.NTools
- com.is2t.s3.io.DefaultSystemOut
- com.is2t.s3.io.FileDescriptor
- com.is2t.s3.io.FileOutputStream
- com.is2t.tools.ArrayTools
- com.is2t.vm.support.lang.SupportNumber
- com.is2t.vm.support.lang.Systools
- com.is2t.vm.support.util.EncUS_ASCII
- com.is2t.vm.support.util.EncodingConversion
- com.microej.demo.widget.animatedimage.AnimatedImagePage

Type search

In the Threads tab, we can clearly notice that the memory leak is coming from the **UIThread**:

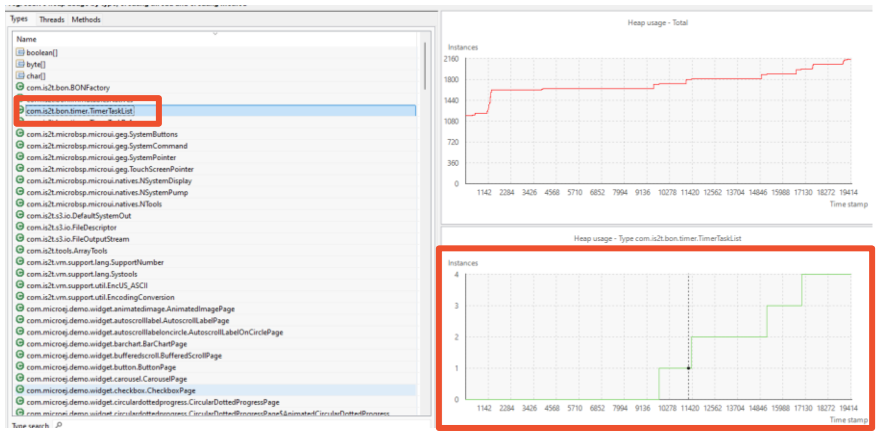


Types | Threads | Methods

- UIPump
- main
- no name
- unknown th...

Thread search

Browsing the types, we notice that the instances of some types are also growing (e.g. **TimerTaskList**):



Types | Threads | Methods

Name
- boolean[]
- byte[]
- char[]
- com.is2t.bon.BONFactory
- com.is2t.bon.timer.TimerTaskList
- com.is2t.microbsp.microui.geg.SystemButtons
- com.is2t.microbsp.microui.geg.SystemCommand
- com.is2t.microbsp.microui.geg.SystemPointer
- com.is2t.microbsp.microui.geg.TouchScreenPointer
- com.is2t.microbsp.microui.natives.NSystemDisplay
- com.is2t.microbsp.microui.natives.NSystemPump
- com.is2t.microbsp.microui.natives.NTools
- com.is2t.s3.io.DefaultSystemOut
- com.is2t.s3.io.FileDescriptor
- com.is2t.s3.io.FileOutputStream
- com.is2t.tools.ArrayTools
- com.is2t.vm.support.lang.SupportNumber
- com.is2t.vm.support.lang.Systools
- com.is2t.vm.support.util.EncUS_ASCII
- com.is2t.vm.support.util.EncodingConversion
- com.microej.demo.widget.animatedimage.AnimatedImagePage
- com.microej.demo.widget.autoscrolllabel.AutoscrollLabelPage
- com.microej.demo.widget.autoscrolllabeloncircle.AutoscrollLabelOnCirclePage
- com.microej.demo.widget.barchart.BarChartPage
- com.microej.demo.widget.bufferedscroll.BufferedScrollPage
- com.microej.demo.widget.button.ButtonPage
- com.microej.demo.widget.carousel.CarouselPage
- com.microej.demo.widget.checkbox.CheckboxPage
- com.microej.demo.widget.circulardottedprogress.CircularDottedProgressPage

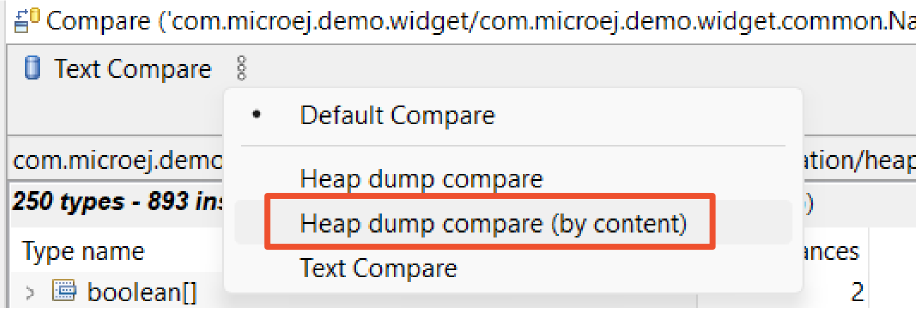
Type search

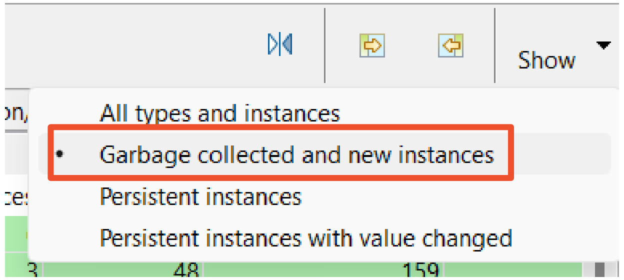
Next step: compare 2 consecutive heap dumps focusing the types that are growing continuously.

HEAP DUMPER & HEAP ANALYZER (6/8)

COMPARE THE HEAP DUMPS

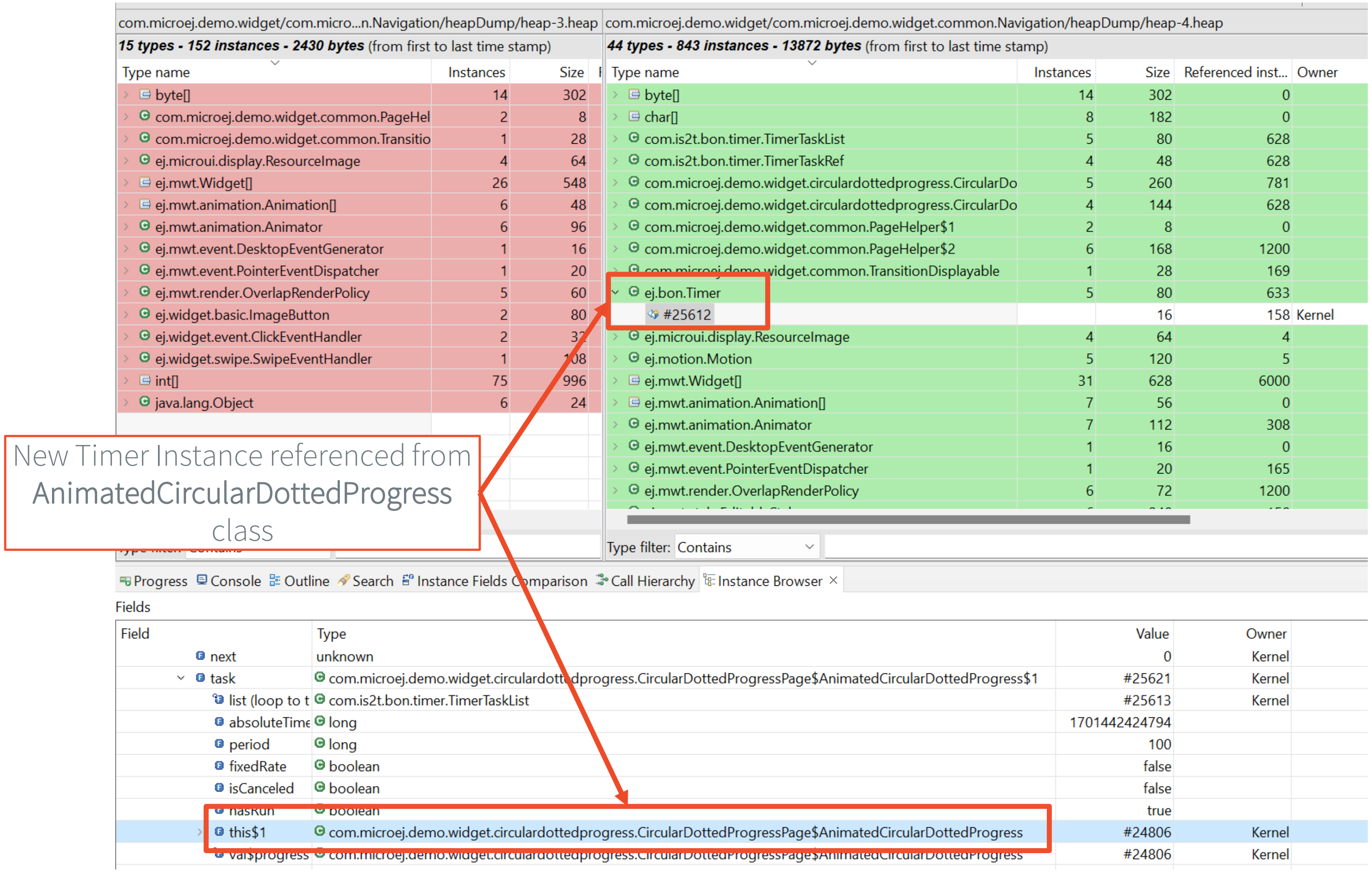
- Right-Click on 2 consecutive **.heap** files.
  Preferably the ones generated just before the Out Of Memory error.
- Click on **Compare With → Each Other**.
- The Heap Viewer opens, select the following configuration:




Show
All types and instances
• Garbage collected and new instances
Persistent instances
Persistent instances with value changed
3
48
159

HEAP DUMPER & HEAP ANALYZER (7/8)

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



com.microej.demo.widget/com.micro...n.Navigation/heapDump/heap-3.heap

15 types - 152 instances - 2430 bytes (from first to last time stamp)

Type name	Instances	Size
byte[]	14	302
com.microej.demo.widget.common.PageHel	2	8
com.microej.demo.widget.common.Transitio	1	28
ej.microui.display.ResourceImage	4	64
ej.mwt.Widget[]	26	548
ej.mwt.animation.Animation[]	6	48
ej.mwt.animation.Animator	6	96
ej.mwt.event.DesktopEventGenerator	1	16
ej.mwt.event.PointerEventDispatcher	1	20
ej.mwt.render.OverlapRenderPolicy	5	60
ej.widget.basic.ImageButton	2	80
ej.widget.event.ClickEventHandler	2	32
ej.widget.swipe.SwipeEventHandler	1	108
int[]	75	996
java.lang.Object	6	24

com.microej.demo.widget/com.microej.demo.widget.common.Navigation/heapDump/heap-4.heap

44 types - 843 instances - 13872 bytes (from first to last time stamp)

Type name	Instances	Size	Referenced inst...	Owner
byte[]	14	302	0	
char[]	8	182	0	
com.is2t.bon.timer.TimerTaskList	5	80	628	
com.is2t.bon.timer.TimerTaskRef	4	48	628	
com.microej.demo.widget.circulardottedprogress.CircularDo	5	260	781	
com.microej.demo.widget.circulardottedprogress.CircularDo	4	144	628	
com.microej.demo.widget.common.PageHelper$1	2	8	0	
com.microej.demo.widget.common.PageHelper$2	6	168	1200	
com.microej.demo.widget.common.TransitionDisplayable	1	28	169	
ej.bon.Timer	5	80	633	
#25612		16	158	Kernel
ej.microui.display.ResourceImage	4	64	4	
ej.motion.Motion	5	120	5	
ej.mwt.Widget[]	31	628	6000	
ej.mwt.animation.Animation[]	7	56	0	
ej.mwt.animation.Animator	7	112	308	
ej.mwt.event.DesktopEventGenerator	1	16	0	
ej.mwt.event.PointerEventDispatcher	1	20	165	
ej.mwt.render.OverlapRenderPolicy	6	72	1200	

Type filter: Contains

New Timer Instance referenced from AnimatedCircularDottedProgress class

Fields

Field	Type	Value	Owner
next	unknown	0	Kernel
task	com.microej.demo.widget.circulardottedprogress.CircularDottedProgressPage$AnimatedCircularDottedProgress$1	#25621	Kernel
list (loop to t	com.is2t.bon.timer.TimerTaskList	#25613	Kernel
absoluteTime	long	1701442424794	
period	long	100	
fixedRate	boolean	false	
isCanceled	boolean	false	
hasRun	boolean	true	
this$1	com.microej.demo.widget.circulardottedprogress.CircularDottedProgressPage$AnimatedCircularDottedProgress	#24806	Kernel
val$progress	com.microej.demo.widget.circulardottedprogress.CircularDottedProgressPage$AnimatedCircularDottedProgress	#24806	Kernel

Guidelines:

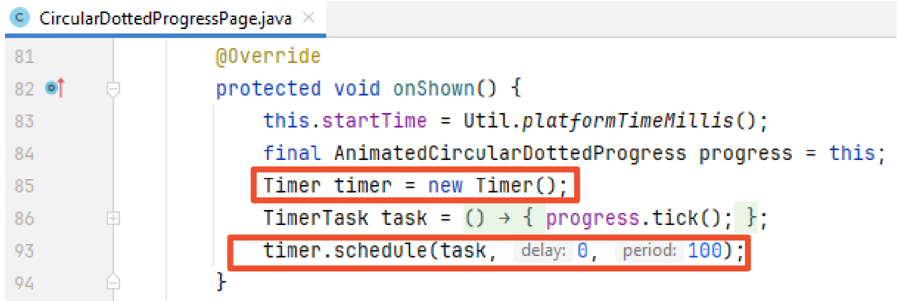
- Lots of new objects have been created (691 new instances)
- Use the **compare by content** option to discard objects that moved but have the same content
- Look for new objects that can have an impact (Thread, Timer, Page, Widget, StyleSheet)

  → 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 (8/8)

ROOT CAUSE ANALYSIS

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



CircularDottedProgressPage.java

```
@Override
protected void onShown() {
    this.startTime = Util.platformTimeMillis();
    final AnimatedCircularDottedProgress progress = this;
    Timer timer = new Timer();
    TimerTask task = () → { progress.tick(); };
    timer.schedule(task,  delay: 0,  period: 100);
}
```

→ 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() {
  System.gc();
  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.

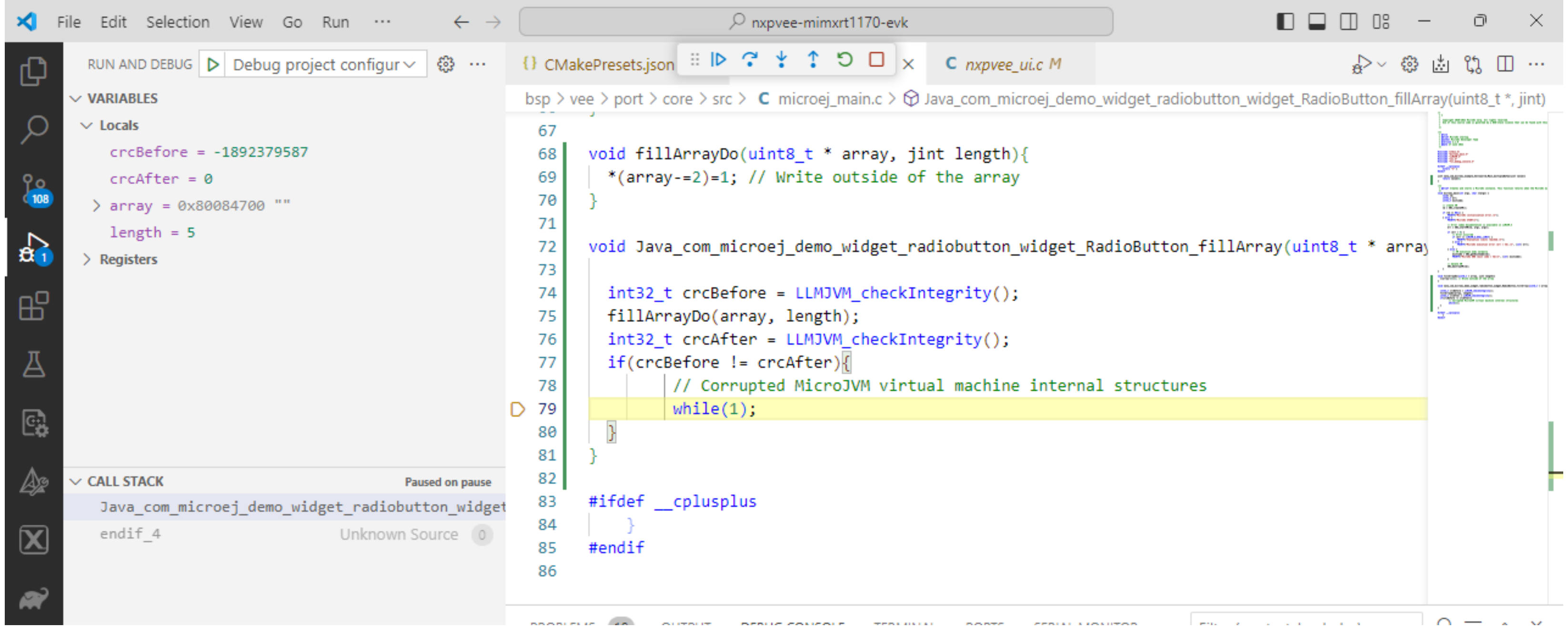
- This feature is for Applications deployed on hardware devices only:
  - If an integrity error is detected, the **LLMJVM_on_CheckIntegrity_error** hook is called and this method returns 0.
  - If no integrity error is detected, a non-zero checksum is returned.

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

CORE ENGINE MEMORY INTEGRITY CHECK (2/3)

REPRODUCE THE ISSUE

- Run the **example-java-widget** application on the device
- Enter the **Radio Button page**, click on one of the buttons
- The GUI should freeze, the Heap is corrupted
- Run the BSP Debug, the execution is stuck in a while loop because the CRC check of the VEE Heap failed:



```
crcBefore = -1892379587
crcAfter = 0
array = 0x80084700 ""
length = 5
```

```
Java_com_microej_demo_widget_radiobutton_widget
endif_4                Unknown Source  0
```

```c
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

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

#ifdef __cplusplus
    }
#endif
```

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.

Identify & Debug Performance Bottlenecks

Study done on a UI Application

IDENTIFYING & DEBUGGING BOTTLENECKS

Tools:

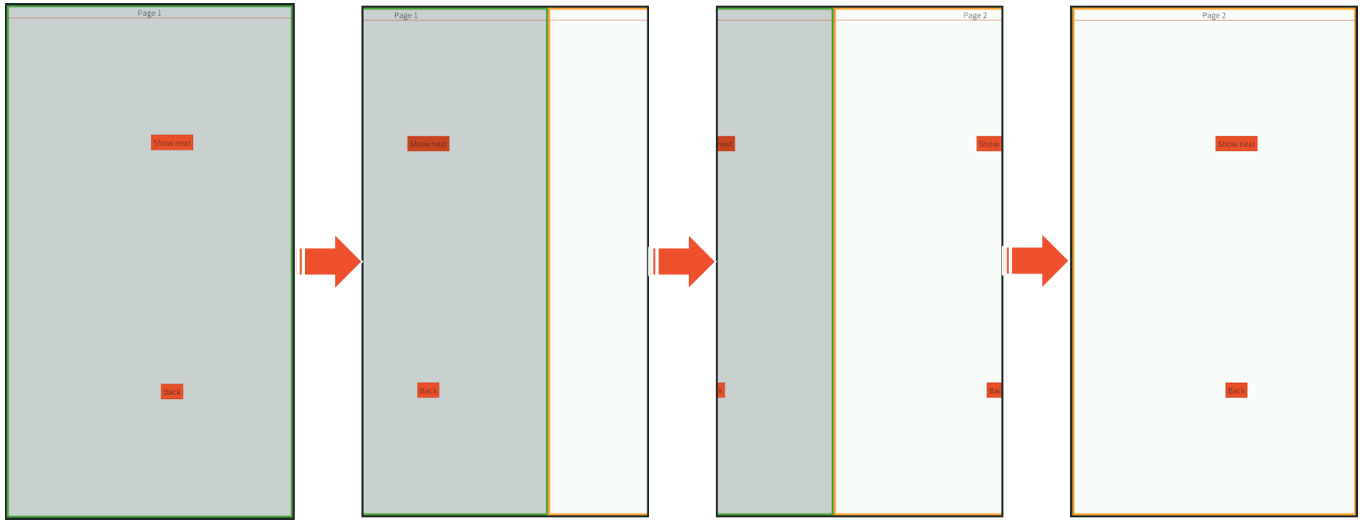
- Flush Visualizer
- Refresh Strategy Highlighting
- SystemView
- MicroUI Event Buffer Dump

Example:

- Identify performance bottlenecks that prevents smooth slide animation

The next slides are using IntelliJ IDEA as IDE.

Note: For UI2 and former versions, please refer to MicroUI and multithreading for a description of the threading model.



Slide animation between 2 pages

Flush Visualizer

Identify & Debug Bottlenecks on the Simulator using the Flush Visualizer tool

PRESENTATION

Building smooth and visually appealing UI applications requires a keen focus on performance. To achieve efficient UI rendering, minimizing unnecessary work that consumes valuable CPU time is essential.

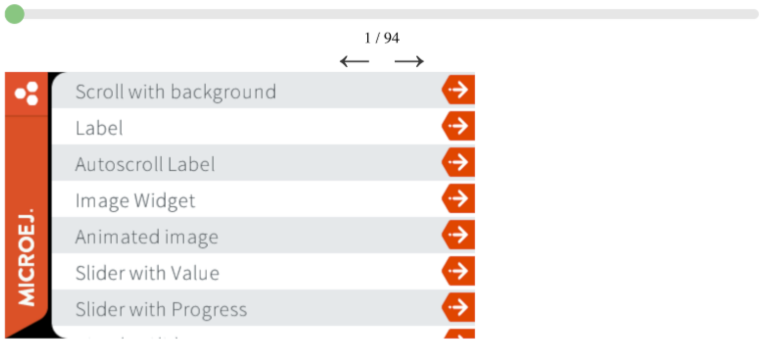
The Flush Visualizer is a tool designed to investigate potential performance bottlenecks in UI applications running on the Simulator. The Flush Visualizer provides the following information:

- A timeline with a step for each flush.
- A screenshot of what is shown on the display at flush time.
- The list of what is done before this flush (and after the previous one) organized as a tree.
- A node of the tree can be either a region (the display or a clip) or a drawing operation.

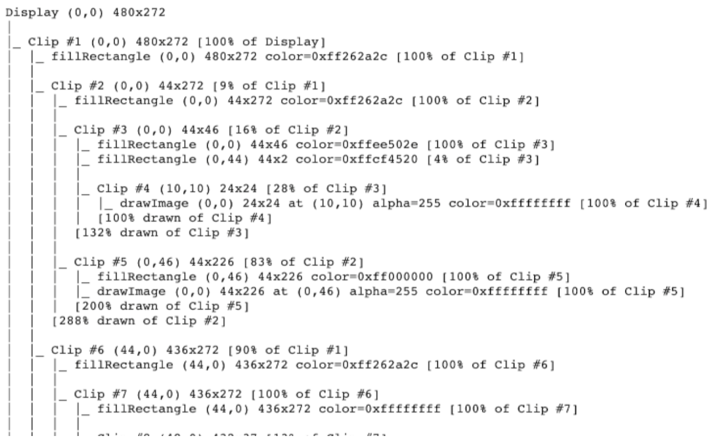
For more information, refer to the Flush Visualizer documentation.

Timeline

Screenshot



Tree of the drawing operations



```
Display (0,0) 480x272
|
|_ Clip #1 (0,0) 480x272 [100% of Display]
|  |_ fillRectangle (0,0) 480x272 color=0xff262a2c [100% of Clip #1]
|  |
|  |_ Clip #2 (0,0) 44x272 [9% of Clip #1]
|  |  |_ fillRectangle (0,0) 44x272 color=0xff262a2c [100% of Clip #2]
|  |  |
|  |  |_ Clip #3 (0,0) 44x46 [16% of Clip #2]
|  |  |  |_ fillRectangle (0,0) 44x46 color=0xffee502e [100% of Clip #3]
|  |  |  |_ fillRectangle (0,44) 44x2 color=0xffcf4520 [4% of Clip #3]
|  |  |  |
|  |  |  |_ Clip #4 (10,10) 24x24 [28% of Clip #3]
|  |  |  |  |_ drawImage (0,0) 24x24 at (10,10) alpha=255 color=0xffffffff [100% of Clip #4]
|  |  |  |  [100% drawn of Clip #4]
|  |  |  [132% drawn of Clip #3]
|  |  |
|  |  |_ Clip #5 (0,46) 44x226 [83% of Clip #2]
|  |  |  |_ fillRectangle (0,46) 44x226 color=0xff000000 [100% of Clip #5]
|  |  |  |_ drawImage (0,0) 44x226 at (0,46) alpha=255 color=0xffffffff [100% of Clip #5]
|  |  |  [200% drawn of Clip #5]
|  |  [288% drawn of Clip #2]
|  |
|  |_ Clip #6 (44,0) 436x272 [90% of Clip #1]
|  |  |_ fillRectangle (44,0) 436x272 color=0xff262a2c [100% of Clip #6]
|  |  |
|  |  |_ Clip #7 (44,0) 436x272 [100% of Clip #6]
|  |  |  |_ fillRectangle (44,0) 436x272 color=0xffffffff [100% of Clip #7]
|  |  |  |
```

ENABLE THE FLUSH VISUALIZER

Requirement (already fulfilled by NXP i.MXRT1170 VEE Port):

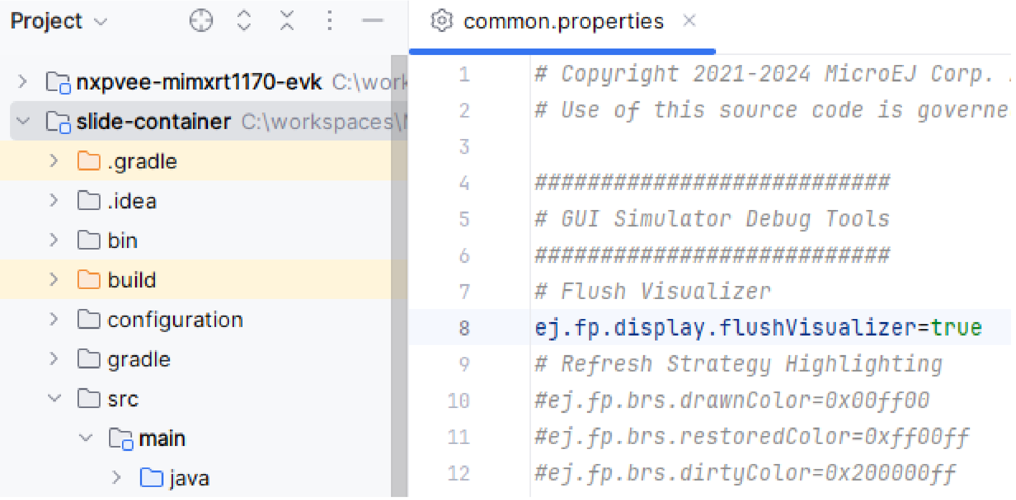
- VEE Port using UI Pack 14.0.0 or later.
- Frontpanel using the Display widget module version 4.+.

Open the **slide-container** example provided in the training package.

Add the dependency to the VEE Port in the **slide-container/build.gradle.kts** file.

To enable the Flush Visualizer on Simulator:

- Set the **ej.fp.display.flushVisualizer** to **true** in the application options (configuration/common.properties)
- Click on **Run.**

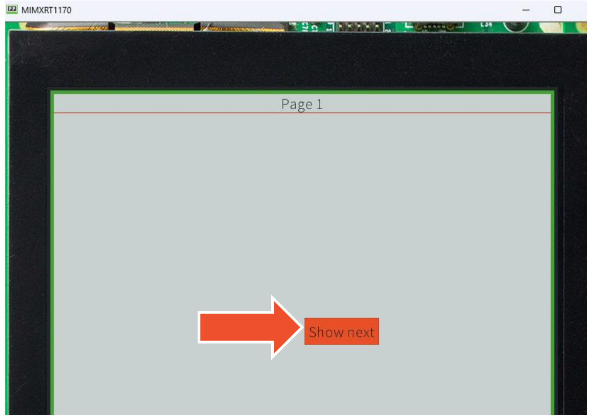


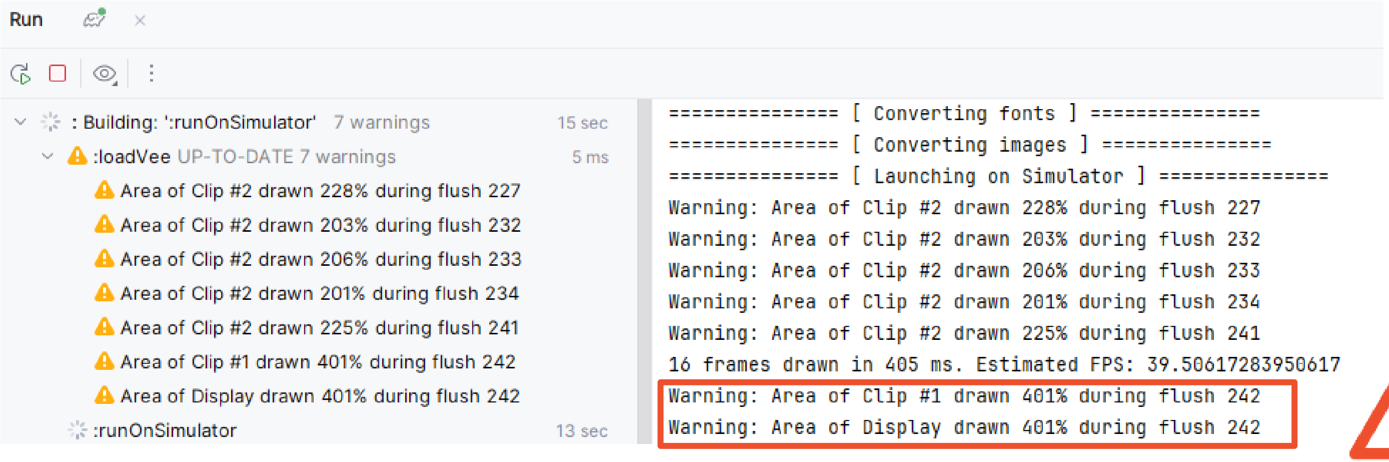
```
# Copyright 2021-2024 MicroEJ Corp. .
# Use of this source code is governe

############################
# GUI Simulator Debug Tools
############################
# Flush Visualizer
ej.fp.display.flushVisualizer=true
# Refresh Strategy Highlighting
#ej.fp.brs.drawnColor=0x00ff00
#ej.fp.brs.restoredColor=0xff00ff
#ej.fp.brs.dirtyColor=0x200000ff
```

Application options of the slide-container sample

REPRODUCE THE ISSUE

Click on the **Show next** button, the Flush Visualizer displays a message in the Console:





```
: Building: ':runOnSimulator'  7 warnings    15 sec
  :loadVee UP-TO-DATE 7 warnings              5 ms
    Area of Clip #2 drawn 228% during flush 227
    Area of Clip #2 drawn 203% during flush 232
    Area of Clip #2 drawn 206% during flush 233
    Area of Clip #2 drawn 201% during flush 234
    Area of Clip #2 drawn 225% during flush 241
    Area of Clip #1 drawn 401% during flush 242
    Area of Display drawn 401% during flush 242
  :runOnSimulator                             13 sec
```

```
=============== [ Converting fonts ] ===============
=============== [ Converting images ] ===============
=============== [ Launching on Simulator ] ===============
Warning: Area of Clip #2 drawn 228% during flush 227
Warning: Area of Clip #2 drawn 203% during flush 232
Warning: Area of Clip #2 drawn 206% during flush 233
Warning: Area of Clip #2 drawn 201% during flush 234
Warning: Area of Clip #2 drawn 225% during flush 241
16 frames drawn in 405 ms. Estimated FPS: 39.50617283950617
Warning: Area of Clip #1 drawn 401% during flush 242
Warning: Area of Display drawn 401% during flush 242
```

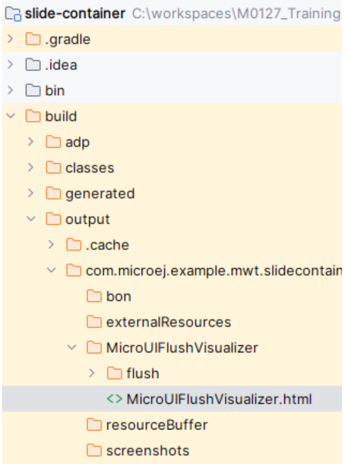
A value of 100% indicates that the area drawn is equivalent to the surface of the region.
A value of 200% indicates that the area drawn is equivalent to twice the surface of the region.
A perfect application has 100% of its root region drawn but its very unlikely for an application that draws anything else than a rectangle or an image.
A total area drawn between 100% to 200% is the norm in practice because widgets often overlap.

However, if the total area drawn is **bigger than 200%,** that means that the total **surface of the region was drawn more than twice**. Probably meaning that a **lot of drawings are done above others**.

→ Next steps are about Identifying those drawings to reduce number of drawings done (or their surface).

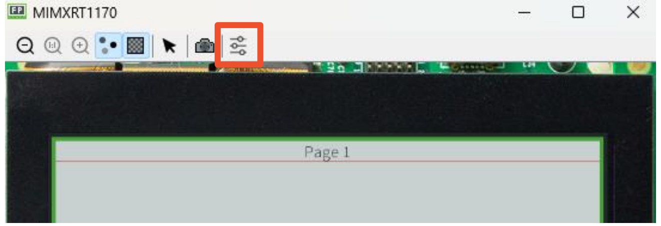
FLUSH ANALYZER REPORT

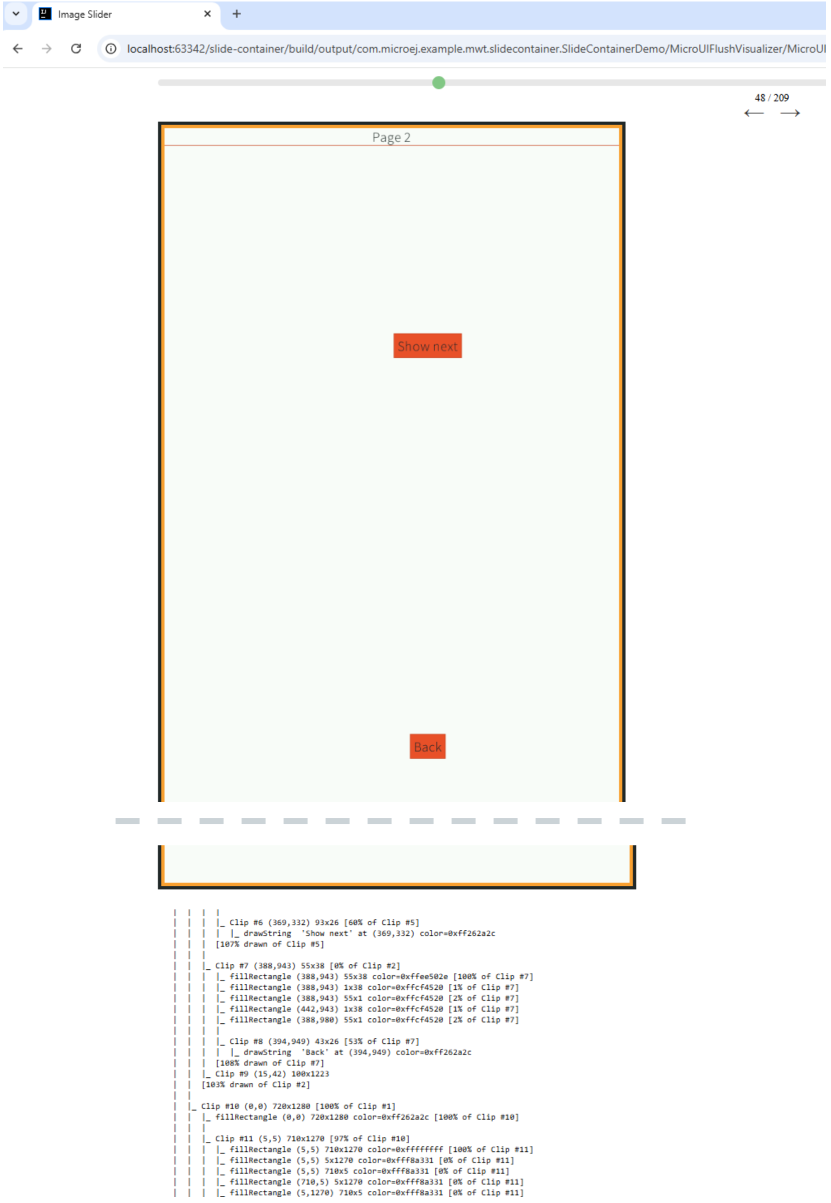
The Flush Visualizer report is accessible in the project output folder. It can be visualized in a web browser:



```
slide-container C:\workspaces\M0127_Training
  > .gradle
  > .idea
  > bin
  v build
    > adp
    > classes
    > generated
    v output
      > .cache
      v com.microej.example.mwt.slidecontain
          bon
          externalResources
        v MicroUIFlushVisualizer
          > flush
            MicroUIFlushVisualizer.html
          resourceBuffer
          screenshots
```

The report can also be opened clicking this button on the Front Panel:



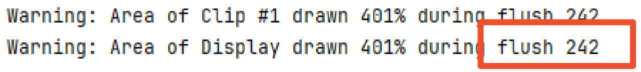


localhost:63342/slide-container/build/output/com.microej.example.mwt.slidecontainer.SlideContainerDemo/MicroUIFlushVisualizer/MicroUI

```
|   |   |   |
|   |   |   |_ Clip #6 (369,332) 93x26 [60% of Clip #5]
|   |   |   |   |_ drawString  'Show next' at (369,332) color=0xff262a2c
|   |   |   [107% drawn of Clip #5]
|   |   |
|   |   |_ Clip #7 (388,943) 55x38 [0% of Clip #2]
|   |   |   |_ fillRectangle (388,943) 55x38 color=0xffee502e [100% of Clip #7]
|   |   |   |_ fillRectangle (388,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|   |   |   |_ fillRectangle (388,943) 55x1 color=0xffcf4520 [2% of Clip #7]
|   |   |   |_ fillRectangle (442,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|   |   |   |_ fillRectangle (388,980) 55x1 color=0xffcf4520 [2% of Clip #7]
|   |   |   |
|   |   |   |_ Clip #8 (394,949) 43x26 [53% of Clip #7]
|   |   |   |   |_ drawString  'Back' at (394,949) color=0xff262a2c
|   |   |   [108% drawn of Clip #7]
|   |   |_ Clip #9 (15,42) 100x1223
|   |   [103% drawn of Clip #2]
|   |
|   |_ Clip #10 (0,0) 720x1280 [100% of Clip #1]
|   |   |_ fillRectangle (0,0) 720x1280 color=0xff262a2c [100% of Clip #10]
|   |   |
|   |   |_ Clip #11 (5,5) 710x1270 [97% of Clip #10]
|   |   |   |_ fillRectangle (5,5) 710x1270 color=0xffffffff [100% of Clip #11]
|   |   |   |_ fillRectangle (5,5) 5x1270 color=0xfff8a331 [0% of Clip #11]
|   |   |   |_ fillRectangle (5,5) 710x5 color=0xfff8a331 [0% of Clip #11]
|   |   |   |_ fillRectangle (710,5) 5x1270 color=0xfff8a331 [0% of Clip #11]
|   |   |   |_ fillRectangle (5,1270) 710x5 color=0xfff8a331 [0% of Clip #11]
```

REPORT ANALYSIS (1/2)

Open the report and move the slider to the flush frame corresponding to the 401% of drawings (e.g. frame 242):



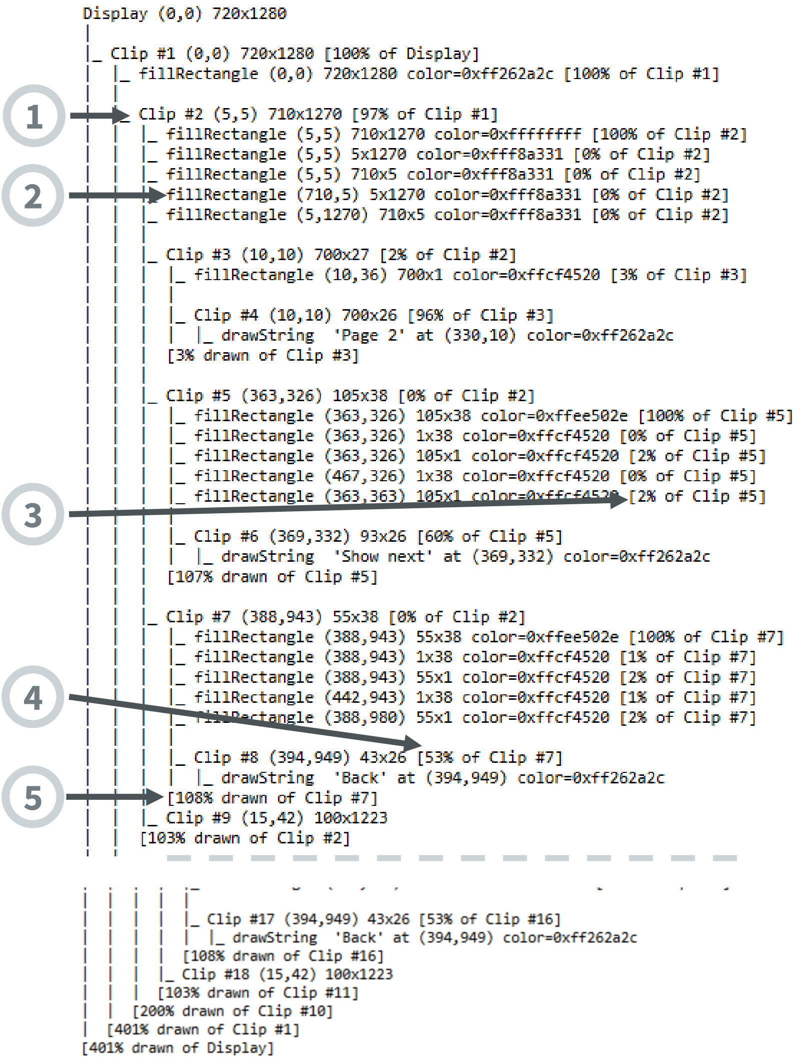
```
Warning: Area of Clip #1 drawn 401% during flush 242
Warning: Area of Display drawn 401% during flush 242
```

The report is available at the bottom of the page.

You should observe the report displayed beside.

The following information are provided:

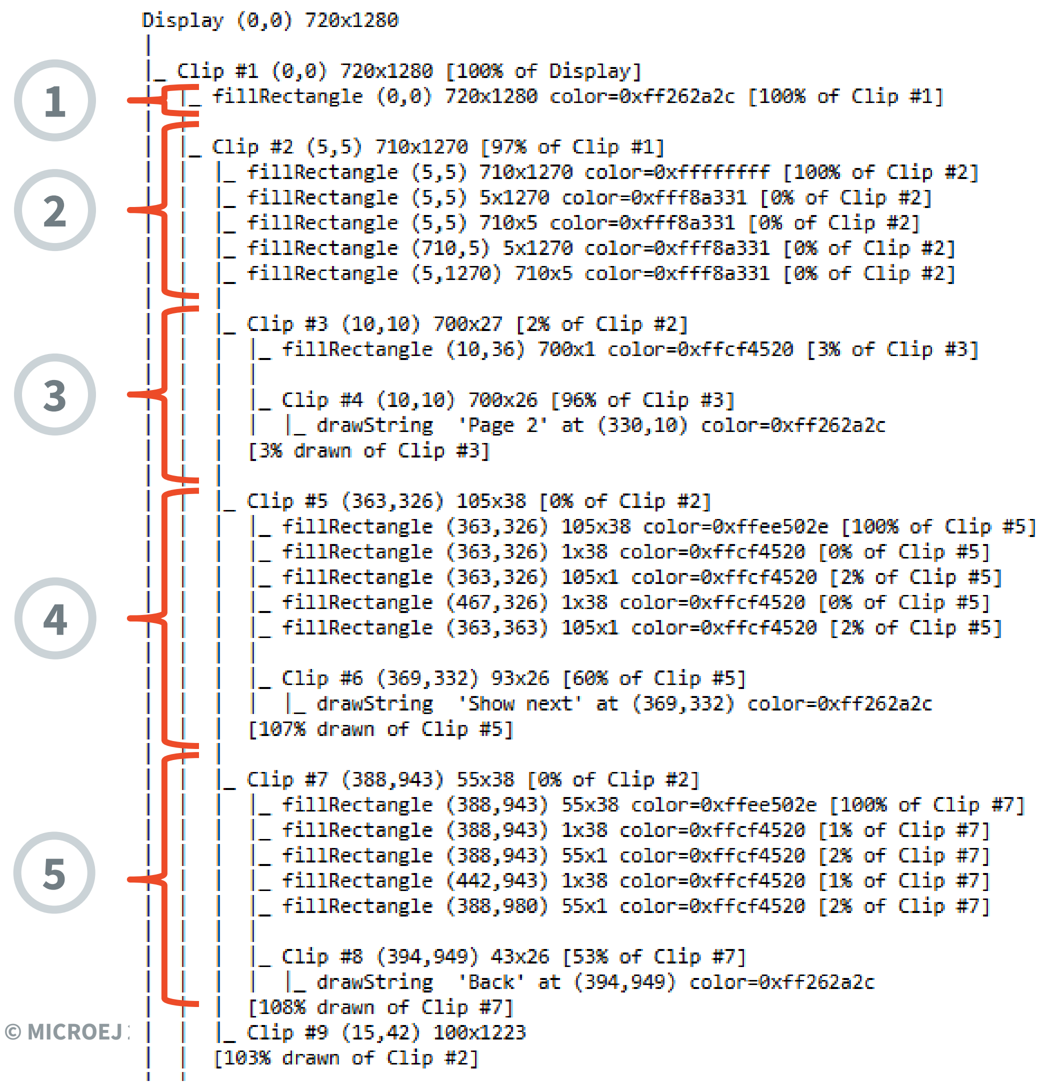
- The operations before a flush are structured as a tree, where nodes represent either:
  1. a region (display or clip)
  2. a drawing operation.
3. Each region has defined bounds, can contain other nodes, and displays the percentage of its parent region it covers.
4. Some drawings compute their coverage percentage
5. Each region provides a summary of the total percentage covered recursively.



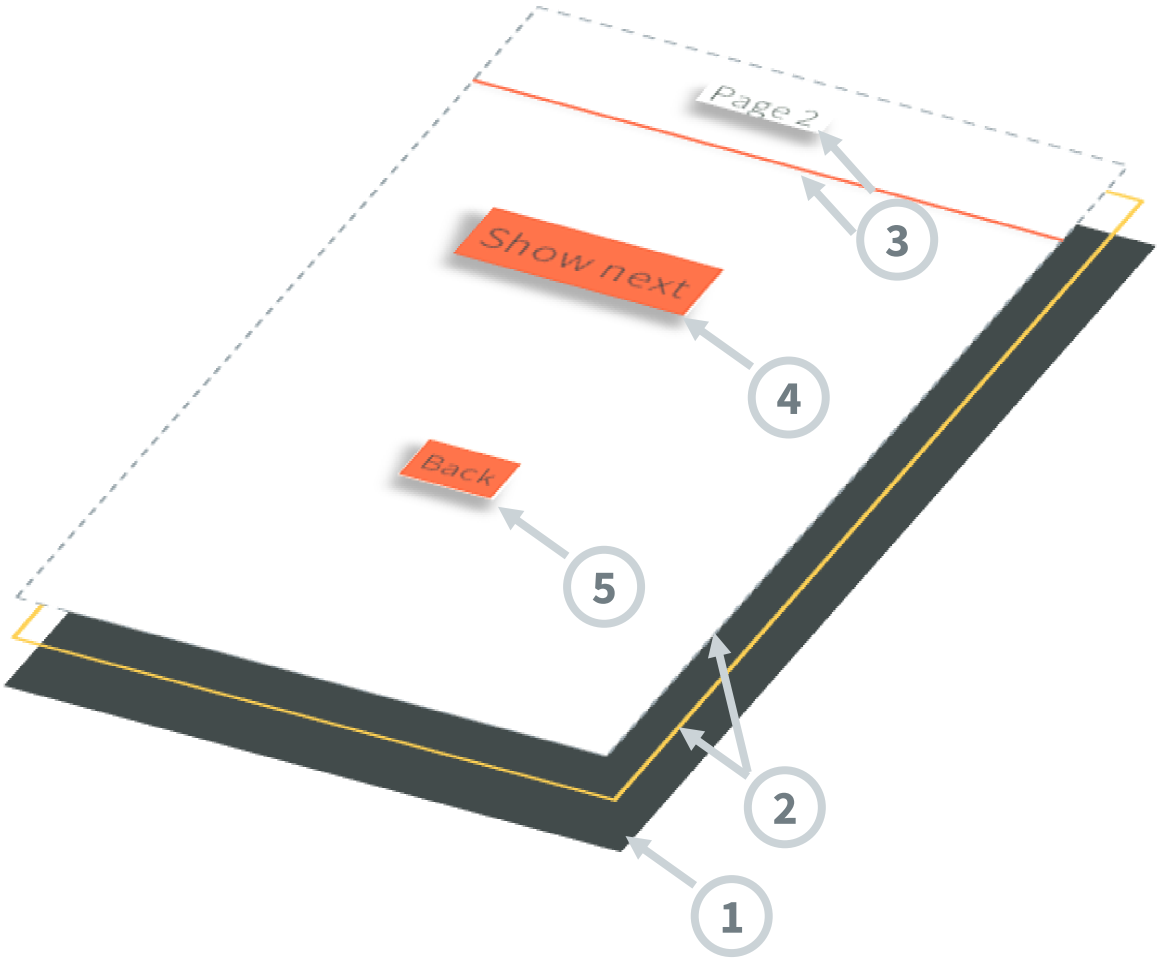
```
Display (0,0) 720x1280
|
|_ Clip #1 (0,0) 720x1280 [100% of Display]
|   |_ fillRectangle (0,0) 720x1280 color=0xff262a2c [100% of Clip #1]
|   |
|   |_ Clip #2 (5,5) 710x1270 [97% of Clip #1]
|   |   |_ fillRectangle (5,5) 710x1270 color=0xffffffff [100% of Clip #2]
|   |   |_ fillRectangle (5,5) 5x1270 color=0xfff8a331 [0% of Clip #2]
|   |   |_ fillRectangle (5,5) 710x5 color=0xfff8a331 [0% of Clip #2]
|   |   |_ fillRectangle (710,5) 5x1270 color=0xfff8a331 [0% of Clip #2]
|   |   |_ fillRectangle (5,1270) 710x5 color=0xfff8a331 [0% of Clip #2]
|   |   |
|   |   |_ Clip #3 (10,10) 700x27 [2% of Clip #2]
|   |   |   |_ fillRectangle (10,36) 700x1 color=0xffcf4520 [3% of Clip #3]
|   |   |   |
|   |   |   |_ Clip #4 (10,10) 700x26 [96% of Clip #3]
|   |   |   |   |_ drawString  'Page 2' at (330,10) color=0xff262a2c
|   |   |   [3% drawn of Clip #3]
|   |   |
|   |   |_ Clip #5 (363,326) 105x38 [0% of Clip #2]
|   |   |   |_ fillRectangle (363,326) 105x38 color=0xffee502e [100% of Clip #5]
|   |   |   |_ fillRectangle (363,326) 1x38 color=0xffcf4520 [0% of Clip #5]
|   |   |   |_ fillRectangle (363,326) 105x1 color=0xffcf4520 [2% of Clip #5]
|   |   |   |_ fillRectangle (467,326) 1x38 color=0xffcf4520 [0% of Clip #5]
|   |   |   |_ fillRectangle (363,363) 105x1 color=0xffcf4520 [2% of Clip #5]
|   |   |   |
|   |   |   |_ Clip #6 (369,332) 93x26 [60% of Clip #5]
|   |   |   |   |_ drawString  'Show next' at (369,332) color=0xff262a2c
|   |   |   [107% drawn of Clip #5]
|   |   |
|   |   |_ Clip #7 (388,943) 55x38 [0% of Clip #2]
|   |   |   |_ fillRectangle (388,943) 55x38 color=0xffee502e [100% of Clip #7]
|   |   |   |_ fillRectangle (388,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|   |   |   |_ fillRectangle (388,943) 55x1 color=0xffcf4520 [2% of Clip #7]
|   |   |   |_ fillRectangle (442,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|   |   |   |_ fillRectangle (388,980) 55x1 color=0xffcf4520 [2% of Clip #7]
|   |   |   |
|   |   |   |_ Clip #8 (394,949) 43x26 [53% of Clip #7]
|   |   |   |   |_ drawString  'Back' at (394,949) color=0xff262a2c
|   |   |   [108% drawn of Clip #7]
|   |   |_ Clip #9 (15,42) 100x1223
|   |   [103% drawn of Clip #2]
|   |

|   |   |   |   |
|   |   |   |   |_ Clip #17 (394,949) 43x26 [53% of Clip #16]
|   |   |   |   |   |_ drawString  'Back' at (394,949) color=0xff262a2c
|   |   |   |   [108% drawn of Clip #16]
|   |   |   |_ Clip #18 (15,42) 100x1223
|   |   |   [103% drawn of Clip #11]
|   |   [200% drawn of Clip #10]
|   [401% drawn of Clip #1]
[401% drawn of Display]
```

REPORT ANALYSIS (2/2)

SLIDE-CONTAINER PAGE BREAKDOWN

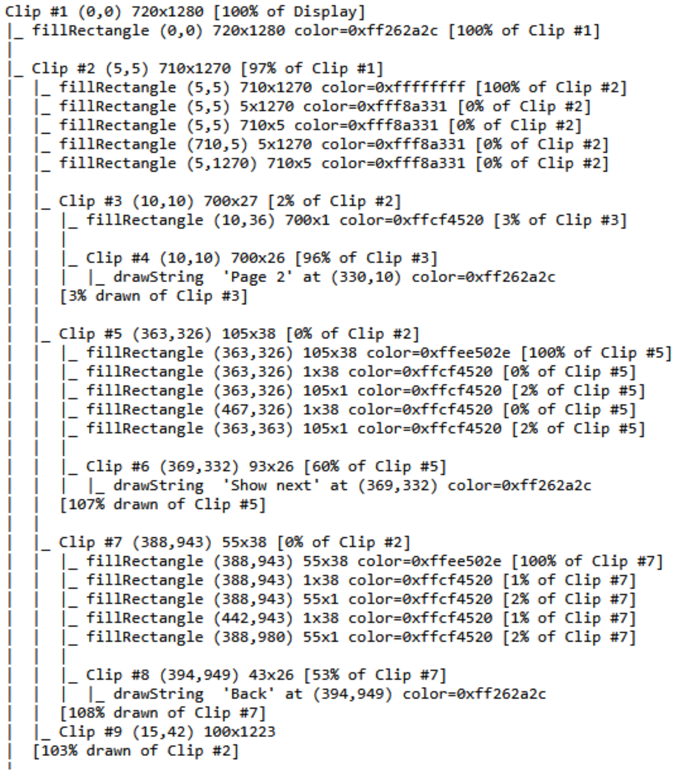


```
Display (0,0) 720x1280
|
|_ Clip #1 (0,0) 720x1280 [100% of Display]
|  |_ fillRectangle (0,0) 720x1280 color=0xff262a2c [100% of Clip #1]
|  |
|  |_ Clip #2 (5,5) 710x1270 [97% of Clip #1]
|  |   |_ fillRectangle (5,5) 710x1270 color=0xffffffff [100% of Clip #2]
|  |   |_ fillRectangle (5,5) 5x1270 color=0xfff8a331 [0% of Clip #2]
|  |   |_ fillRectangle (5,5) 710x5 color=0xfff8a331 [0% of Clip #2]
|  |   |_ fillRectangle (710,5) 5x1270 color=0xfff8a331 [0% of Clip #2]
|  |   |_ fillRectangle (5,1270) 710x5 color=0xfff8a331 [0% of Clip #2]
|  |   |
|  |   |_ Clip #3 (10,10) 700x27 [2% of Clip #2]
|  |   |   |_ fillRectangle (10,36) 700x1 color=0xffcf4520 [3% of Clip #3]
|  |   |   |
|  |   |   |_ Clip #4 (10,10) 700x26 [96% of Clip #3]
|  |   |   |   |_ drawString  'Page 2' at (330,10) color=0xff262a2c
|  |   |   [3% drawn of Clip #3]
|  |   |
|  |   |_ Clip #5 (363,326) 105x38 [0% of Clip #2]
|  |   |   |_ fillRectangle (363,326) 105x38 color=0xffee502e [100% of Clip #5]
|  |   |   |_ fillRectangle (363,326) 1x38 color=0xffcf4520 [0% of Clip #5]
|  |   |   |_ fillRectangle (363,326) 105x1 color=0xffcf4520 [2% of Clip #5]
|  |   |   |_ fillRectangle (467,326) 1x38 color=0xffcf4520 [0% of Clip #5]
|  |   |   |_ fillRectangle (363,363) 105x1 color=0xffcf4520 [2% of Clip #5]
|  |   |   |
|  |   |   |_ Clip #6 (369,332) 93x26 [60% of Clip #5]
|  |   |   |   |_ drawString  'Show next' at (369,332) color=0xff262a2c
|  |   |   [107% drawn of Clip #5]
|  |   |
|  |   |_ Clip #7 (388,943) 55x38 [0% of Clip #2]
|  |   |   |_ fillRectangle (388,943) 55x38 color=0xffee502e [100% of Clip #7]
|  |   |   |_ fillRectangle (388,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|  |   |   |_ fillRectangle (388,943) 55x1 color=0xffcf4520 [2% of Clip #7]
|  |   |   |_ fillRectangle (442,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|  |   |   |_ fillRectangle (388,980) 55x1 color=0xffcf4520 [2% of Clip #7]
|  |   |   |
|  |   |   |_ Clip #8 (394,949) 43x26 [53% of Clip #7]
|  |   |   |   |_ drawString  'Back' at (394,949) color=0xff262a2c
|  |   |   [108% drawn of Clip #7]
|  |   |_ Clip #9 (15,42) 100x1223
|  |  [103% drawn of Clip #2]
|  |
```



ROOT CAUSE ANALYSIS

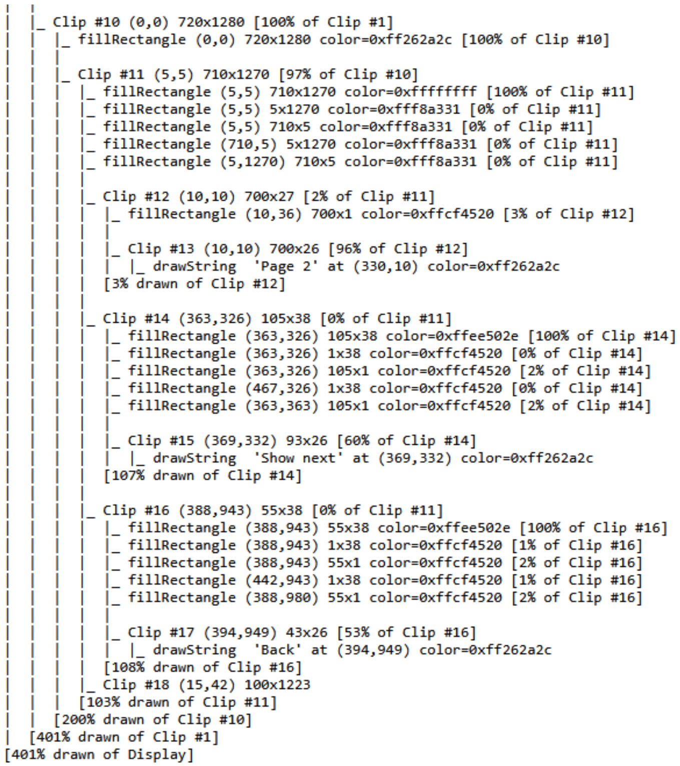
Taking a look at the report, we notice that the page is actually rendered 2 times, exactly the same way:



```
Clip #1 (0,0) 720x1280 [100% of Display]
|_ fillRectangle (0,0) 720x1280 color=0xff262a2c [100% of Clip #1]
|
|_ Clip #2 (5,5) 710x1270 [97% of Clip #1]
|   |_ fillRectangle (5,5) 710x1270 color=0xffffffff [100% of Clip #2]
|   |_ fillRectangle (5,5) 5x1270 color=0xfff8a331 [0% of Clip #2]
|   |_ fillRectangle (5,5) 710x5 color=0xfff8a331 [0% of Clip #2]
|   |_ fillRectangle (710,5) 5x1270 color=0xfff8a331 [0% of Clip #2]
|   |_ fillRectangle (5,1270) 710x5 color=0xfff8a331 [0% of Clip #2]
|   |
|   |_ Clip #3 (10,10) 700x27 [2% of Clip #2]
|   |   |_ fillRectangle (10,36) 700x1 color=0xffcf4520 [3% of Clip #3]
|   |   |
|   |   |_ Clip #4 (10,10) 700x26 [96% of Clip #3]
|   |   |   |_ drawString  'Page 2' at (330,10) color=0xff262a2c
|   |   [3% drawn of Clip #3]
|   |
|   |_ Clip #5 (363,326) 105x38 [0% of Clip #2]
|   |   |_ fillRectangle (363,326) 105x38 color=0xffee502e [100% of Clip #5]
|   |   |_ fillRectangle (363,326) 1x38 color=0xffcf4520 [0% of Clip #5]
|   |   |_ fillRectangle (363,326) 105x1 color=0xffcf4520 [2% of Clip #5]
|   |   |_ fillRectangle (467,326) 1x38 color=0xffcf4520 [0% of Clip #5]
|   |   |_ fillRectangle (363,363) 105x1 color=0xffcf4520 [2% of Clip #5]
|   |   |
|   |   |_ Clip #6 (369,332) 93x26 [60% of Clip #5]
|   |   |   |_ drawString  'Show next' at (369,332) color=0xff262a2c
|   |   [107% drawn of Clip #5]
|   |
|   |_ Clip #7 (388,943) 55x38 [0% of Clip #2]
|   |   |_ fillRectangle (388,943) 55x38 color=0xffee502e [100% of Clip #7]
|   |   |_ fillRectangle (388,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|   |   |_ fillRectangle (388,943) 55x1 color=0xffcf4520 [2% of Clip #7]
|   |   |_ fillRectangle (442,943) 1x38 color=0xffcf4520 [1% of Clip #7]
|   |   |_ fillRectangle (388,980) 55x1 color=0xffcf4520 [2% of Clip #7]
|   |   |
|   |   |_ Clip #8 (394,949) 43x26 [53% of Clip #7]
|   |   |   |_ drawString  'Back' at (394,949) color=0xff262a2c
|   |   [108% drawn of Clip #7]
|   |_ Clip #9 (15,42) 100x1223
|   [103% drawn of Clip #2]
|
```

1st drawing of the page

updatePosition()

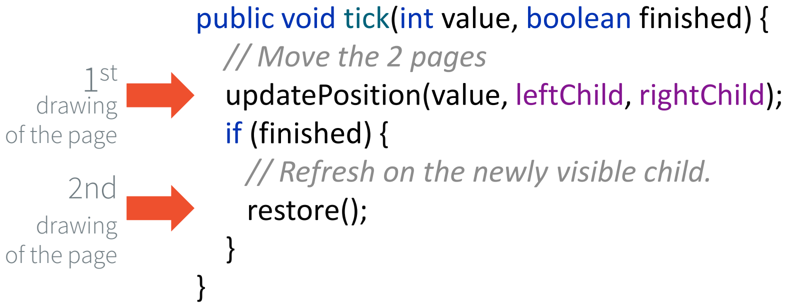


```
|   |
|   |_ Clip #10 (0,0) 720x1280 [100% of Clip #1]
|   |   |_ fillRectangle (0,0) 720x1280 color=0xff262a2c [100% of Clip #10]
|   |   |
|   |   |_ Clip #11 (5,5) 710x1270 [97% of Clip #10]
|   |   |   |_ fillRectangle (5,5) 710x1270 color=0xffffffff [100% of Clip #11]
|   |   |   |_ fillRectangle (5,5) 5x1270 color=0xfff8a331 [0% of Clip #11]
|   |   |   |_ fillRectangle (5,5) 710x5 color=0xfff8a331 [0% of Clip #11]
|   |   |   |_ fillRectangle (710,5) 5x1270 color=0xfff8a331 [0% of Clip #11]
|   |   |   |_ fillRectangle (5,1270) 710x5 color=0xfff8a331 [0% of Clip #11]
|   |   |   |
|   |   |   |_ Clip #12 (10,10) 700x27 [2% of Clip #11]
|   |   |   |   |_ fillRectangle (10,36) 700x1 color=0xffcf4520 [3% of Clip #12]
|   |   |   |   |
|   |   |   |   |_ Clip #13 (10,10) 700x26 [96% of Clip #12]
|   |   |   |   |   |_ drawString  'Page 2' at (330,10) color=0xff262a2c
|   |   |   |   [3% drawn of Clip #12]
|   |   |   |
|   |   |   |_ Clip #14 (363,326) 105x38 [0% of Clip #11]
|   |   |   |   |_ fillRectangle (363,326) 105x38 color=0xffee502e [100% of Clip #14]
|   |   |   |   |_ fillRectangle (363,326) 1x38 color=0xffcf4520 [0% of Clip #14]
|   |   |   |   |_ fillRectangle (363,326) 105x1 color=0xffcf4520 [2% of Clip #14]
|   |   |   |   |_ fillRectangle (467,326) 1x38 color=0xffcf4520 [0% of Clip #14]
|   |   |   |   |_ fillRectangle (363,363) 105x1 color=0xffcf4520 [2% of Clip #14]
|   |   |   |   |
|   |   |   |   |_ Clip #15 (369,332) 93x26 [60% of Clip #14]
|   |   |   |   |   |_ drawString  'Show next' at (369,332) color=0xff262a2c
|   |   |   |   [107% drawn of Clip #14]
|   |   |   |
|   |   |   |_ Clip #16 (388,943) 55x38 [0% of Clip #11]
|   |   |   |   |_ fillRectangle (388,943) 55x38 color=0xffee502e [100% of Clip #16]
|   |   |   |   |_ fillRectangle (388,943) 1x38 color=0xffcf4520 [1% of Clip #16]
|   |   |   |   |_ fillRectangle (388,943) 55x1 color=0xffcf4520 [2% of Clip #16]
|   |   |   |   |_ fillRectangle (442,943) 1x38 color=0xffcf4520 [1% of Clip #16]
|   |   |   |   |_ fillRectangle (388,980) 55x1 color=0xffcf4520 [2% of Clip #16]
|   |   |   |   |
|   |   |   |   |_ Clip #17 (394,949) 43x26 [53% of Clip #16]
|   |   |   |   |   |_ drawString  'Back' at (394,949) color=0xff262a2c
|   |   |   |   [108% drawn of Clip #16]
|   |   |   |_ Clip #18 (15,42) 100x1223
|   |   |   [103% drawn of Clip #11]
|   |   [200% drawn of Clip #10]
|   [401% drawn of Clip #1]
[401% drawn of Display]
```

2nd drawing of the page

restore()

Taking a look at the code, we notice that the page is fully redrawn when the transition animation is over (SlideContainer class):



1st drawing of the page

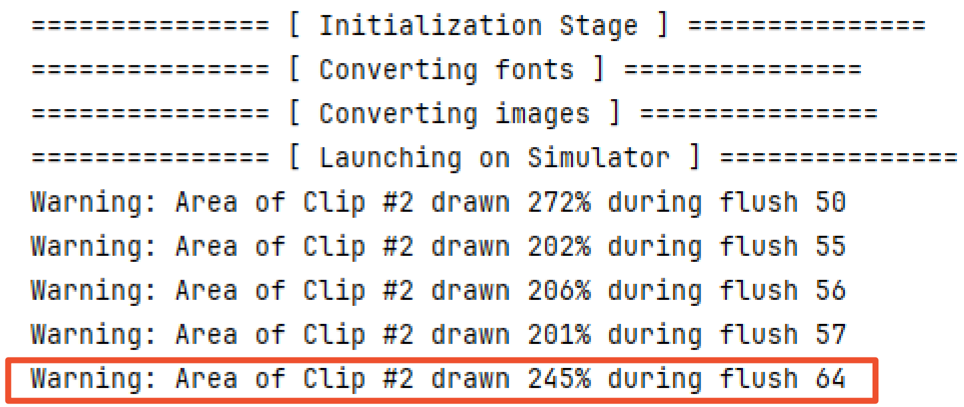
2nd drawing of the page

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

FIX PROPOSAL

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);
   }
}
```



```
=============== [ Initialization Stage ] ===============
=============== [ Converting fonts ] ===============
=============== [ Converting images ] ===============
=============== [ Launching on Simulator ] ===============
Warning: Area of Clip #2 drawn 272% during flush 50
Warning: Area of Clip #2 drawn 202% during flush 55
Warning: Area of Clip #2 drawn 206% during flush 56
Warning: Area of Clip #2 drawn 201% during flush 57
Warning: Area of Clip #2 drawn 245% during flush 64
```

Next step: investigate why the area drawn is still above 200% (not part of this training)

Refresh Strategy Highlighting

Identify & Debug Bottlenecks on the Simulator using the Refresh Strategy Highlighting

REFRESH STRATEGY HIGHLIGHTING

This tool is complementary to the Flush Visualizer tool.

A buffer refresh strategy is responsible for making sure that what is shown on the display contains all the drawings—the ones done since the last flush and the past.

To achieve that, it detects the drawn regions and refreshes the necessary data in the back buffer.

This information can also be used to understand what happens for each frame in terms of drawings and refreshes. It may be beneficial to identify performance issues.

The drawn and restored regions can be very different depending on the selected strategy and the associated options. See Buffer Refresh Strategy for more information about the different strategies and their behavior.

See Refresh Strategy Highlighting documentation for more information about this tool.

ENABLE THE REFRESH STRATEGY HIGHLIGHTING

The following highlights can be enabled:

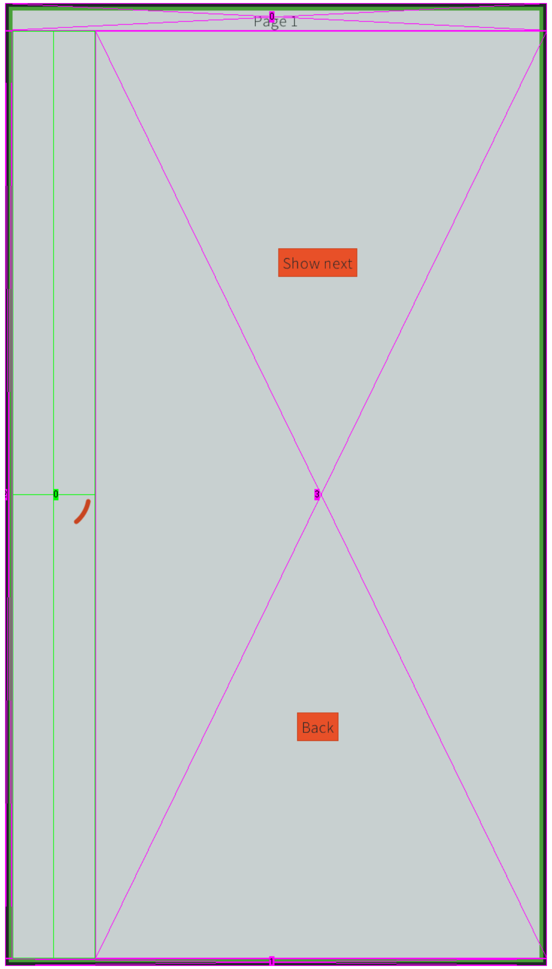
- Drawn Region(s)
- Restored Region(s)
- Dirty Region(s)

Enable the highlights in the **slide-container** example:

- Open the **configuration/common.properties** file
- Add the following properties, one per highlight type:

```
ej.fp.brs.drawnColor=0xff00ff00 // GREEN color
ej.fp.brs.restoredColor=0xffff00ff // PURPLE color
ej.fp.brs.dirtyColor=0x200000ff // BLUE color
```

- Save **common.properties**
- Run the application on Simulator
- Highlights are displayed in the Simulator. They can also be visualized in the Flush Visualizer report.



ANALYSIS

The render area of the **CircularIndeterminateProgress** widget (green area in DOCK LEFT) is taking all the height of the screen.

Analysis:

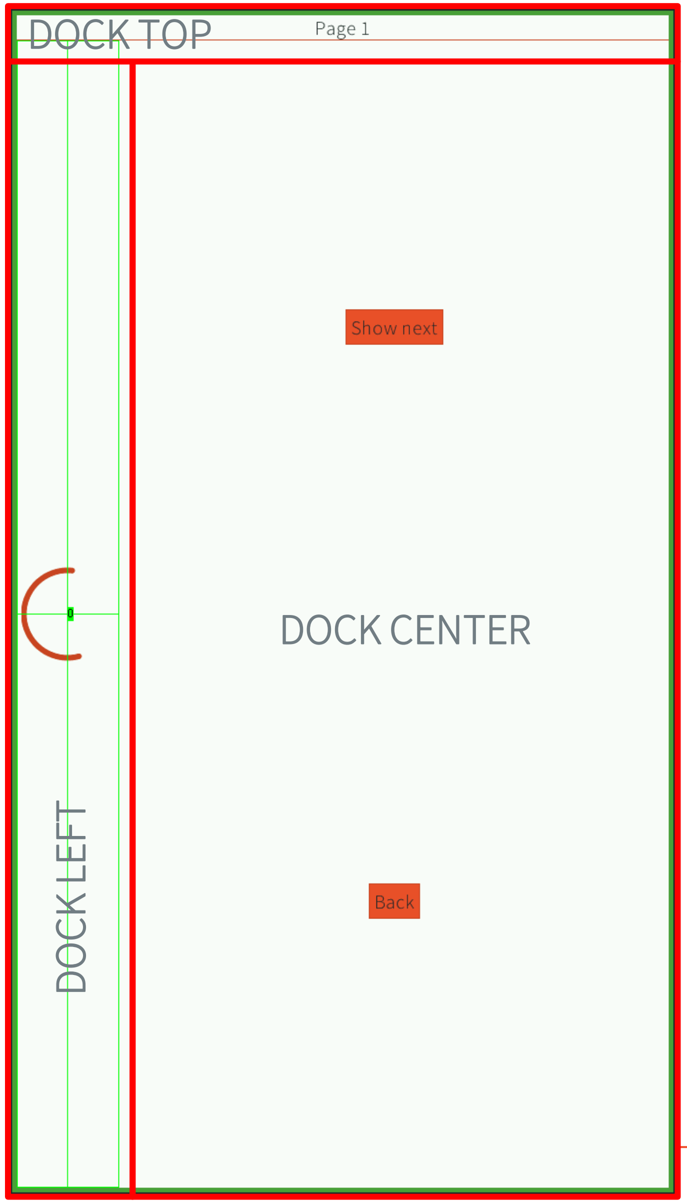
- The **CircularIndeterminateProgress** widget is included in a Dock Container (see **SlideContainerDemo** class):

```
CircularIndeterminateProgress progress = new CircularIndeterminateProgress();
dock.addChildOnLeft(progress);
```

- The style configuration of the widget is not defining any size constraint (see **SlideContainerDemo** class):

```
private static Stylesheet createStylesheet() {
  CascadingStylesheet stylesheet = new CascadingStylesheet();
  ...
  style = stylesheet.getSelectorStyle(new TypeSelector(CircularIndeterminateProgress.class));
  style.setColor(POMEGRANATE);
  style.setVerticalAlignment(Alignment.VCENTER);
  style.setPadding(new UniformOutline(PADDING_MARGIN));
```

→ **The widget fills all the space available in the left part of the Dock container**.



REDUCE THE REFRESH AREA OF THE WIDGET

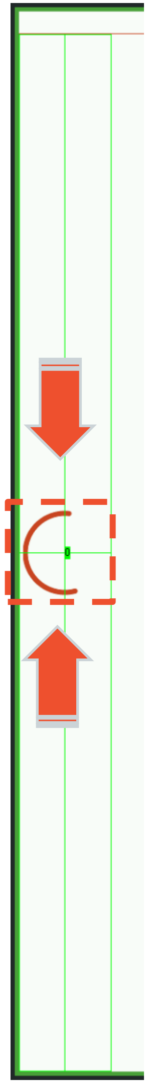
The refresh area could be reduced to fit the size of the **CircularIndeterminateProgress** widget.

This would allow to save some CPU time, avoiding useless drawing and potentially improving the fluidity of the animation.

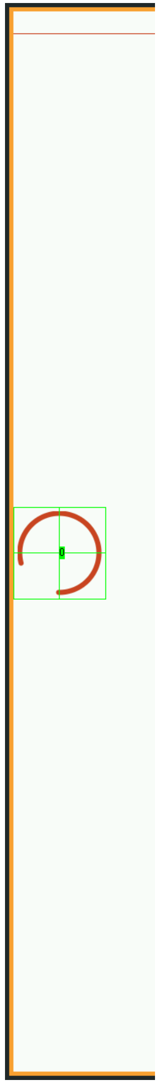
Update the style configuration of the example (*createStylesheet* method in the **SlideContainerDemo** class) to set an optimal dimension to the **CircularIndeterminateProgress** widget:

```
private static Stylesheet createStylesheet() {
   CascadingStylesheet stylesheet = new CascadingStylesheet();
    ...
   style = stylesheet.getSelectorStyle(new TypeSelector(CircularIndeterminateProgress.class));
   style.setColor(POMEGRANATE);
   style.setDimension(OptimalDimension.OPTIMAL_DIMENSION_XY);
   style.setVerticalAlignment(Alignment.VCENTER);
   style.setPadding(new UniformOutline(PADDING_MARGIN));
```

Run the updated code on Simulator to check that the refresh area has been reduced.



Before widget style update



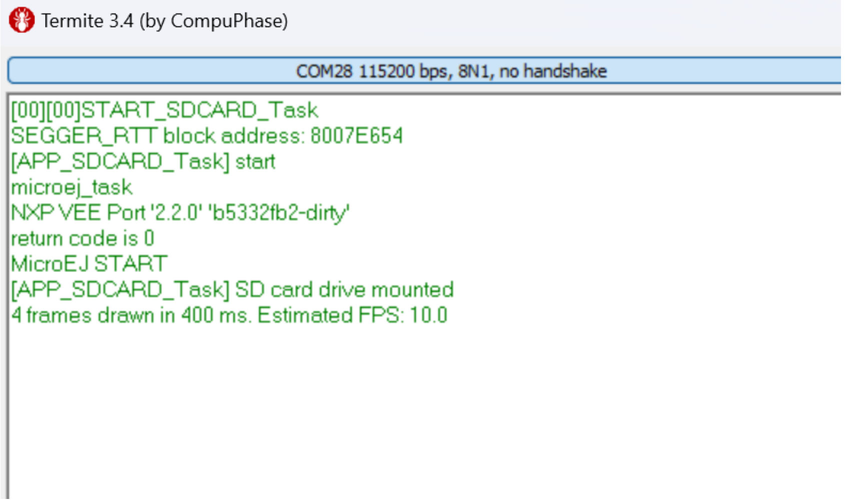
After widget style update

RUN THE APPLICATION ON THE DEVICE

Run the sample on the device.

The transition is laggy when clicking on the **Show Next** button (see **slide-container/videos/slide_containrer_nxp_rt1170_non-optimized.m4v**).

An estimated FPS count is provided in the logs:


Termite 3.4 (by CompuPhase)
COM28 115200 bps, 8N1, no handshake

```
[00][00]START_SDCARD_Task
SEGGER_RTT block address: 8007E654
[APP_SDCARD_Task] start
microej_task
NXP VEE Port '2.2.0' 'b5332fb2-dirty'
return code is 0
MicroEJ START
[APP_SDCARD_Task] SD card drive mounted
4 frames drawn in 400 ms. Estimated FPS: 10.0
```

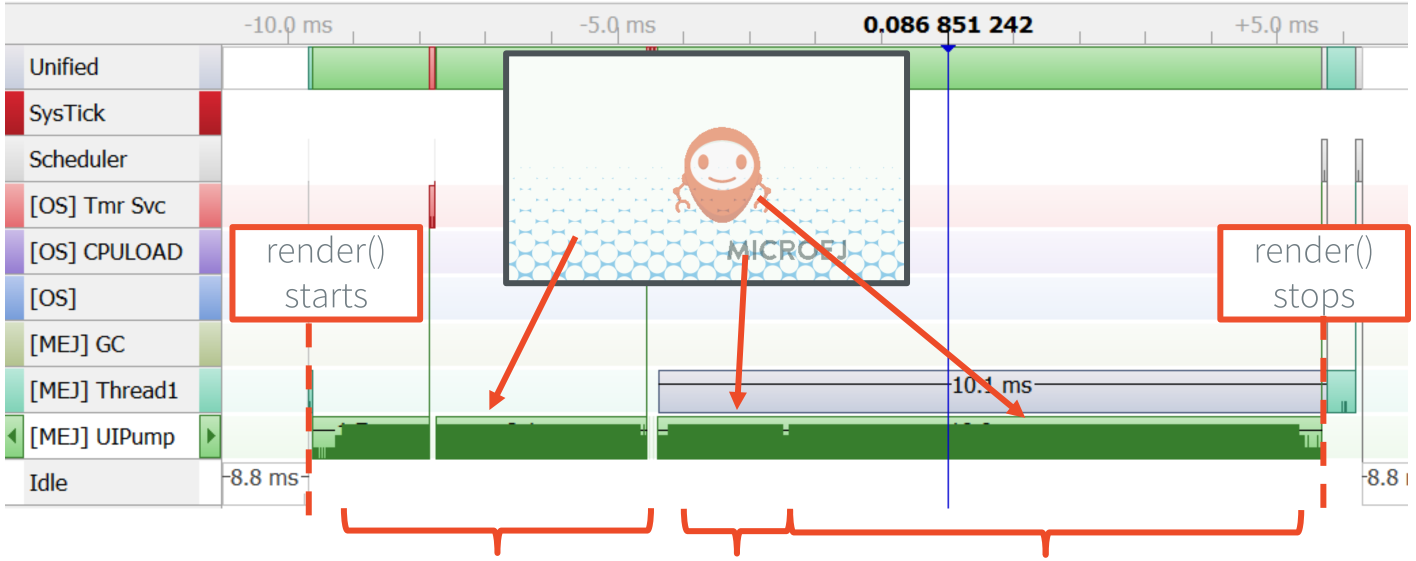
→ This board is theoretically able to run GUI applications near 60FPS. The next steps are about investigating the bottlenecks in the application and in the VEE Port that prevent having a smooth animation.

SystemView

Identify & Debug Bottlenecks on the Device using SEGGER SystemView

SYSTEMVIEW

- SystemView is a real-time recording and visualization tool for embedded systems that reveals the actual runtime behavior of an application, going far deeper than the system insights provided by debuggers.
- SystemView can ensure a system performs as designed, can track down inefficiencies, and show unintended interactions and resource conflicts, with a focus on the details of every single system tick.
- A specific SystemView extension made by MicroEJ allows to trace the OS tasks and the MicroEJ Java threads at the same time.
- For example, it can be used to measure the rendering time of images in a GUI application:



render() starts

render() stops

Background: 4.5ms

Logo: 1.7ms

Mascot: 7.7ms

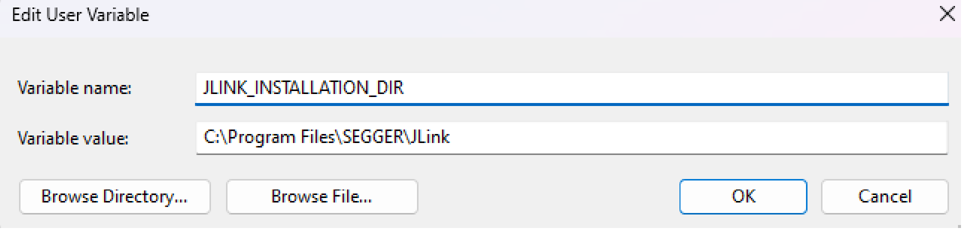
```
public void render(GraphicsContext gc) {
    drawTracer.recordEvent(MY_EVT_ID);
    int length = images.size();
    for(int i = -1; ++i<length;) {
        images.get(i).paint(gc);
    }
    drawTracer.recordEventEnd(MY_EVT_ID);
}
```

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

SETUP THE ENVIRONMENT FOR SYSTEMVIEW (1/2)

The following software are required:

- Install SEGGER J-Link:
  - Create the **JLINK_INSTALLATION_DIR** environment variable that points to the SEGGER J-Link installation directory (e.g. C:\Program Files\SEGGER\Jlink):



Edit User Variable

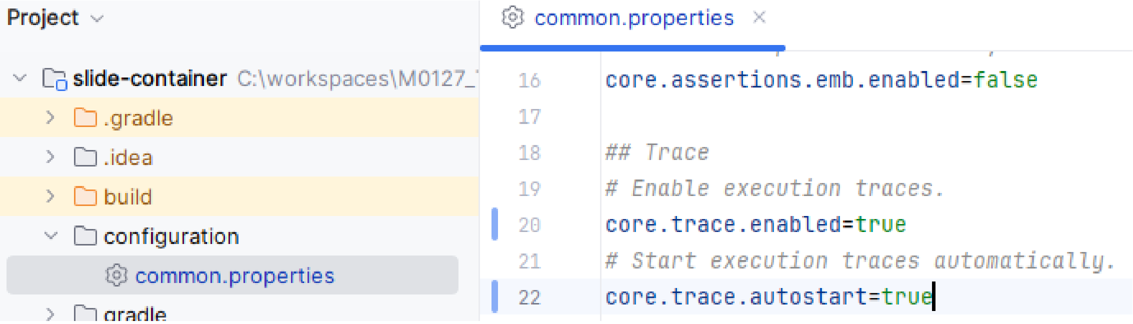
Variable name: JLINK_INSTALLATION_DIR

Variable value: C:\Program Files\SEGGER\JLink

- Install SEGGER SystemView

The runtime traces of the application needs to be enabled to see MicroEJ VEE tasks activity in SystemView:

- Open the configuration file of the application project (e.g. common.properties):
- Set **core.trace.enabled** and **core.trace.autostart** to **true**

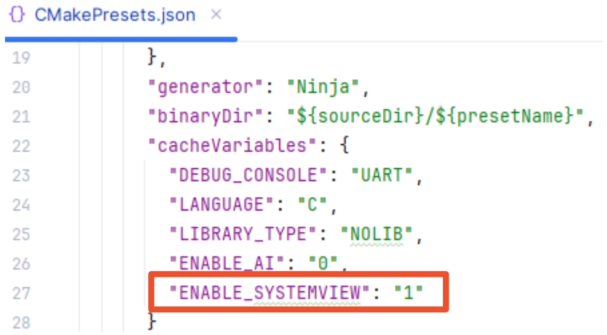


```
core.assertions.emb.enabled=false

## Trace
# Enable execution traces.
core.trace.enabled=true
# Start execution traces automatically.
core.trace.autostart=true
```

SETUP THE ENVIRONMENT FOR SYSTEMVIEW (2/2)

Enable SystemView in the VEE Port:

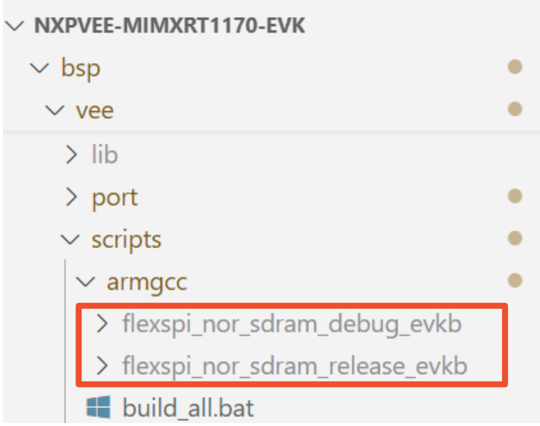
- Open **CMakePresets.json** file located in **nxp-vee-mimxrt1170-evk/bsp/vee/scripts/armgcc/CMakePresets.json**
- Set **ENABLE_SYSTEM_VIEW** to **1**:



CMakePresets.json ×

```
        },
        "generator": "Ninja",
        "binaryDir": "${sourceDir}/${presetName}",
        "cacheVariables": {
          "DEBUG_CONSOLE": "UART",
          "LANGUAGE": "C",
          "LIBRARY_TYPE": "NOLIB",
          "ENABLE_AI": "0",
          "ENABLE_SYSTEMVIEW": "1"
        }
```

Remove the build folders of the BSP to ensure the **ENABLE_SYSTEM_VIEW** property to be taken into account during next build (nxp-vee-mimxrt1170-evk/bsp/vee/scripts/armgcc/):



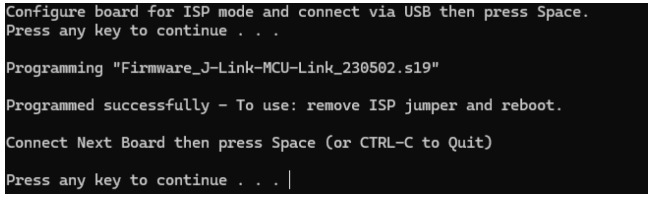
```
NXPVEE-MIMXRT1170-EVK
  bsp
    vee
      lib
      port
      scripts
        armgcc
          flexspi_nor_sdram_debug_evkb
          flexspi_nor_sdram_release_evkb
          build_all.bat
```

USE A J-LINK PROBE

SystemView is a SEGGER tool, consequently it requires a J-Link probe to be used.

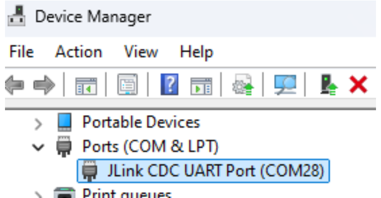
There are 2 ways to use a J-Link probe with the NXP i.MX RT1170 EVK:

1. **Option 1:** Connect an external J-Link probe:
   - Connect jumper JP5
   - Connect the probe to the J1 Connector
2. **Option 2:** Reprogram the embedded probe of the EVK (MCU-Link):
   - Unplug J86 and J43 connectors. Plug the J43 connector, **then** the J86.
   - Make sure jumper JP5 is removed and the USB cable is connected to J86
   - Run the following script: **C:\nxp\MCU-LINK_installer_{version}\scripts\program_JLINK.cmd**
   - Turn OFF the EVK + unplug the J86 USB cable, then Connect jumper JP3 and turn the board back ON + plug J86
   - Press **SPACE**

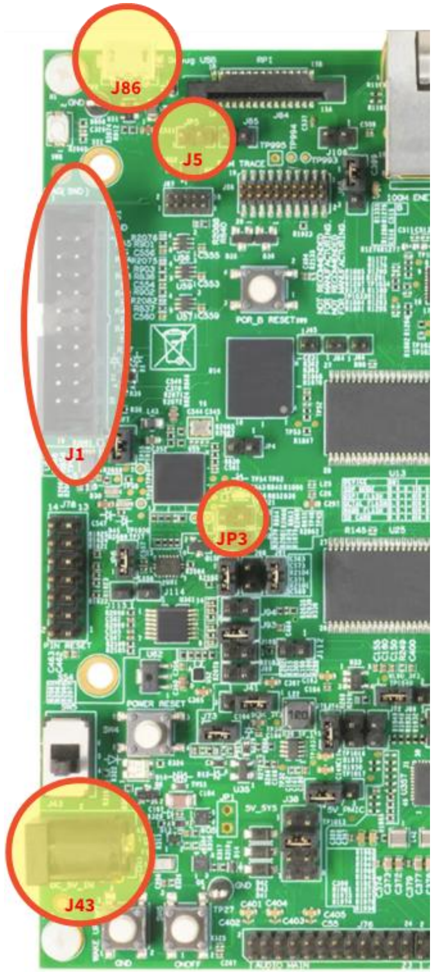


```
Configure board for ISP mode and connect via USB then press Space.
Press any key to continue . . .

Programming "Firmware_J-Link-MCU-Link_230502.s19"

Programmed successfully - To use: remove ISP jumper and reboot.

Connect Next Board then press Space (or CTRL-C to Quit)

Press any key to continue . . . |
```

   - Remove the JP3 jumper
   - Turn OFF and ON the EVK + unplug / plug the J86 USB cable.
   - The probe can be seen in the device manager:

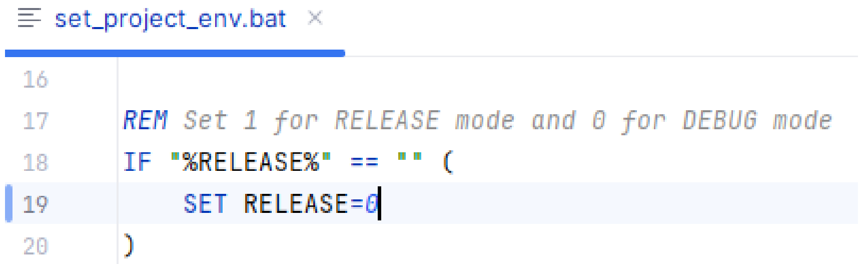


Portable Devices
Ports (COM & LPT)
JLink CDC UART Port (COM28)
Print queues



UPDATE THE BUILD SCRIPT CONFIGURATION

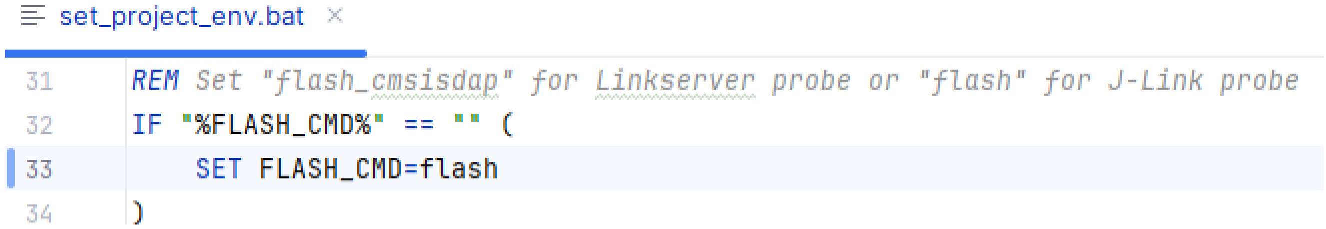
1. If not already done, enable the DEBUG mode by setting **RELEASE=0** in **nxpvee-mimxrt1170-evk\bsp\vee\scripts\set_project_env.bat:**



set_project_env.bat

```
REM Set 1 for RELEASE mode and 0 for DEBUG mode
IF "%RELEASE%" == "" (
    SET RELEASE=0
)
```

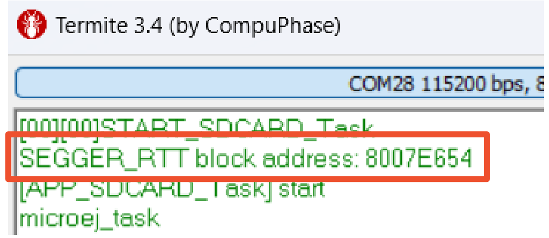
2. Update the **FLASH_CMD** variable to use a J-Link probe:



set_project_env.bat

```
REM Set "flash_cmsisdap" for Linkserver probe or "flash" for J-Link probe
IF "%FLASH_CMD%" == "" (
    SET FLASH_CMD=flash
)
```

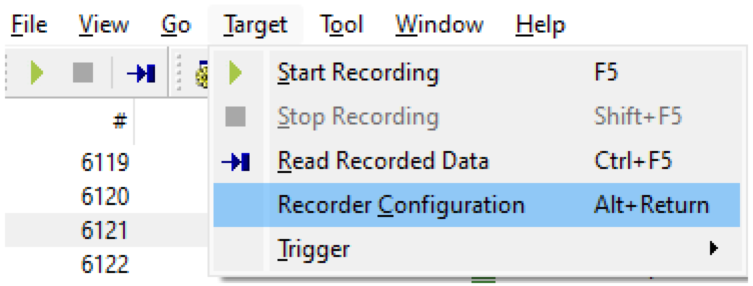
Once done, build and flash the application on the device using the **runOnDevice** task.
Once done, the following trace should appear in the console:

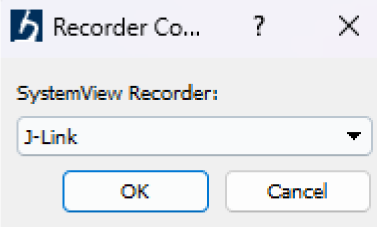


```
[00][00]START_SDCARD_Task
SEGGER_RTT block address: 8007E654
[APP_SDCARD_Task] start
microej_task
```

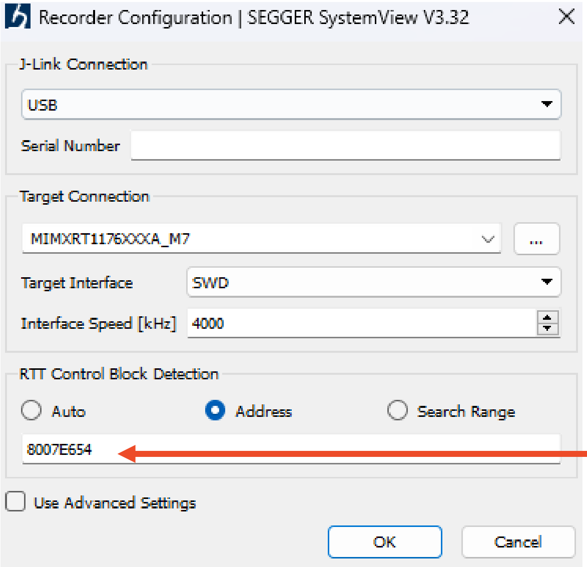
START THE ACQUISITION ON SYSTEMVIEW

- Start SystemView (tested with version 3.32)
- Set the following recorder configuration:





SystemView Recorder: J-Link



J-Link Connection: USB

Serial Number:

Target Connection: MIMXRT1176XXXA_M7

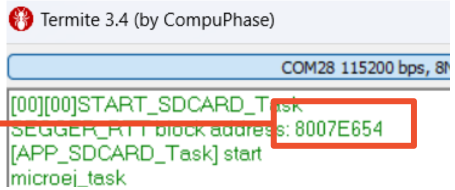
Target Interface: SWD

Interface Speed [kHz]: 4000

RTT Control Block Detection: ○ Auto ● Address ○ Search Range

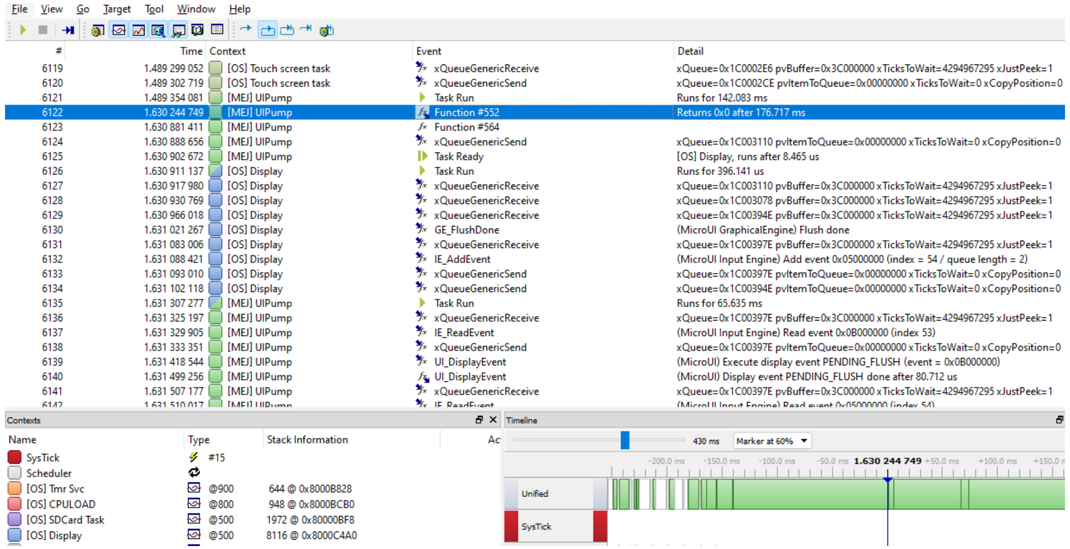
8007E654

□ Use Advanced Settings



```
[00][00]START_SDCARD_Task
SEGGER_RTT block address: 8007E654
[APP_SDCARD_Task] start
microej_task
```

- Once done, click on the **Play** button. The acquisition starts, events can be see in SystemView:



#	Time	Context	Event	Detail
6119	1.489 299 052	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C0002E6 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6120	1.489 302 719	[OS] Touch screen task	xQueueGenericSend	xQueue=0x1C0002CE pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
6121	1.489 354 081	[MEJ] UIPump	Task Run	Runs for 142.083 ms
6122	1.630 244 749	[MEJ] UIPump	Function #552	Returns 0x0 after 176.717 ms
6123	1.630 881 411	[MEJ] UIPump	Function #564	
6124	1.630 888 656	[MEJ] UIPump	xQueueGenericSend	xQueue=0x1C003110 pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
6125	1.630 902 672	[MEJ] UIPump	Task Ready	[OS] Display, runs after 8.465 us
6126	1.630 911 137	[OS] Display	Task Run	Runs for 396.141 us
6127	1.630 917 980	[OS] Display	xQueueGenericReceive	xQueue=0x1C003110 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6128	1.630 930 769	[OS] Display	xQueueGenericReceive	xQueue=0x1C003078 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6129	1.630 966 018	[OS] Display	xQueueGenericReceive	xQueue=0x1C00394E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6130	1.631 021 267	[OS] Display	GE_FlushDone	(MicroUI GraphicalEngine) Flush done
6131	1.631 083 006	[OS] Display	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6132	1.631 088 421	[OS] Display	IE_AddEvent	(MicroUI Input Engine) Add event 0x05000000 (index = 54 / queue length = 2)
6133	1.631 093 010	[OS] Display	xQueueGenericSend	xQueue=0x1C00397E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
6134	1.631 102 118	[OS] Display	xQueueGenericSend	xQueue=0x1C00394E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
6135	1.631 307 277	[MEJ] UIPump	Task Run	Runs for 65.635 ms
6136	1.631 325 197	[MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6137	1.631 329 905	[MEJ] UIPump	IE_ReadEvent	(MicroUI Input Engine) Read event 0x0B000000 (index 53)
6138	1.631 333 351	[MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00397E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
6139	1.631 418 544	[MEJ] UIPump	UI_DisplayEvent	(MicroUI) Execute display event PENDING_FLUSH (event = 0x0B000000)
6140	1.631 499 256	[MEJ] UIPump	UI_DisplayEvent	(MicroUI) Display event PENDING_FLUSH done after 80.712 us
6141	1.631 507 177	[MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
6142	1.631 510 017	[MEJ] UIPump	IE_ReadEvent	(MicroUI Input Engine) Read event 0x05000000 (index 54)

Contexts

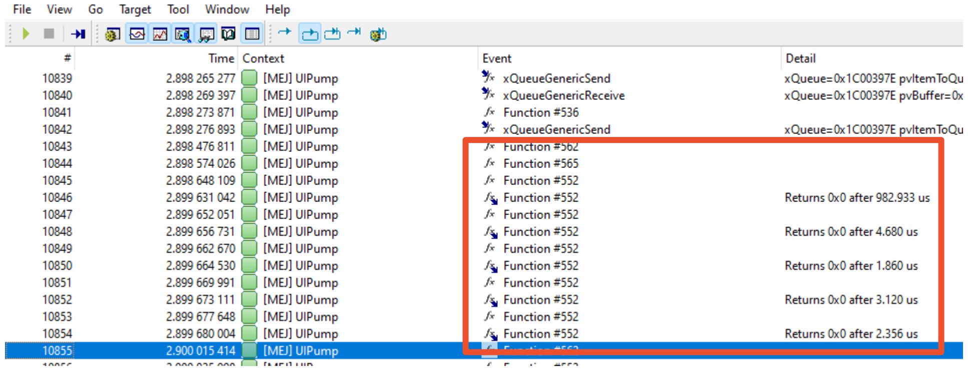
Name	Type	Stack Information	Ac
SysTick	#15		
Scheduler			
[OS] Tmr Svc	@900	644 @ 0x8000B828	
[OS] CPULOAD	@800	948 @ 0x8000BCB0	
[OS] SDCard Task	@500	1972 @ 0x80000BF8	
[OS] Display	@500	8116 @ 0x8000C4A0	

GET MICROUI DEBUG TRACES

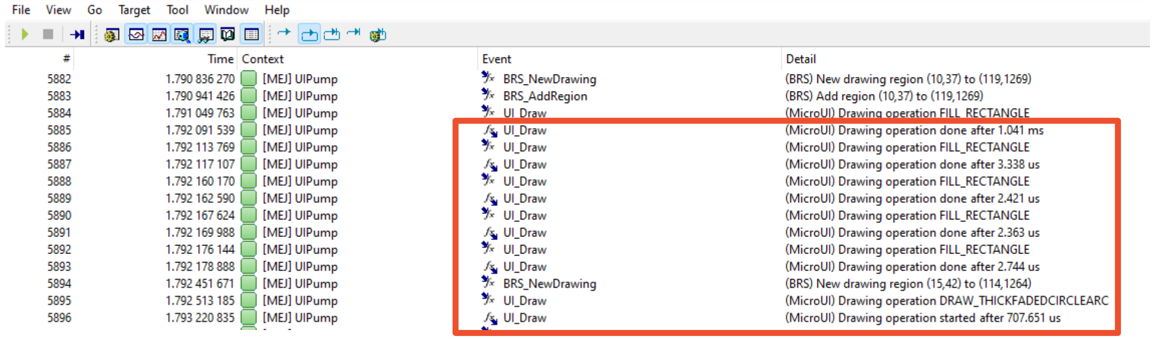
MicroUI logs several actions when traces are enabled (see Event Tracing).

Those traces can be added in SystemView to ease the analysis:

- Copy the traces provided in the following section: https://docs.microej.com/en/latest/ApplicationDeveloperGuide/UI/MicroUI/traces.html#systemview-integration
    - **Warning**: jump a line at the end of the line for it to be taken into account
- Save them in a **SYSVIEW_MicroUI.txt** file (the syntax of the file matters) in **C:\Program Files\SEGGER\SystemView\Description**
- Restart SystemView and start a new acquisition
- The UI events are now detailed:



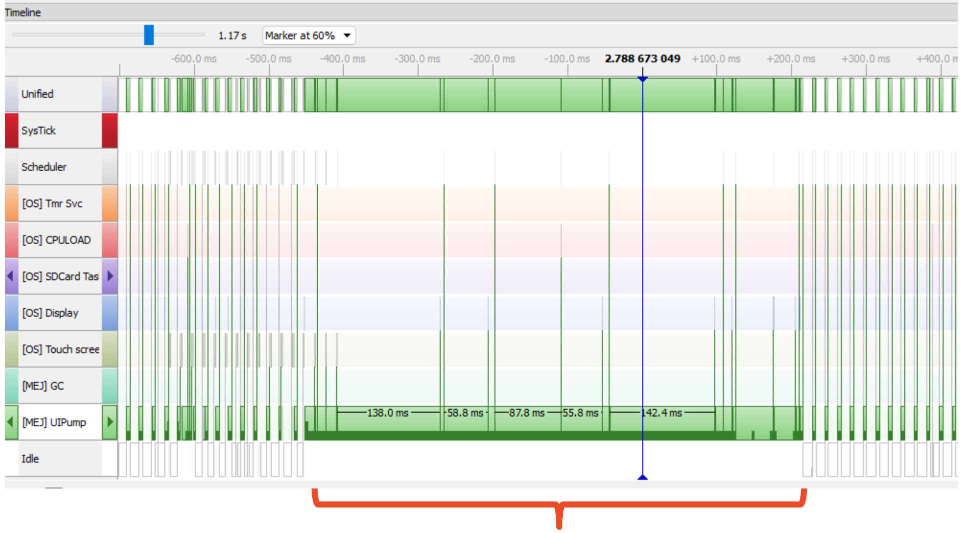
#	Time	Context	Event	Detail
10839	2.898 265 277	[MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00397E pvItemToQu
10840	2.898 269 397	[MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x
10841	2.898 273 871	[MEJ] UIPump	Function #536	
10842	2.898 276 893	[MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00397E pvItemToQu
10843	2.898 476 811	[MEJ] UIPump	Function #562	
10844	2.898 574 026	[MEJ] UIPump	Function #565	
10845	2.898 648 109	[MEJ] UIPump	Function #552	
10846	2.899 631 042	[MEJ] UIPump	Function #552	Returns 0x0 after 982.933 us
10847	2.899 652 051	[MEJ] UIPump	Function #552	
10848	2.899 656 731	[MEJ] UIPump	Function #552	Returns 0x0 after 4.680 us
10849	2.899 662 670	[MEJ] UIPump	Function #552	
10850	2.899 664 530	[MEJ] UIPump	Function #552	Returns 0x0 after 1.860 us
10851	2.899 669 991	[MEJ] UIPump	Function #552	
10852	2.899 673 111	[MEJ] UIPump	Function #552	Returns 0x0 after 3.120 us
10853	2.899 677 648	[MEJ] UIPump	Function #552	
10854	2.899 680 004	[MEJ] UIPump	Function #552	Returns 0x0 after 2.356 us
10855	2.900 015 414	[MEJ] UIPump	Function #552	



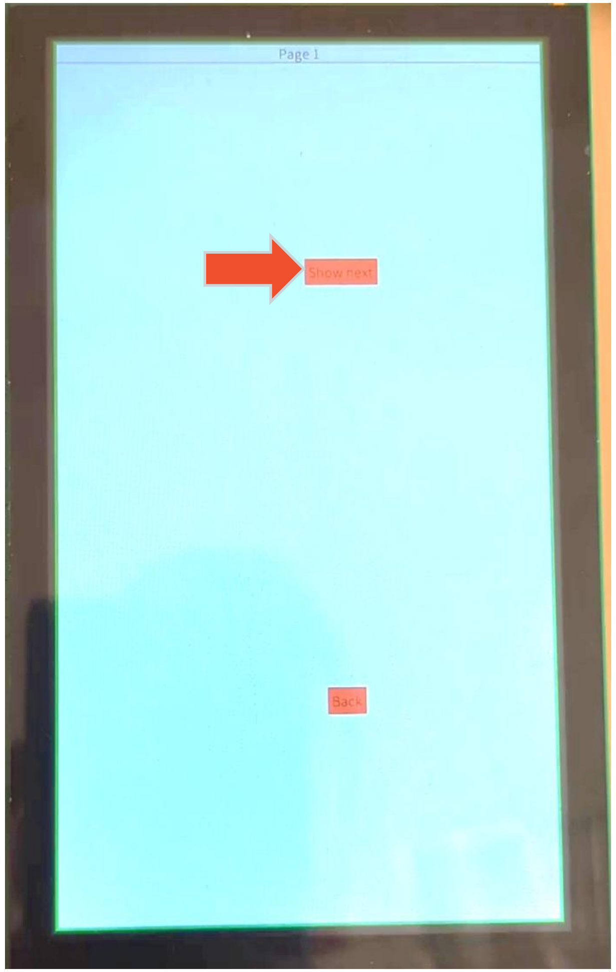
#	Time	Context	Event	Detail
5882	1.790 836 270	[MEJ] UIPump	BRS_NewDrawing	(BRS) New drawing region (10,37) to (119,1269)
5883	1.790 941 426	[MEJ] UIPump	BRS_AddRegion	(BRS) Add region (10,37) to (119,1269)
5884	1.791 049 763	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation FILL_RECTANGLE
5885	1.792 091 539	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation done after 1.041 ms
5886	1.792 113 769	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation FILL_RECTANGLE
5887	1.792 117 107	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation done after 3.338 us
5888	1.792 160 170	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation FILL_RECTANGLE
5889	1.792 162 590	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation done after 2.421 us
5890	1.792 167 624	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation FILL_RECTANGLE
5891	1.792 169 988	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation done after 2.363 us
5892	1.792 176 144	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation FILL_RECTANGLE
5893	1.792 178 888	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation done after 2.744 us
5894	1.792 451 671	[MEJ] UIPump	BRS_NewDrawing	(BRS) New drawing region (15,42) to (114,1264)
5895	1.792 513 185	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation DRAW_THICKFADEDCIRCLEARC
5896	1.793 220 835	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation started after 707.651 us

TRACE ANALYSIS

- Start a new acquisition in SystemView.
- Press the **Show next** button.
- Stop the acquisition.
- The following diagram can be seen:



Slide transition



- At a first glimpse, we can see that the CPU is fully loaded during the slide transition. There are no “Idle” events, all the time is spent in the UIPump thread.

ADD A CUSTOM TRACE (1/2)

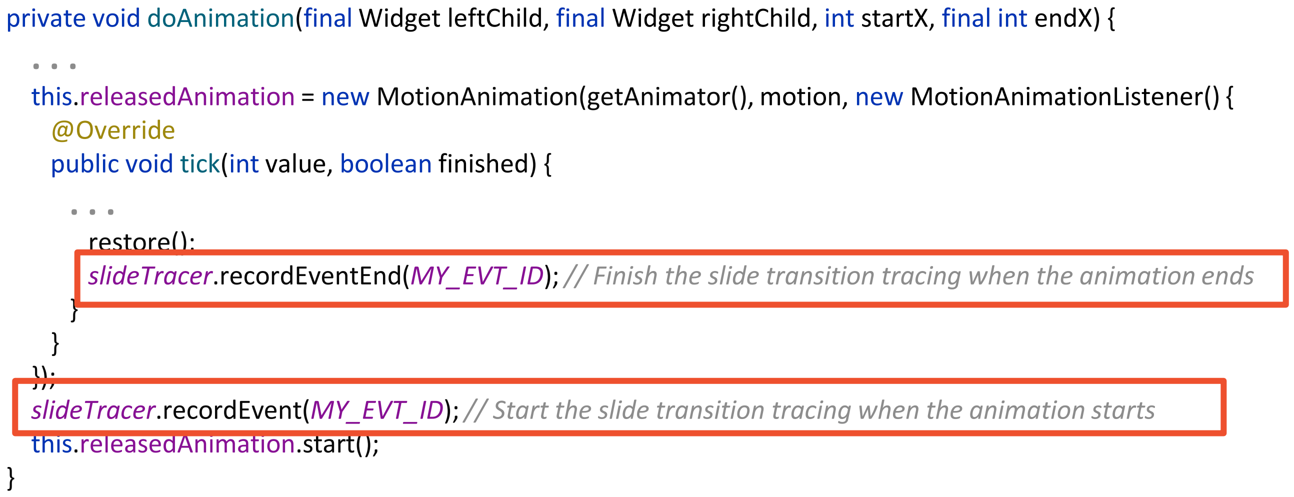
- Custom traces can be added in SystemView to figure out which events are occurring during the slide transition.
- Follow the steps below to add a custom trace:
  - Add Trace library in the **build.gradle.kts**:

    ```
    implementation("ej.api:trace:1.1.1")
    ```

  - **Reload the Gradle project** to get the dependency. Otherwise, the wrong TRACE library might be imported (sun.java2d.windows.GDIRenderer)
  - Create a new Tracer in the **SlideContainer** class:

    ```
    private static final Tracer slideTracer = new Tracer("MyTracer", 1); // The Tracer object
    private static final int MY_EVT_ID = 0; // The ID that will be used to track the slide transition event
    ```

  - Update the **doAnimation** method to track the start and the end of the transition:



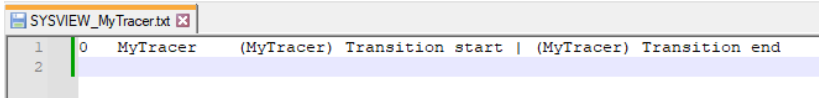
```
private void doAnimation(final Widget leftChild, final Widget rightChild, int startX, final int endX) {
  . . .
  this.releasedAnimation = new MotionAnimation(getAnimator(), motion, new MotionAnimationListener() {
    @Override
    public void tick(int value, boolean finished) {
      . . .
        restore();
        slideTracer.recordEventEnd(MY_EVT_ID); // Finish the slide transition tracing when the animation ends
      }
    }
  });
  slideTracer.recordEvent(MY_EVT_ID); // Start the slide transition tracing when the animation starts
  this.releasedAnimation.start();
}
```

ADD A CUSTOM TRACE (2/2)

- The custom **MyTracer** event needs to be added to SystemView description files in order to be taken into account during the analysis.
- Create a **SYSVIEW_MyTracer.txt** file in **C:\Program Files\SEGGER\SystemView\Description**
- Add the following content in it:

```
0        MyTracer (MyTracer) Transition start | (MyTracer) Transition end
```

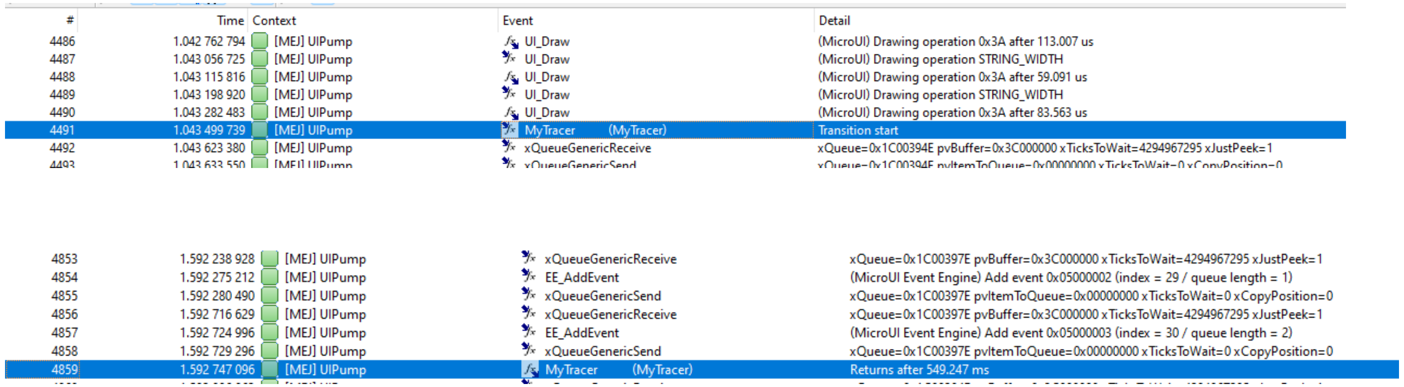
  - ○ **Warning**: jump a line at the end of the line for it to be taken into account:



SYSVIEW_MyTracer.txt

```
0   MyTracer     (MyTracer) Transition start | (MyTracer) Transition end

```

- Restart SystemView and start a new acquisition + press the **Show next** button in the application.
- The **MyTracer** events can be seen in the events view:



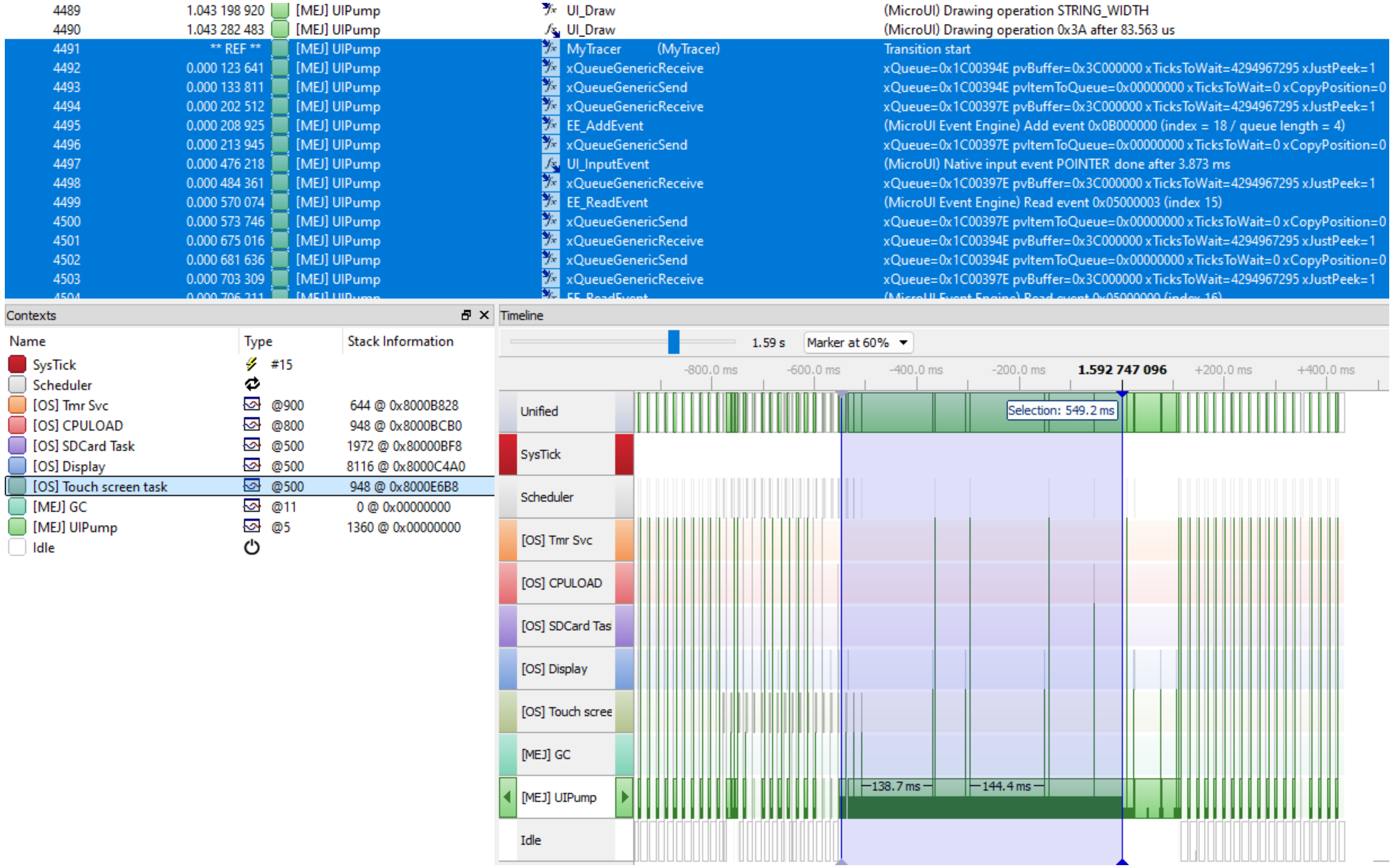
#	Time	Context	Event	Detail
4486	1.042 762 794	🟩 [MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation 0x3A after 113.007 us
4487	1.043 056 725	🟩 [MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation STRING_WIDTH
4488	1.043 115 816	🟩 [MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation 0x3A after 59.091 us
4489	1.043 198 920	🟩 [MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation STRING_WIDTH
4490	1.043 282 483	🟩 [MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation 0x3A after 83.563 us
4491	1.043 499 739	🟩 [MEJ] UIPump	MyTracer (MyTracer)	Transition start
4492	1.043 623 380	🟩 [MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00394E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4493	1.043 633 550	🟩 [MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00394E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0

#	Time	Context	Event	Detail
4853	1.592 238 928	🟩 [MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4854	1.592 275 212	🟩 [MEJ] UIPump	EE_AddEvent	(MicroUI Event Engine) Add event 0x05000002 (index = 29 / queue length = 1)
4855	1.592 280 490	🟩 [MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00397E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4856	1.592 716 629	🟩 [MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4857	1.592 724 996	🟩 [MEJ] UIPump	EE_AddEvent	(MicroUI Event Engine) Add event 0x05000003 (index = 30 / queue length = 2)
4858	1.592 729 296	🟩 [MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00397E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4859	1.592 747 096	🟩 [MEJ] UIPump	MyTracer (MyTracer)	Returns after 549.247 ms

ANALYSIS (1/6)

TIMELINE OVERVIEW

The following events are occurring between MyTracer start and end:



```
4489     1.043 198 920  [MEJ] UIPump   UI_Draw                              (MicroUI) Drawing operation STRING_WIDTH
4490     1.043 282 483  [MEJ] UIPump   UI_Draw                              (MicroUI) Drawing operation 0x3A after 83.563 us
4491          ** REF **  [MEJ] UIPump   MyTracer       (MyTracer)            Transition start
4492     0.000 123 641  [MEJ] UIPump   xQueueGenericReceive                 xQueue=0x1C00394E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4493     0.000 133 811  [MEJ] UIPump   xQueueGenericSend                    xQueue=0x1C00394E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4494     0.000 202 512  [MEJ] UIPump   xQueueGenericReceive                 xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4495     0.000 208 925  [MEJ] UIPump   EE_AddEvent                          (MicroUI Event Engine) Add event 0x0B000000 (index = 18 / queue length = 4)
4496     0.000 213 945  [MEJ] UIPump   xQueueGenericSend                    xQueue=0x1C00397E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4497     0.000 476 218  [MEJ] UIPump   UI_InputEvent                        (MicroUI) Native input event POINTER done after 3.873 ms
4498     0.000 484 361  [MEJ] UIPump   xQueueGenericReceive                 xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4499     0.000 570 074  [MEJ] UIPump   EE_ReadEvent                         (MicroUI Event Engine) Read event 0x05000003 (index 15)
4500     0.000 573 746  [MEJ] UIPump   xQueueGenericSend                    xQueue=0x1C00397E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4501     0.000 675 016  [MEJ] UIPump   xQueueGenericReceive                 xQueue=0x1C00394E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4502     0.000 681 636  [MEJ] UIPump   xQueueGenericSend                    xQueue=0x1C00394E pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4503     0.000 703 309  [MEJ] UIPump   xQueueGenericReceive                 xQueue=0x1C00397E pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPeek=1
4504     0.000 706 211  [MEJ] UIPump   EE_ReadEvent                         (MicroUI Event Engine) Read event 0x05000000 (index 16)
```

Contexts

Name	Type	Stack Information
SysTick	#15	
Scheduler		
[OS] Tmr Svc	@900	644 @ 0x8000B828
[OS] CPULOAD	@800	948 @ 0x8000BCB0
[OS] SDCard Task	@500	1972 @ 0x80000BF8
[OS] Display	@500	8116 @ 0x8000C4A0
[OS] Touch screen task	@500	948 @ 0x8000E6B8
[MEJ] GC	@11	0 @ 0x00000000
[MEJ] UIPump	@5	1360 @ 0x00000000
Idle		

What we can observe:

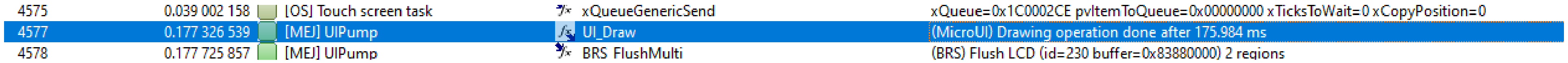
- The transition is during approximately 550ms → ideally it should last 400ms see (**TRANSITION_DURATION** in **SlideContainer** class)
- There are big "blocks" in the timeline (138.7ms, 144.4ms)

→ The next slide will provide a way to interpret the results

ANALYSIS (2/6)

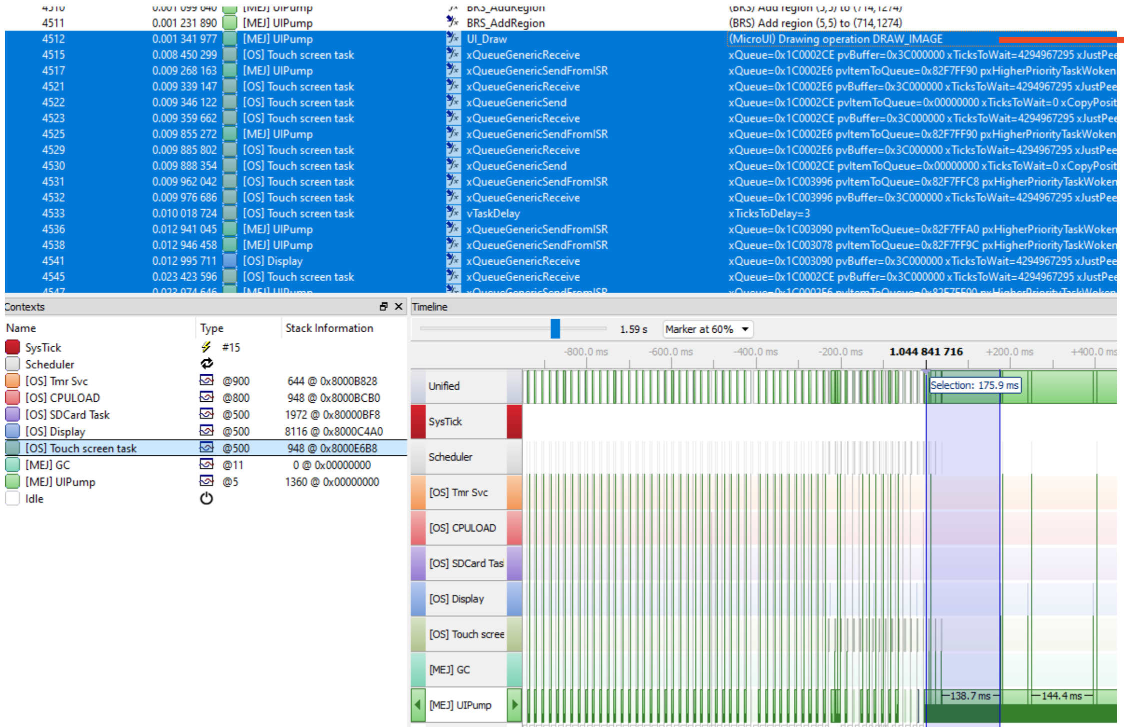
LOOK FOR TIME CONSUMING OPERATIONS

- Go down in the events list, between **MyTracer start** and **MyTracer end**
- Look for the big operations, for example this drawing operation that took 175.9ms:



4575	0.039 002 158	[OS] Touch screen task	xQueueGenericSend	xQueue=0x1C0002CE pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosition=0
4577	0.177 326 539	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation done after 175.984 ms
4578	0.177 725 857	[MEJ] UIPump	BRS_FlushMulti	(BRS) Flush LCD (id=230 buffer=0x83880000) 2 regions

- Once identified, scroll up 175.9ms earlier to see what was the nature of the drawing operation:



4510	0.001 099 040	[MEJ] UIPump	BRS_AddRegion	(BRS) Add region (5,5) to (714,1274)
4511	0.001 231 890	[MEJ] UIPump	BRS_AddRegion	(BRS) Add region (5,5) to (714,1274)
4512	0.001 341 977	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation DRAW_IMAGE
4515	0.008 450 299	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C0002CE pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4517	0.009 268 163	[MEJ] UIPump	xQueueGenericSendFromISR	xQueue=0x1C0002E6 pvItemToQueue=0x82F7FF90 pxHigherPriorityTaskWoken
4521	0.009 339 147	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C0002E6 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4522	0.009 346 122	[OS] Touch screen task	xQueueGenericSend	xQueue=0x1C0002CE pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosit
4523	0.009 359 662	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C0002CE pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4525	0.009 855 272	[MEJ] UIPump	xQueueGenericSendFromISR	xQueue=0x1C0002E6 pvItemToQueue=0x82F7FF90 pxHigherPriorityTaskWoken
4529	0.009 885 802	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C0002E6 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4530	0.009 888 354	[OS] Touch screen task	xQueueGenericSend	xQueue=0x1C0002CE pvItemToQueue=0x00000000 xTicksToWait=0 xCopyPosit
4531	0.009 962 042	[OS] Touch screen task	xQueueGenericSendFromISR	xQueue=0x1C003996 pvItemToQueue=0x82F7FFC8 pxHigherPriorityTaskWoken
4532	0.009 976 686	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C003996 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4533	0.010 018 724	[OS] Touch screen task	vTaskDelay	xTicksToDelay=3
4536	0.012 941 045	[MEJ] UIPump	xQueueGenericSendFromISR	xQueue=0x1C003090 pvItemToQueue=0x82F7FFA0 pxHigherPriorityTaskWoken
4538	0.012 946 458	[MEJ] UIPump	xQueueGenericSendFromISR	xQueue=0x1C003078 pvItemToQueue=0x82F7FF9C pxHigherPriorityTaskWoken
4541	0.012 995 711	[OS] Display	xQueueGenericReceive	xQueue=0x1C003090 pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4545	0.023 423 596	[OS] Touch screen task	xQueueGenericReceive	xQueue=0x1C0002CE pvBuffer=0x3C000000 xTicksToWait=4294967295 xJustPee
4547	0.023 974 646	[MEJ] UIPump	xQueueGenericSendFromISR	xQueue=0x1C0002E6 pvItemToQueue=0x82F7FF90 pxHigherPriorityTaskWoken

Contexts

Name	Type	Stack Information
SysTick	#15	
Scheduler		
[OS] Tmr Svc	@900	644 @ 0x8000B828
[OS] CPULOAD	@800	948 @ 0x8000BCB0
[OS] SDCard Task	@500	1972 @ 0x80000BF8
[OS] Display	@500	8116 @ 0x8000C4A0
[OS] Touch screen task	@500	948 @ 0x8000E6B8
[MEJ] GC	@11	0 @ 0x00000000
[MEJ] UIPump	@5	1360 @ 0x00000000
Idle		

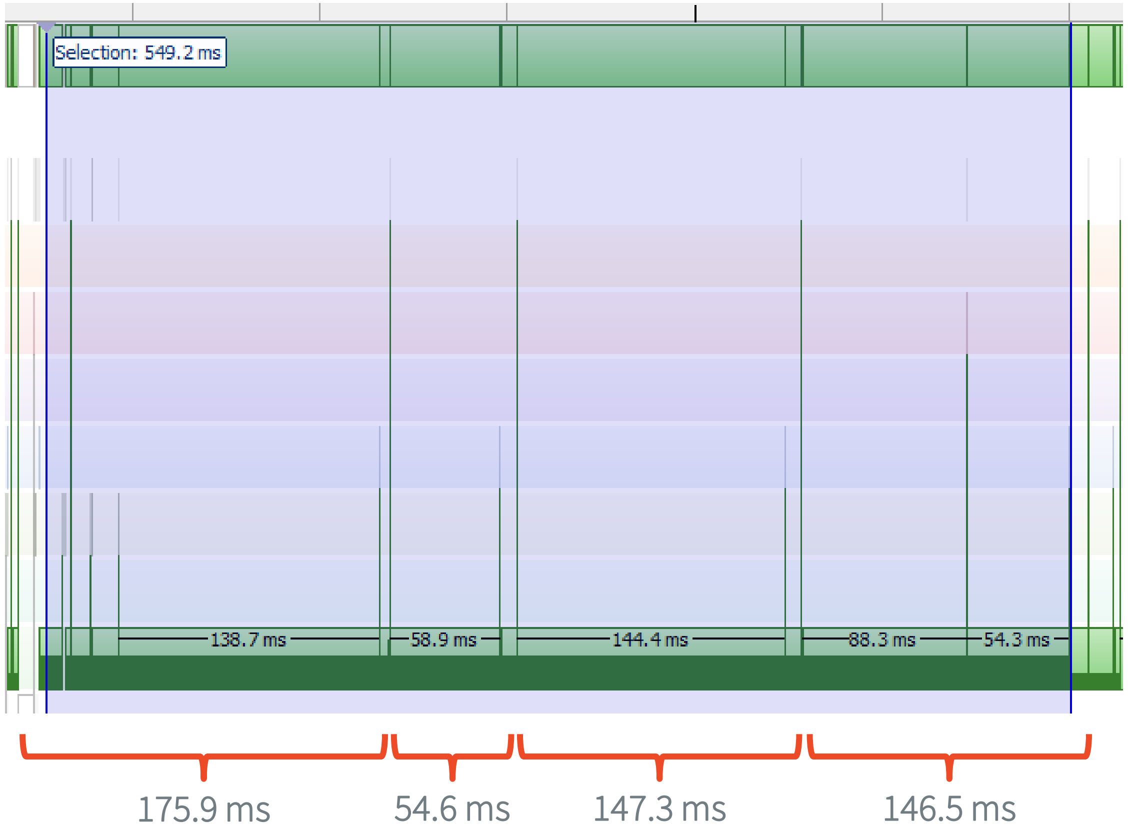
This is a **drawImage** operation

→ Locate all the other “time consuming” operations until **MyTracer end**

ANALYSIS (3/6)

DRAWIMAGE OPERATIONS ARE THE MAIN BOTTLENECK

Most of the time-consuming operations are related to **drawImage** operations:



drawImage operations are taking 95% of the slide transition time.

We can see that there are 4 drawImage operations performed. It is related to the 4 FPS observed in the console logs.

The next steps are:

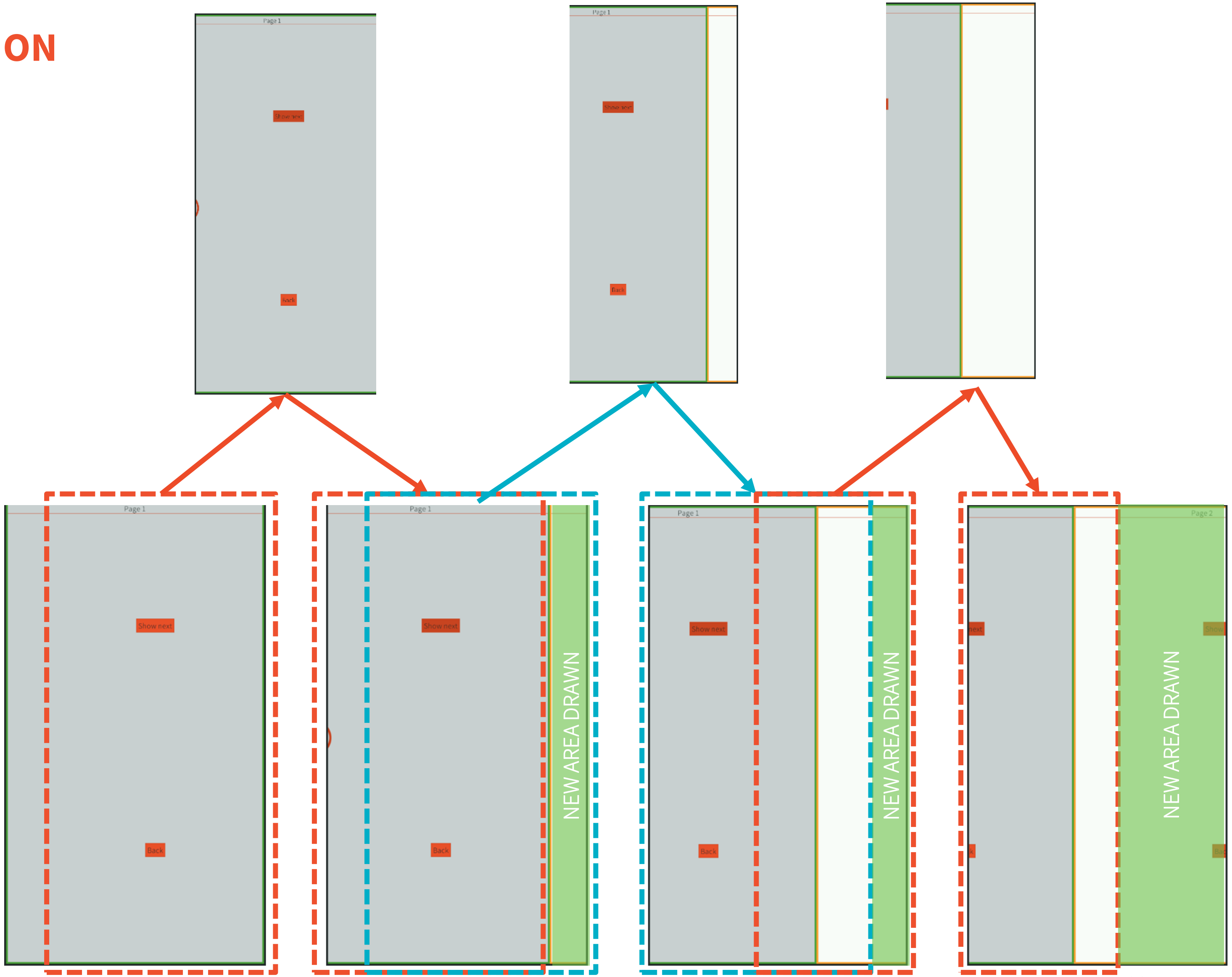
1. Check the application implementation to understand why / how **drawImage** operations are done.
2. Check that the **drawImage** LLAPI is properly implemented in the BSP (use hardware accelerator, front buffer located in a high speed memory, memory cache enabled, …)

ANALYSIS (4/6)

ABOUT THE SLIDE CONTAINER IMPLEMENTATION

The slide container sample is optimized for low CPU usage:

- Drawings are done as less as possible during the transition (from right to left).
- For each frame:
  - The content previously drawn on the screen is reused in the next frame on the left part of the screen (like a screenshot).
  - Only the right part of the screen is drawn (Green area).
- The screen content can be reused using the **drawDisplayRegion** API.
  It allows to copy a part of the screen on itself.
- **The drawDisplayRegion implementation is calling the drawImage API. This confirms the drawImage events seen in SystemView.**



ANALYSIS (5/6)

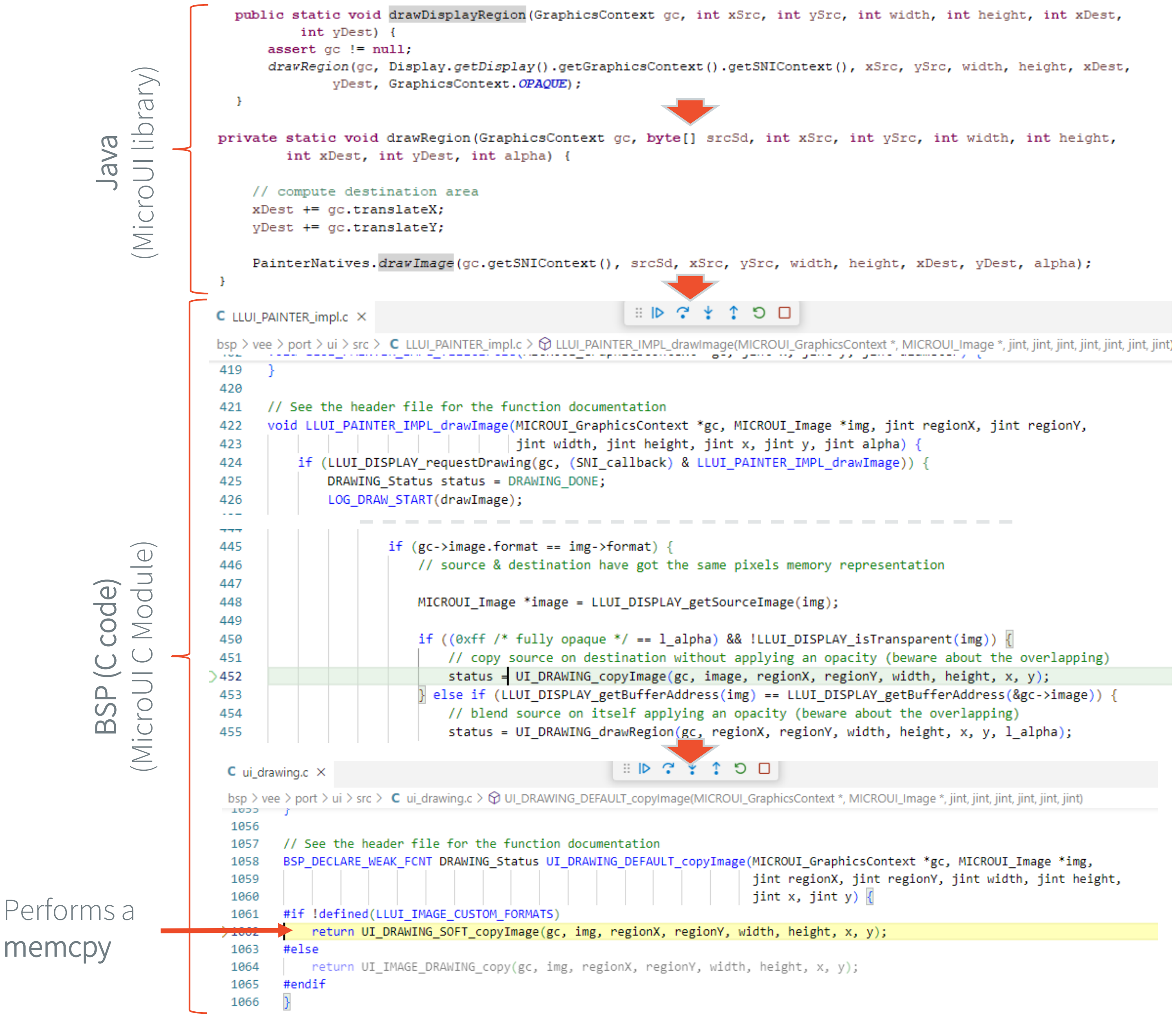
ABOUT THE DRAWIMAGE IMPLEMENTATION

The **drawImage** method is implemented in the **LLUI_PAINTER_impl.c** source file.

The implementation chosen on the NXP i.MXRT1170 is performing a **memcpy** when it comes to copying the screen content on itself.

Check the Image Renderer documentation to learn more about the drawImage implementations.

Next step: perform benchmarks on the NXP i.MXRT1170


Java
(MicroUI library)
BSP (C code)
(MicroUI C Module)
LLUI_PAINTER_impl.c
bsp > vee > port > ui > src > LLUI_PAINTER_impl.c > LLUI_PAINTER_IMPL_drawImage(MICROUI_GraphicsContext *, MICROUI_Image *, jint, jint, jint, jint, jint, jint, jint)
ui_drawing.c
bsp > vee > port > ui > src > ui_drawing.c > UI_DRAWING_DEFAULT_copyImage(MICROUI_GraphicsContext *, MICROUI_Image *, jint, jint, jint, jint, jint, jint)

```java
  public static void drawDisplayRegion(GraphicsContext gc, int xSrc, int ySrc, int width, int height, int xDest,
          int yDest) {
      assert gc != null;
      drawRegion(gc, Display.getDisplay().getGraphicsContext().getSNIContext(), xSrc, ySrc, width, height, xDest,
              yDest, GraphicsContext.OPAQUE);
  }

private static void drawRegion(GraphicsContext gc, byte[] srcSd, int xSrc, int ySrc, int width, int height,
        int xDest, int yDest, int alpha) {

    // compute destination area
    xDest += gc.translateX;
    yDest += gc.translateY;

    PainterNatives.drawImage(gc.getSNIContext(), srcSd, xSrc, ySrc, width, height, xDest, yDest, alpha);
}
```

```c
}

// See the header file for the function documentation
void LLUI_PAINTER_IMPL_drawImage(MICROUI_GraphicsContext *gc, MICROUI_Image *img, jint regionX, jint regionY,
                                jint width, jint height, jint x, jint y, jint alpha) {
    if (LLUI_DISPLAY_requestDrawing(gc, (SNI_callback) & LLUI_PAINTER_IMPL_drawImage)) {
        DRAWING_Status status = DRAWING_DONE;
        LOG_DRAW_START(drawImage);

                if (gc->image.format == img->format) {
                    // source & destination have got the same pixels memory representation

                    MICROUI_Image *image = LLUI_DISPLAY_getSourceImage(img);

                    if ((0xff /* fully opaque */ == l_alpha) && !LLUI_DISPLAY_isTransparent(img)) {
                        // copy source on destination without applying an opacity (beware about the overlapping)
                        status = UI_DRAWING_copyImage(gc, image, regionX, regionY, width, height, x, y);
                    } else if (LLUI_DISPLAY_getBufferAddress(img) == LLUI_DISPLAY_getBufferAddress(&gc->image)) {
                        // blend source on itself applying an opacity (beware about the overlapping)
                        status = UI_DRAWING_drawRegion(gc, regionX, regionY, width, height, x, y, l_alpha);
```

```c
// See the header file for the function documentation
BSP_DECLARE_WEAK_FCNT DRAWING_Status UI_DRAWING_DEFAULT_copyImage(MICROUI_GraphicsContext *gc, MICROUI_Image *img,
                                                                  jint regionX, jint regionY, jint width, jint height,
                                                                  jint x, jint y) {
#if !defined(LLUI_IMAGE_CUSTOM_FORMATS)
    return UI_DRAWING_SOFT_copyImage(gc, img, regionX, regionY, width, height, x, y);
#else
    return UI_IMAGE_DRAWING_copy(gc, img, regionX, regionY, width, height, x, y);
#endif
}
```

Performs a memcpy

ANALYSIS (6/6)

PERFORM BENCHMARKS ON THE TARGET

- Knowing that **drawImage** operations are “taking too much time” to execute, benchmarks should be performed on the target to figure out which hardware element is the bottleneck.
- Several kinds of benchmarks can be executed:
  - At the BSP level (see Core Testsuite Engine):
    - EEMBC Coremark (see
    - RAM speed tests
  - At MICROEJ VEE level:
    - Run GUI benchmarks in Java (see java-testsuite-runner-ui3)

The procedure on how to run benchmarks is not described in this training.

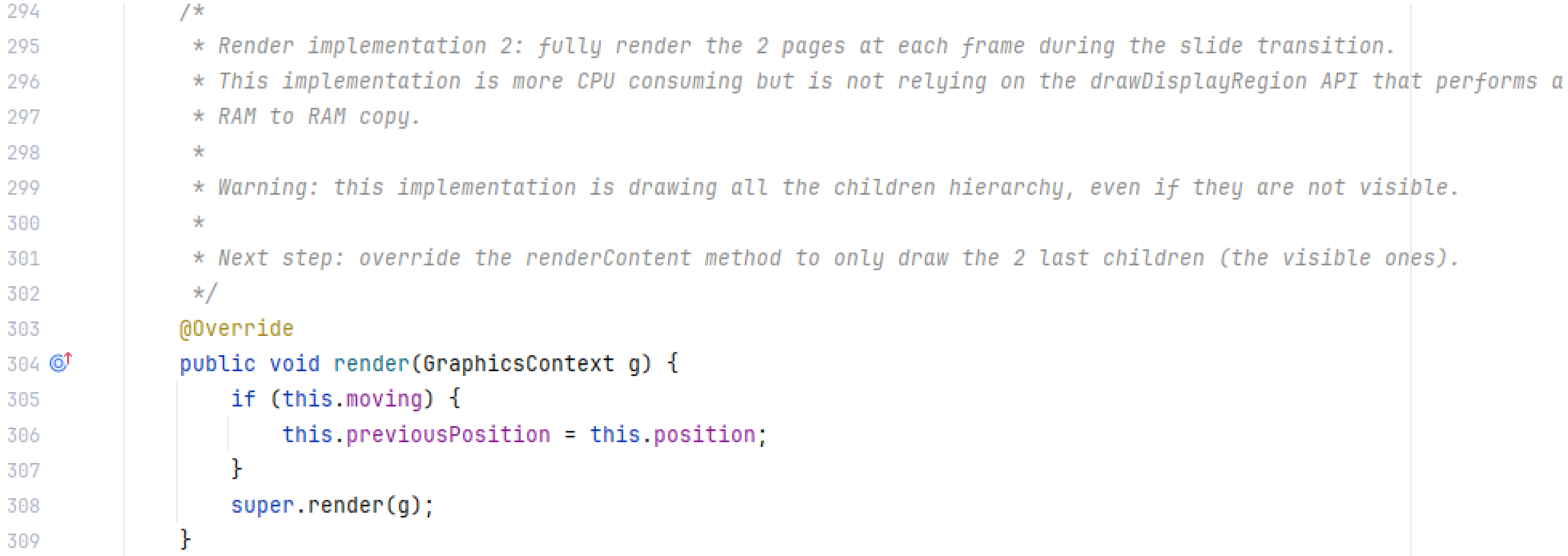
Conclusions on NXP i.MXRT1170:

- The screen has a high resolution (1280x720), thus a high number of pixels to drive:
  1280x720x16BPP/8 ~ 1.8Mb to transfer each time the screen is fully refreshed
- Front buffers are located in External RAM due to memory requirements.
- The benchmarks are showing that External RAM to External RAM copy is the bottleneck when it comes to copy a such amount of data. Hardware accelerators such as DMA or PXP are not improving results in that case.

→ On NXP i.MXRT1170 it is more interesting to limit RAM to RAM copy and perform drawings using the CPU to get a better framerate.

UPDATE THE APPLICATION CODE

- Open the **SlideContainer** class of the **slide-container** example.
- Comment the **Render implementation 1** (render() and renderContent() methods).
- Look for the Render implementation 2, uncomment the render() content method:

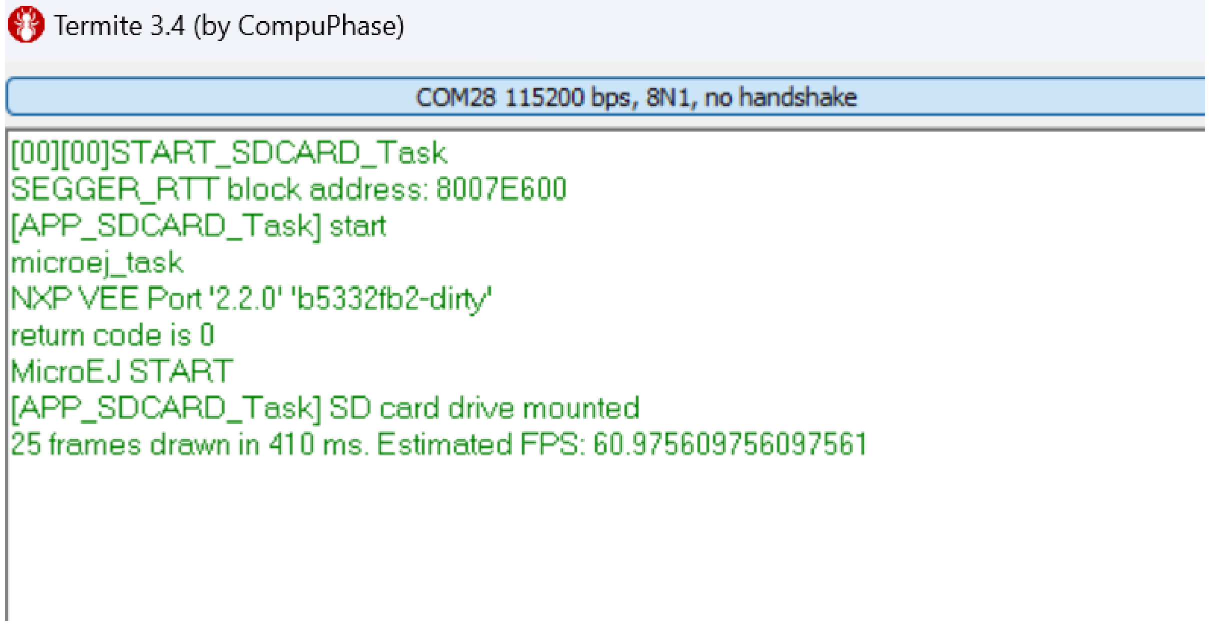


```
/*
 * Render implementation 2: fully render the 2 pages at each frame during the slide transition.
 * This implementation is more CPU consuming but is not relying on the drawDisplayRegion API that performs a
 * RAM to RAM copy.
 *
 * Warning: this implementation is drawing all the children hierarchy, even if they are not visible.
 *
 * Next step: override the renderContent method to only draw the 2 last children (the visible ones).
 */
@Override
public void render(GraphicsContext g) {
    if (this.moving) {
        this.previousPosition = this.position;
    }
    super.render(g);
}
```

- Run the application on the device.

RUN THE APPLICATION ON THE DEVICE

- Click on the **Show next** button on the screen.
- The implementation looks way smoother, see video in **slide-container/videos/slide_containrer_nxp_rt1170_optimized.m4v**
- The FPS have increased to 60 FPS:

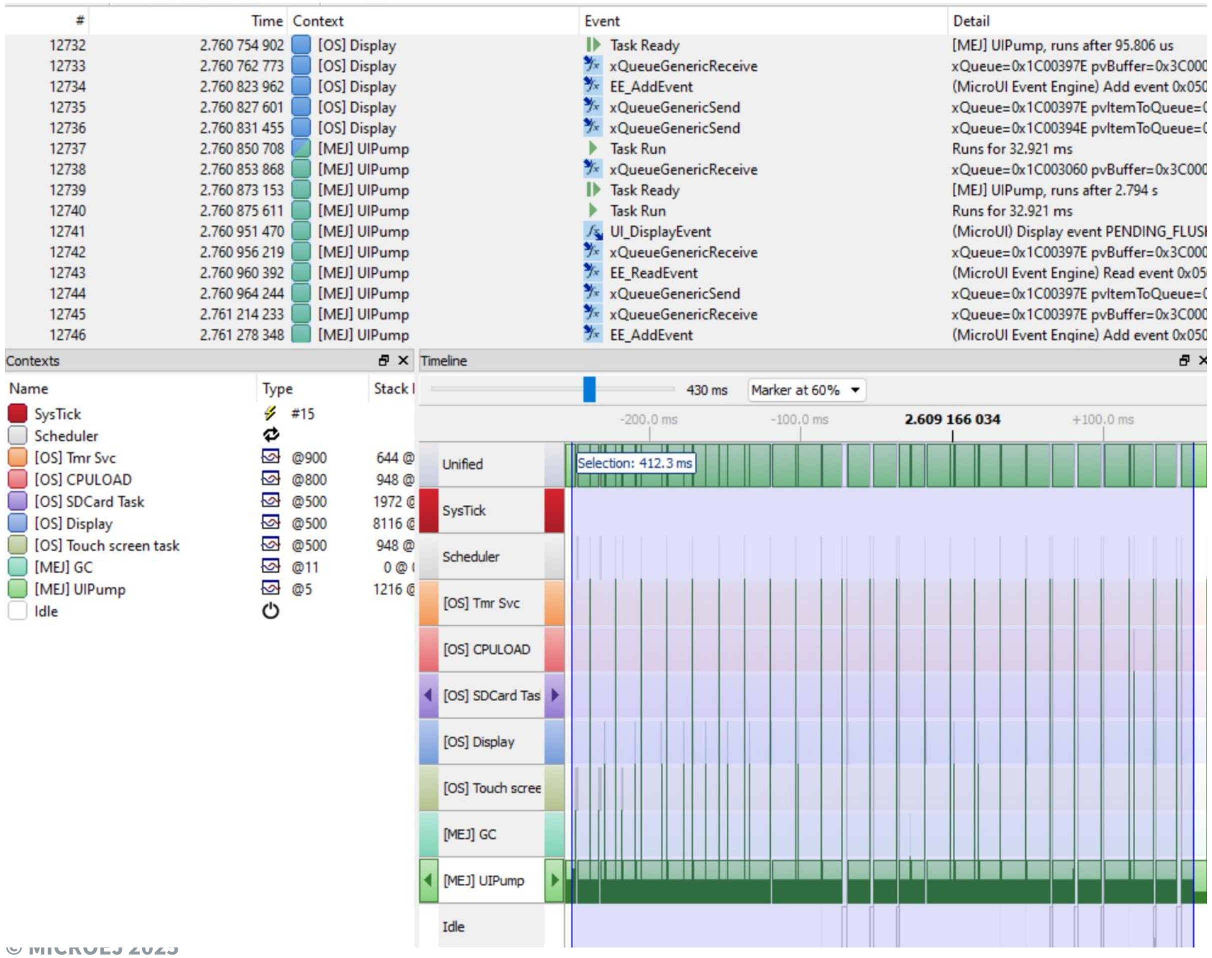

Termite 3.4 (by CompuPhase)
COM28 115200 bps, 8N1, no handshake

```
[00][00]START_SDCARD_Task
SEGGER_RTT block address: 8007E600
[APP_SDCARD_Task] start
microej_task
NXP VEE Port '2.2.0' 'b5332fb2-dirty'
return code is 0
MicroEJ START
[APP_SDCARD_Task] SD card drive mounted
25 frames drawn in 410 ms. Estimated FPS: 60.975609756097561
```

- Next step (not part of this training): to go further in the optimizations, override the renderContent method to only draw the 2 last children (the visible ones).

SYSTEMVIEW ANALYSIS OF IMPLEMENTATION 2 (1/2)

The SystemView Analysis shows that all drawImage operations are gone.
They have been replaced by many **DRAW_STRING** and **FILL_RECTANGLE** operations (corresponding to what is drawn on the screen).



#	Time	Context	Event	Detail
12732	2.760 754 902	[OS] Display	Task Ready	[MEJ] UIPump, runs after 95.806 us
12733	2.760 762 773	[OS] Display	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000
12734	2.760 823 962	[OS] Display	EE_AddEvent	(MicroUI Event Engine) Add event 0x050
12735	2.760 827 601	[OS] Display	xQueueGenericSend	xQueue=0x1C00397E pvItemToQueue=(
12736	2.760 831 455	[OS] Display	xQueueGenericSend	xQueue=0x1C00394E pvItemToQueue=(
12737	2.760 850 708	[MEJ] UIPump	Task Run	Runs for 32.921 ms
12738	2.760 853 868	[MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C003060 pvBuffer=0x3C000
12739	2.760 873 153	[MEJ] UIPump	Task Ready	[MEJ] UIPump, runs after 2.794 s
12740	2.760 875 611	[MEJ] UIPump	Task Run	Runs for 32.921 ms
12741	2.760 951 470	[MEJ] UIPump	UI_DisplayEvent	(MicroUI) Display event PENDING_FLUSI
12742	2.760 956 219	[MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000
12743	2.760 960 392	[MEJ] UIPump	EE_ReadEvent	(MicroUI Event Engine) Read event 0x05
12744	2.760 964 244	[MEJ] UIPump	xQueueGenericSend	xQueue=0x1C00397E pvItemToQueue=(
12745	2.761 214 233	[MEJ] UIPump	xQueueGenericReceive	xQueue=0x1C00397E pvBuffer=0x3C000
12746	2.761 278 348	[MEJ] UIPump	EE_AddEvent	(MicroUI Event Engine) Add event 0x050

Contexts

Name	Type	Stack I
SysTick	#15	
Scheduler		
[OS] Tmr Svc	@900	644 @
[OS] CPULOAD	@800	948 @
[OS] SDCard Task	@500	1972 @
[OS] Display	@500	8116 @
[OS] Touch screen task	@500	948 @
[MEJ] GC	@11	0 @ (
[MEJ] UIPump	@5	1216 @
Idle		

The transition duration is now close to the expected 400ms transition time (412 ms).

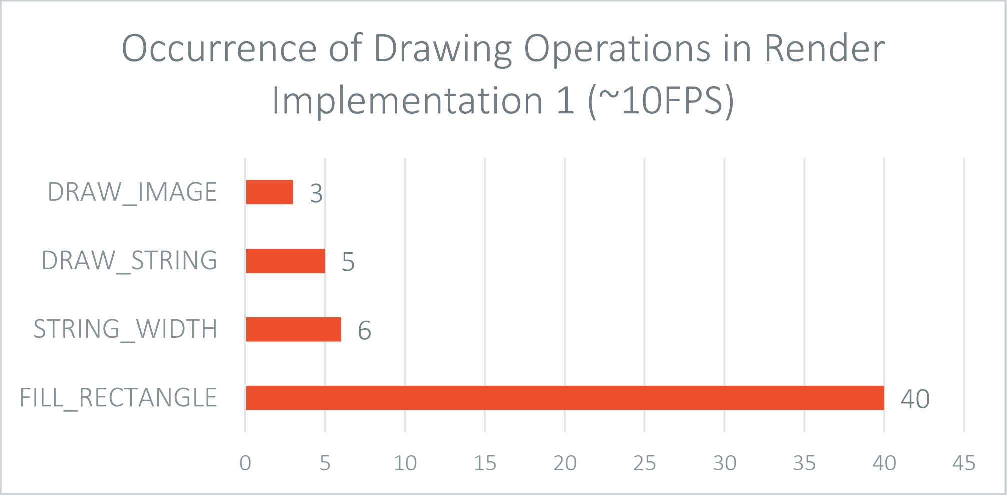
Note that the CPU load is still near 100% (almost no idle time in the timeline)

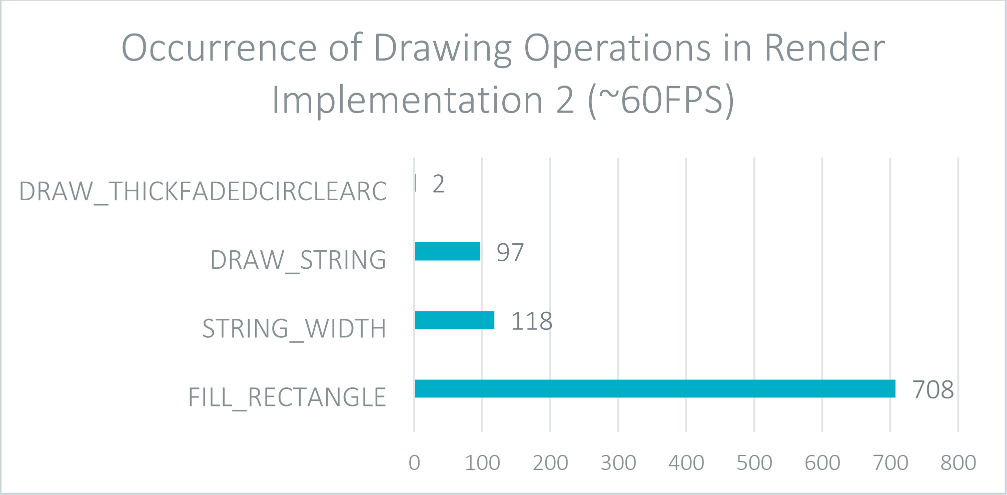
- This is due to the implementation of the slide animation. An Animator is used, it executes animations as fast as possible to get the best framerate.
- Check the Animations implementation documentation to learn more about the various implementations available.

SYSTEMVIEW ANALYSIS OF IMPLEMENTATION 2 (2/2)

- The SystemView analysis results are available **slide-container/systemView.**
  They have been exported to CSV format to perform a deeper analysis.

- Number of drawing operations:
  - Implementation 1 (~10FPS): 54
  - Implementation 2 (~60FPS): 925





MicroUI Event Buffer Dump

MICROUI EVENT BUFFER DUMP

MicroUI is using a circular buffer to manage the input events.

As soon as an event is added, removed, or replaced in the queue, the event engine calls the associated Abstraction Layer API (LLAPI) **LLUI_INPUT_IMPL_log_queue_xxx().** This LLAPI allows the BSP to log this event and to dump it later thanks to a call to **LLUI_INPUT_dump()** (see dump beside).

For more information, read MicroUI Event Buffer documentation.

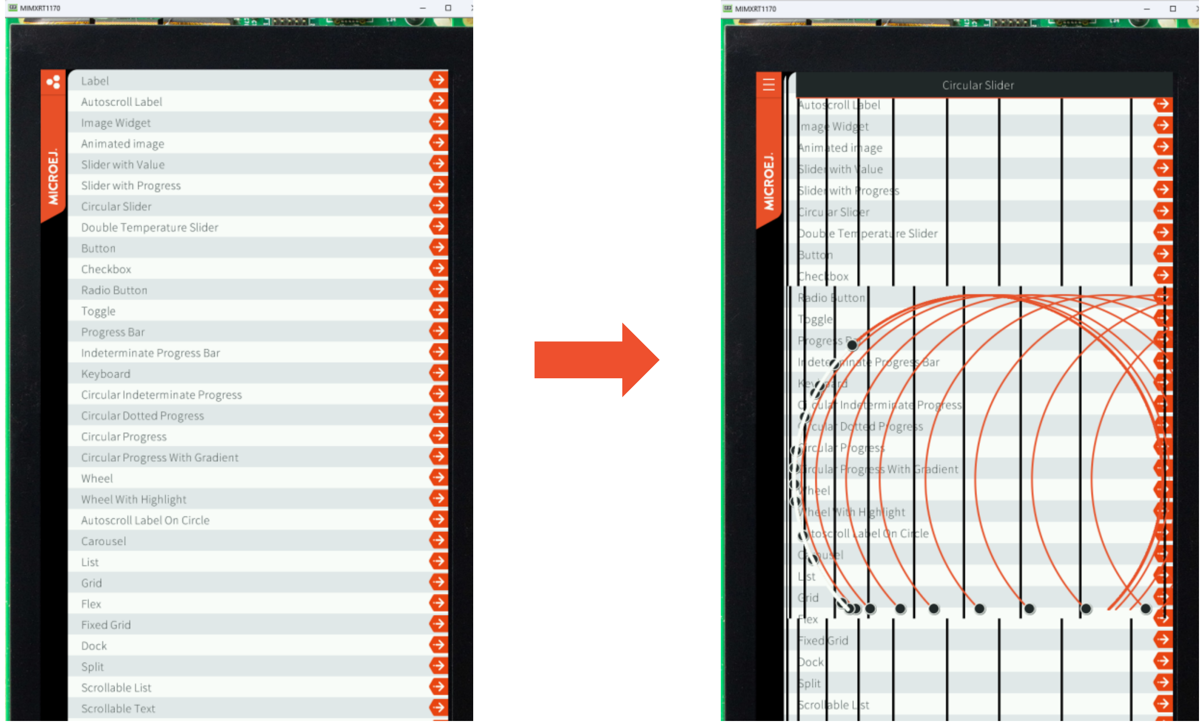
```
============================== MicroUI FIFO Dump ===============================
---------------------------------- Old Events ----------------------------------
[27: 0x00000000] garbage
[28: 0x00000000] garbage
[...]
[99: 0x00000000] garbage
[00: 0x08000000] Display SHOW Displayable (Displayable index = 0)
[01: 0x00000008] Command HELP (event generator 0)
[02: 0x0d000000] Display REPAINT Displayable (Displayable index = 0)
[03: 0x07030000] Input event: Pointer pressed (event generator 3)
[04: 0x009f0063]    at 159,99 (absolute)
[05: 0x07030600] Input event: Pointer moved (event generator 3)
[06: 0x00aa0064]    at 170,100 (absolute)
[07: 0x02030700] Pointer dragged (event generator 3)
[08: 0x0d000000] Display REPAINT Displayable (Displayable index = 0)
[09: 0x07030600] Input event: Pointer moved (event generator 3)
[10: 0x00b30066]    at 179,102 (absolute)
[11: 0x02030700] Pointer dragged (event generator 3)
[12: 0x0d000000] Display REPAINT Displayable (Displayable index = 0)
[13: 0x07030600] Input event: Pointer moved (event generator 3)
[14: 0x00c50067]    at 197,103 (absolute)
[15: 0x02030700] Pointer dragged (event generator 3)
[16: 0x0d000000] Display REPAINT Displayable (Displayable index = 0)
[17: 0x07030600] Input event: Pointer moved (event generator 3)
[18: 0x00d00066]    at 208,102 (absolute)
[19: 0x02030700] Pointer dragged (event generator 3)
[20: 0x0d000000] Display REPAINT Displayable (Displayable index = 0)
[21: 0x07030100] Input event: Pointer released (event generator 3)
[22: 0x00000000]    at 0,0 (absolute)
[23: 0x00000008] Command HELP (event generator 0)
---------------------------------- New Events ----------------------------------
[24: 0x0d000000] Display REPAINT Displayable (Displayable index = 0)
[25: 0x07030000] Input event: Pointer pressed (event generator 3)
[26: 0x002a0029]    at 42,41 (absolute)
--------------------------- New Events' Java objects ---------------------------
[java/lang/Object[2]@0xC000FD1C
 [0] com/microej/examples/microui/mvc/MVCDisplayable@0xC000BAC0
 [1] null
================================================================================
```

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

Debug tools provided in the Widget library

WIDGET DEBUG UTILITIES (1/3)

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

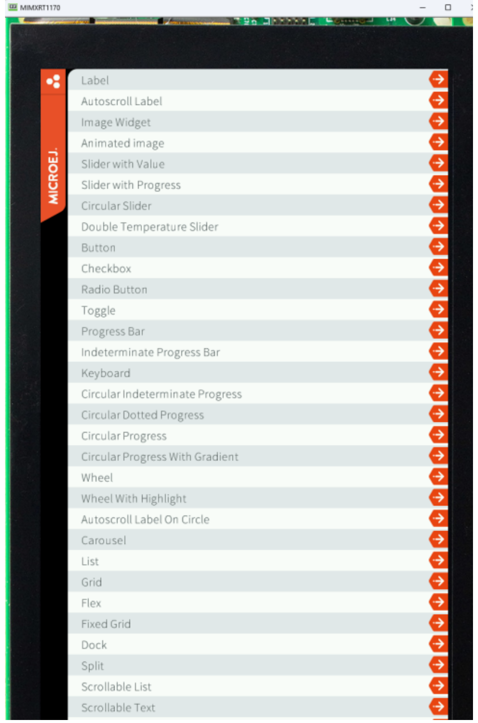
- Print the hierarchy of widgets and styles
- Print the path to a widget
- Count the number of widgets or containers
- Count the maximum depth of a hierarchy
- Print the bounds of a widget
- Print the bounds of all the widgets in a hierarchy

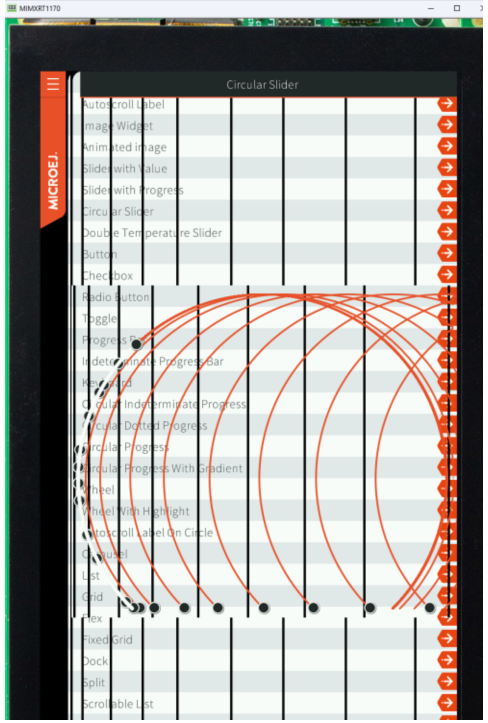
Check the Debug Utilities page for more information.

WIDGET DEBUG UTILITIES (2/3)

REPRODUCE THE ISSUE

- Run the **example-java-widget** project the on Simulator.
- Enter the **Circular Slider Page** to see the rendering issue:





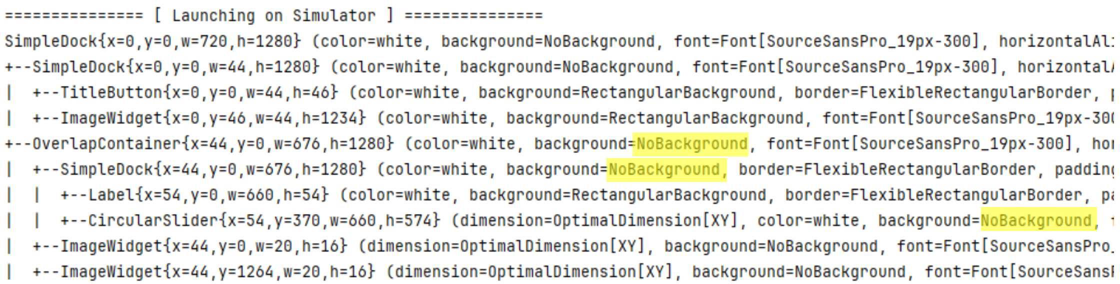
WIDGET DEBUG UTILITIES (3/3)

ROOT CAUSE ANALYSIS

- The background is not redrawn when the page shows up
- Add the following code in the **CircularSlider** page to print the style hierarchy of the Desktop:

```
@Override
protected void onShown(){
   HierarchyInspector.printHierarchyStyle(getDesktop().getWidget());
   super.onShown();
}
```

- The following output can be seen in the console:

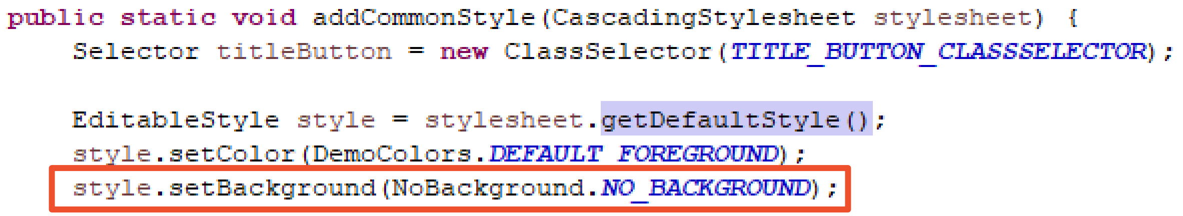


```
=============== [ Launching on Simulator ] ===============
SimpleDock{x=0,y=0,w=720,h=1280} (color=white, background=NoBackground, font=Font[SourceSansPro_19px-300], horizontalAl
+--SimpleDock{x=0,y=0,w=44,h=1280} (color=white, background=NoBackground, font=Font[SourceSansPro_19px-300], horizontal
|  +--TitleButton{x=0,y=0,w=44,h=46} (color=white, background=RectangularBackground, border=FlexibleRectangularBorder, 
|  +--ImageWidget{x=0,y=46,w=44,h=1234} (color=white, background=RectangularBackground, font=Font[SourceSansPro_19px-30
+--OverlapContainer{x=44,y=0,w=676,h=1280} (color=white, background=NoBackground, font=Font[SourceSansPro_19px-300], ho
|  +--SimpleDock{x=44,y=0,w=676,h=1280} (color=white, background=NoBackground, border=FlexibleRectangularBorder, paddin
|  |  +--Label{x=54,y=0,w=660,h=54} (color=white, background=RectangularBackground, border=FlexibleRectangularBorder, p
|  |  +--CircularSlider{x=54,y=370,w=660,h=574} (dimension=OptimalDimension[XY], color=white, background=NoBackground, 
|  +--ImageWidget{x=44,y=0,w=20,h=16} (dimension=OptimalDimension[XY], background=NoBackground, font=Font[SourceSansPro_
|  +--ImageWidget{x=44,y=1264,w=20,h=16} (dimension=OptimalDimension[XY], background=NoBackground, font=Font[SourceSans
```

→ There are only transparent backgrounds used in the widget hierarchy

FIX

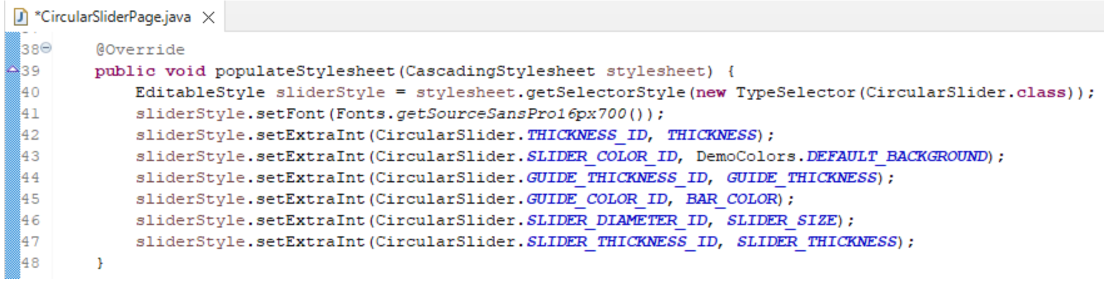
- Check the default style configuration:



```
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:



```
    @Override
    public void populateStylesheet(CascadingStylesheet stylesheet) {
        EditableStyle sliderStyle = stylesheet.getSelectorStyle(new TypeSelector(CircularSlider.class));
        sliderStyle.setFont(Fonts.getSourceSansPro16px700());
        sliderStyle.setExtraInt(CircularSlider.THICKNESS_ID, THICKNESS);
        sliderStyle.setExtraInt(CircularSlider.SLIDER_COLOR_ID, DemoColors.DEFAULT_BACKGROUND);
        sliderStyle.setExtraInt(CircularSlider.GUIDE_THICKNESS_ID, GUIDE_THICKNESS);
        sliderStyle.setExtraInt(CircularSlider.GUIDE_COLOR_ID, BAR_COLOR);
        sliderStyle.setExtraInt(CircularSlider.SLIDER_DIAMETER_ID, SLIDER_SIZE);
        sliderStyle.setExtraInt(CircularSlider.SLIDER_THICKNESS_ID, SLIDER_THICKNESS);
    }
```

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)

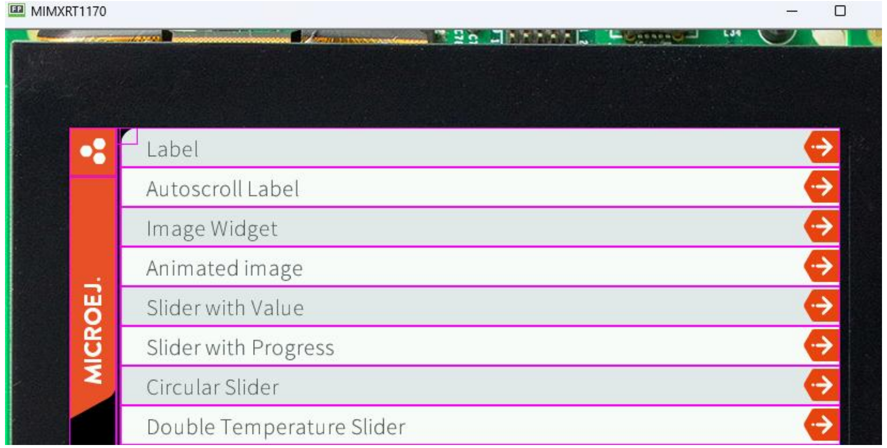
MWT Debug Utilities

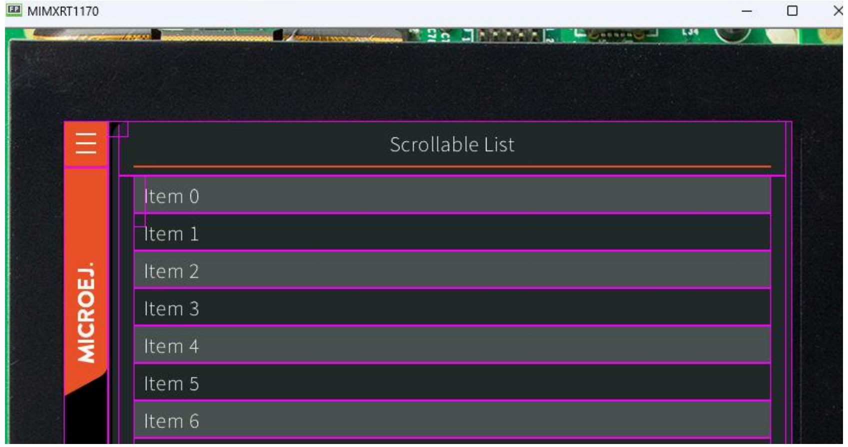
Debug tools provided in the MWT library

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:





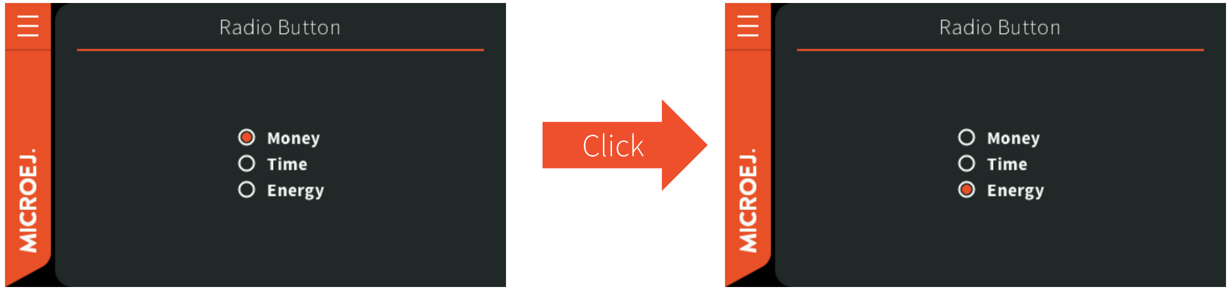
Check the Debug Utilities page for more information.

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}
```



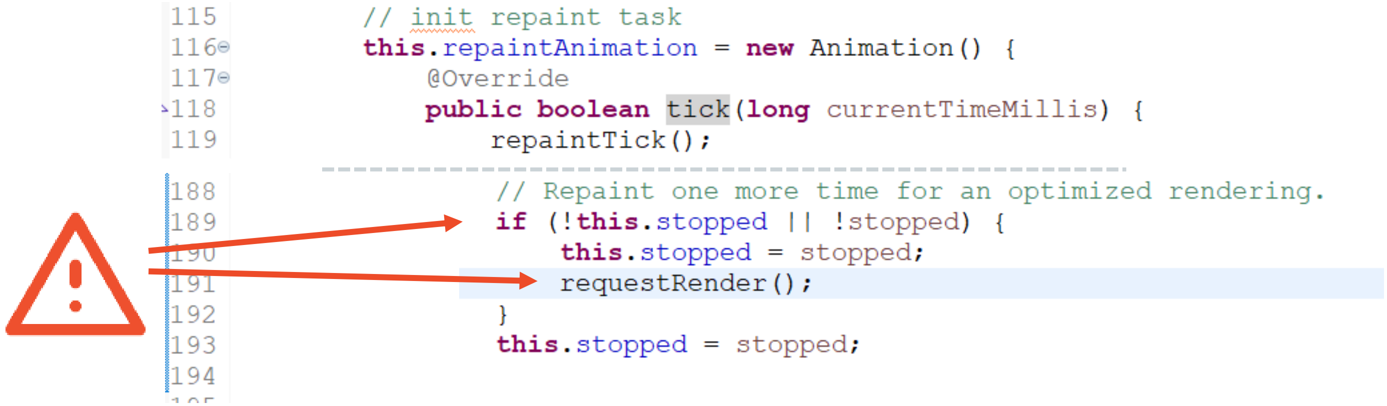
Check the Debug Utilities page for more information.

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]
```



```
// init repaint task
this.repaintAnimation = new Animation() {
    @Override
    public boolean tick(long currentTimeMillis) {
        repaintTick();
```

```
        // Repaint one more time for an optimized rendering.
        if (!this.stopped || !stopped) {
            this.stopped = stopped;
            requestRender();
        }
        this.stopped = stopped;
```

- 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

Check the Debug Utilities page for more information.

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 front buffer flushes
  → avoid useless redraws, improve performances
- MWT & Widget Debug utilities: detect issues with the widget hierarchy
  → debug rendering issues

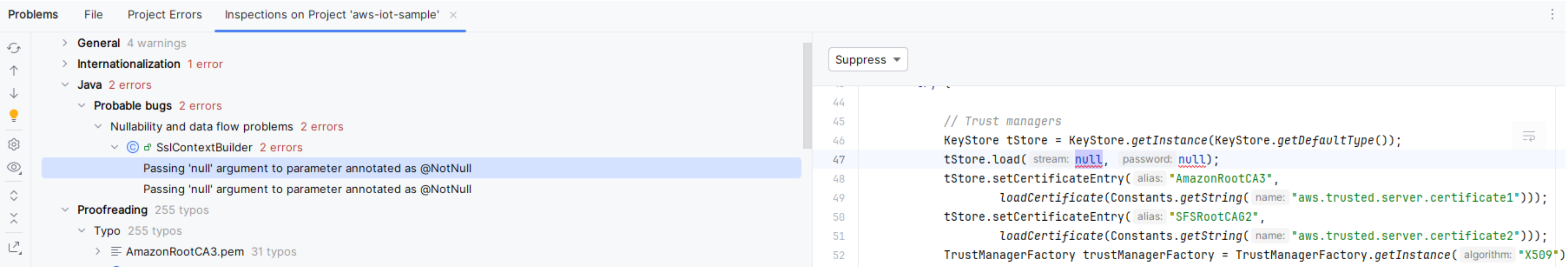
Static Analysis Tools

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.



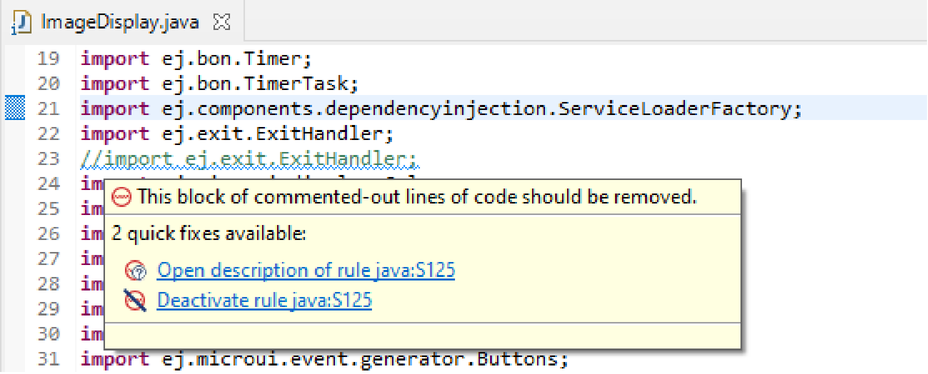
- General 4 warnings
- Internationalization 1 error
- Java 2 errors
  - Probable bugs 2 errors
    - Nullability and data flow problems 2 errors
      - SslContextBuilder 2 errors
        - Passing 'null' argument to parameter annotated as @NotNull
        - Passing 'null' argument to parameter annotated as @NotNull
- Proofreading 255 typos
  - Typo 255 typos
    - AmazonRootCA3.pem 31 typos

```

        // Trust managers
        KeyStore tStore = KeyStore.getInstance(KeyStore.getDefaultType());
        tStore.load( stream: null,  password: null);
        tStore.setCertificateEntry( alias: "AmazonRootCA3",
                loadCertificate(Constants.getString( name: "aws.trusted.server.certificate1")));
        tStore.setCertificateEntry( alias: "SFSRootCAG2",
                loadCertificate(Constants.getString( name: "aws.trusted.server.certificate2")));
        TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance( algorithm: "X509")
```

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)



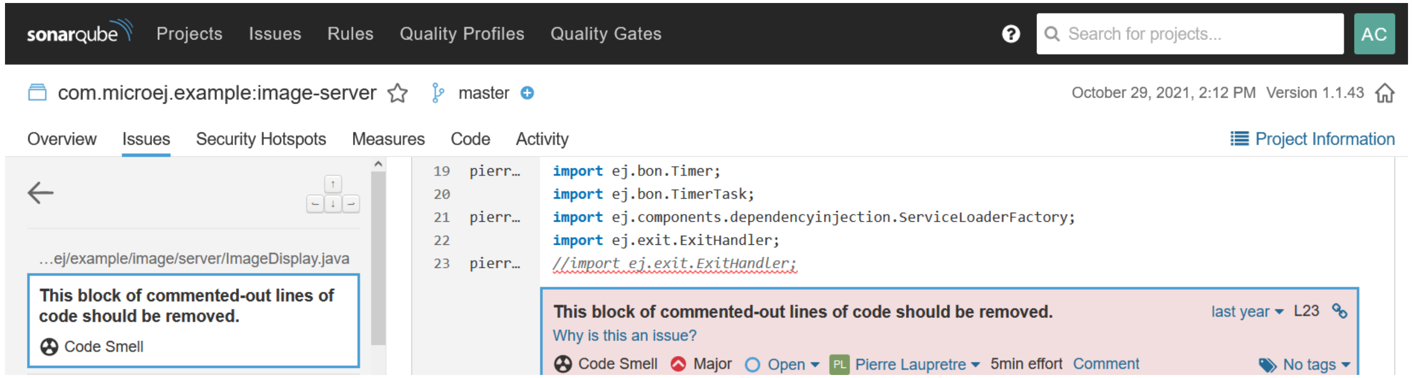
```
import ej.bon.Timer;
import ej.bon.TimerTask;
import ej.components.dependencyinjection.ServiceLoaderFactory;
import ej.exit.ExitHandler;
//import ej.exit.ExitHandler;
im
im
im
im
im
im
im
import ej.microui.event.generator.Buttons;
```

This block of commented-out lines of code should be removed.

2 quick fixes available:

- Open description of rule java:S125
- Deactivate rule java:S125

SonarQube code analysis performed inside Eclipse IDE



com.microej.example:image-server   master

October 29, 2021, 2:12 PM Version 1.1.43

…ej/example/image/server/ImageDisplay.java

This block of commented-out lines of code should be removed.

Code Smell

```
pierr…  import ej.bon.Timer;
        import ej.bon.TimerTask;
pierr…  import ej.components.dependencyinjection.ServiceLoaderFactory;
        import ej.exit.ExitHandler;
pierr…  //import ej.exit.ExitHandler;
```

This block of commented-out lines of code should be removed. last year L23

Why is this an issue?

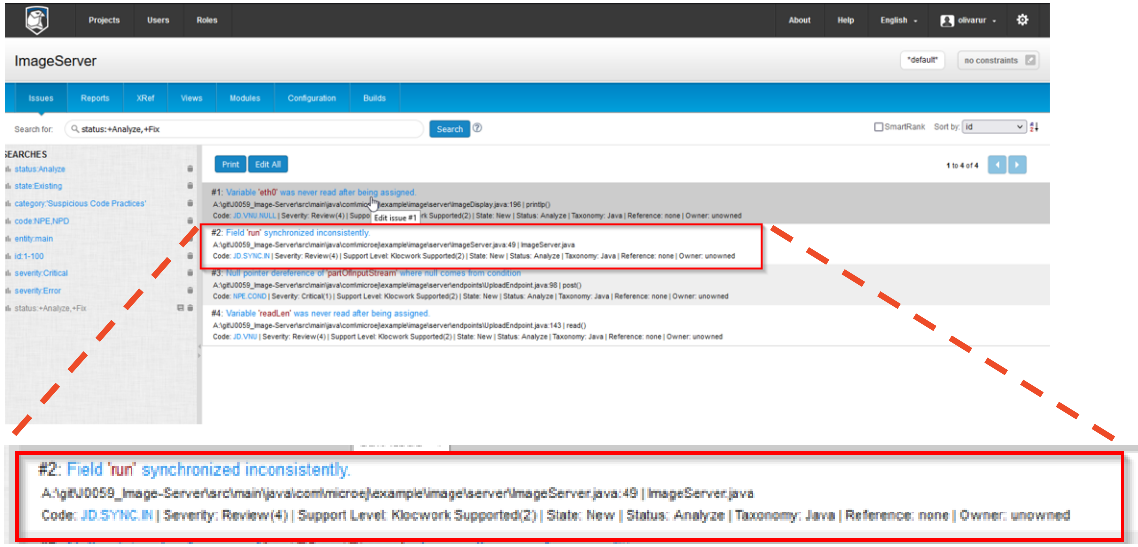
Code Smell Major Open PL Pierre Laupretre 5min effort Comment No tags

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.



ImageServer

Search for: status:+Analyze,+Fix

SEARCHES
- status:Analyze
- state:Existing
- category:'Suspicious Code Practices'
- code:NPE,NPD
- entity:main
- id:1-100
- severity:Critical
- severity:Error
- status:+Analyze,+Fix

#1: Variable 'eth0' was never read after being assigned.
A:\git\J0059_Image-Server\src\main\java\com\micro…\example\image\server\ImageDisplay.java:196 | println()
Code: JD.VNU.NULL | Severity: Review(4) | Suppo… …rk Supported(2) | State: New | Status: Analyze | Taxonomy: Java | Reference: none | Owner: unowned

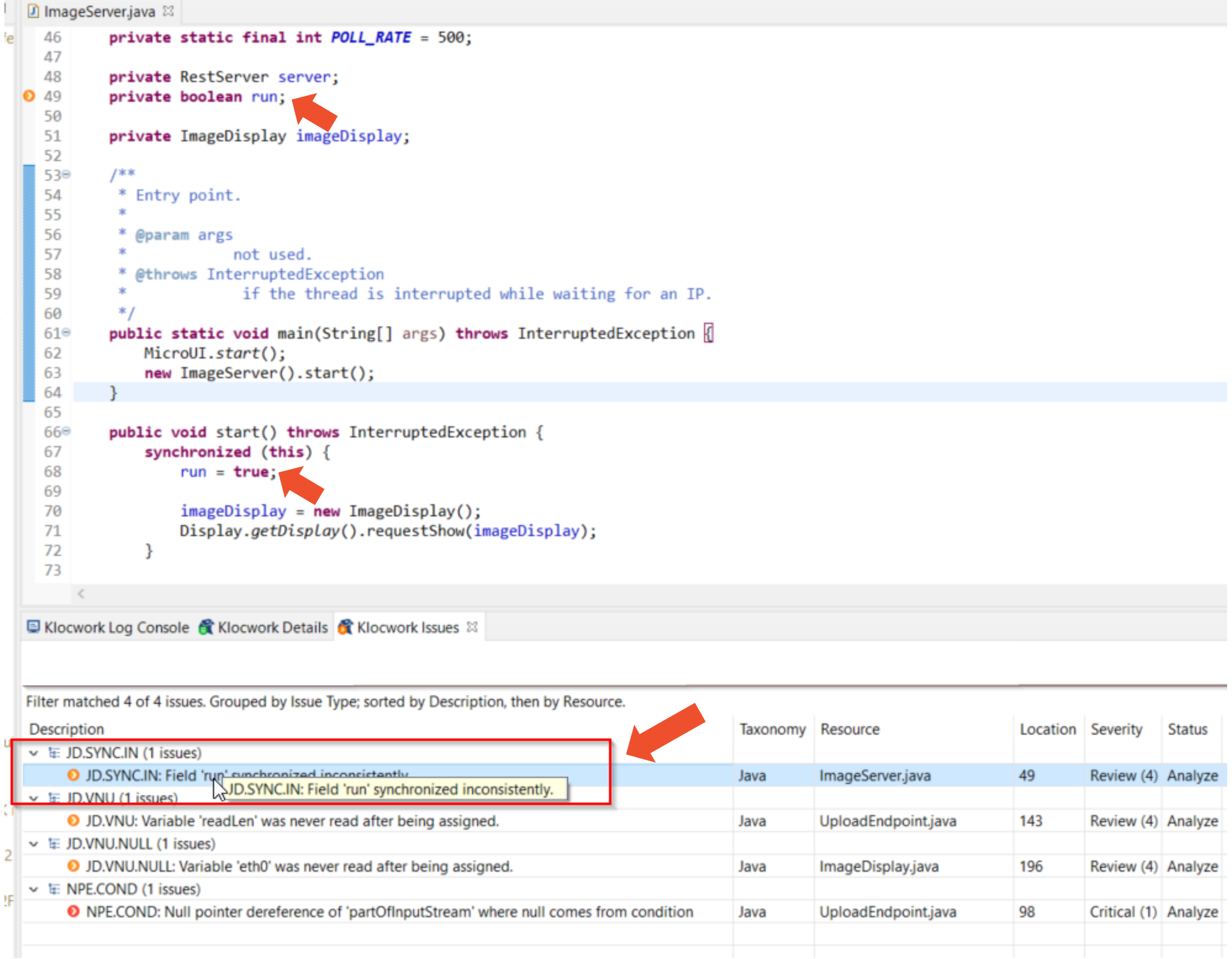
#2: Field 'run' synchronized inconsistently.
A:\git\J0059_Image-Server\src\main\java\com\microej\example\image\server\ImageServer.java:49 | ImageServer.java
Code: JD.SYNC.IN | Severity: Review(4) | Support Level: Klocwork Supported(2) | State: New | Status: Analyze | Taxonomy: Java | Reference: none | Owner: unowned

#3: Null pointer dereference of 'partOfInputStream' where null comes from condition
A:\git\J0059_Image-Server\src\main\java\com\microej\example\image\server\endpoints\UploadEndpoint.java:98 | post()
Code: NPE.COND | Severity: Critical(1) | Support Level: Klocwork Supported(2) | State: New | Status: Analyze | Taxonomy: Java | Reference: none | Owner: unowned

#4: Variable 'readLen' was never read after being assigned.
A:\git\J0059_Image-Server\src\main\java\com\microej\example\image\server\endpoints\UploadEndpoint.java:143 | read()
Code: JD.VNU | Severity: Review(4) | Support Level: Klocwork Supported(2) | State: New | Status: Analyze | Taxonomy: Java | Reference: none | Owner: unowned

#2: Field 'run' synchronized inconsistently.
A:\git\J0059_Image-Server\src\main\java\com\microej\example\image\server\ImageServer.java:49 | ImageServer.java
Code: JD.SYNC.IN | Severity: Review(4) | Support Level: Klocwork Supported(2) | State: New | Status: Analyze | Taxonomy: Java | Reference: none | Owner: unowned

Klocwork code analysis performed on Klocwork server



```java
    private static final int POLL_RATE = 500;

    private RestServer server;
    private boolean run;

    private ImageDisplay imageDisplay;

    /**
     * Entry point.
     *
     * @param args
     *            not used.
     * @throws InterruptedException
     *             if the thread is interrupted while waiting for an IP.
     */
    public static void main(String[] args) throws InterruptedException {
        MicroUI.start();
        new ImageServer().start();
    }

    public void start() throws InterruptedException {
        synchronized (this) {
            run = true;

            imageDisplay = new ImageDisplay();
            Display.getDisplay().requestShow(imageDisplay);
        }

```

Filter matched 4 of 4 issues. Grouped by Issue Type; sorted by Description, then by Resource.

Description	Taxonomy	Resource	Location	Severity	Status
JD.SYNC.IN (1 issues)					
JD.SYNC.IN: Field 'run' synchronized inconsistently.	Java	ImageServer.java	49	Review (4)	Analyze
JD.VNU (1 issues)					
JD.VNU: Variable 'readLen' was never read after being assigned.	Java	UploadEndpoint.java	143	Review (4)	Analyze
JD.VNU.NULL (1 issues)					
JD.VNU.NULL: Variable 'eth0' was never read after being assigned.	Java	ImageDisplay.java	196	Review (4)	Analyze
NPE.COND (1 issues)					
NPE.COND: Null pointer dereference of 'partOfInputStream' where null comes from condition	Java	UploadEndpoint.java	98	Critical (1)	Analyze

JD.SYNC.IN: Field 'run' synchronized inconsistently.

Klocwork code analysis performed inside Eclipse IDE

THANK YOU

for your attention !

MICROEJ®