Mastering MICROEJ SDK Development Tools

Accelerate your product development using MICROEJ SDK Tools

DISCLAIMER

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
- 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.
  - ✓ : This checkmark means the tool will be presented in this training.

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 1.3.1**. 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 (1/5)

MICROEJ

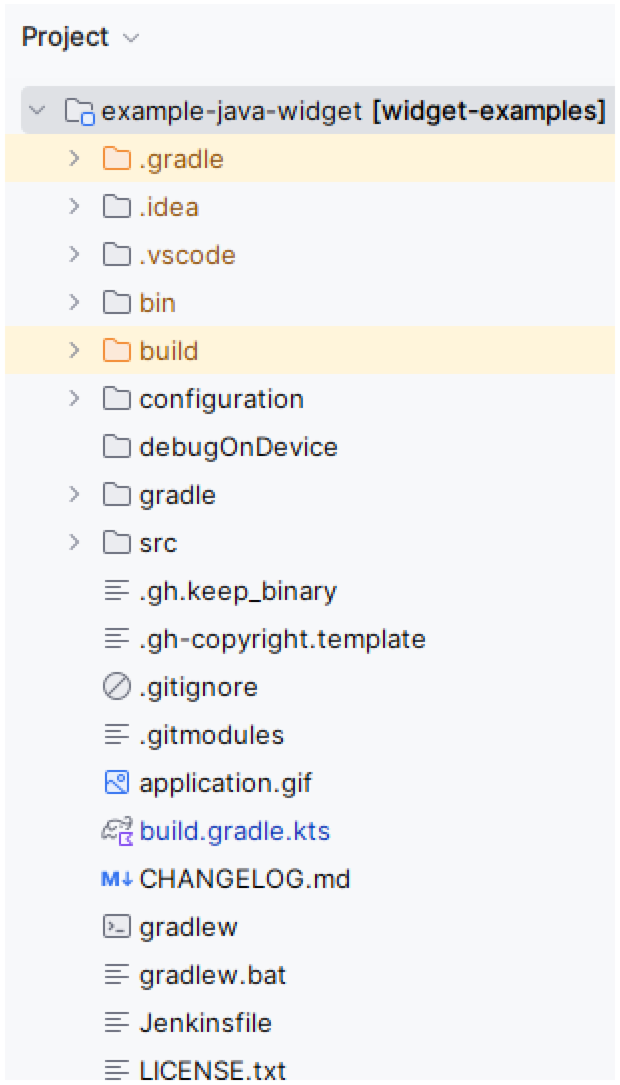
GET TRAINING RESOURCES

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

ENVIRONMENT SETUP (2/5)

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 (3/5)

VEE PORT SELECTION

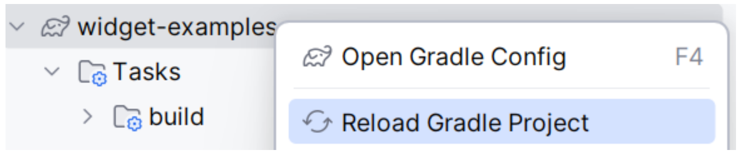
- Get the path to the **NXP i.MX RT1170 VEE Port** (e.g. C:\workspaces\training\nxpvee-mimxrt1170-evk\nxpvee-mimxrt1170-evk)
- Add the path to the VEE Port in the **settings.gradle.kts** file of the **example-java-widget** project:

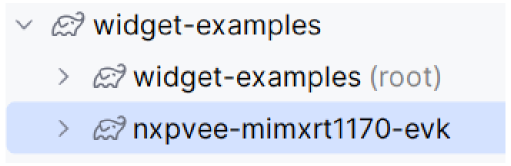
```
rootProject.name = “widget-examples"
includeBuild("C:\\workspaces\\training\\nxpvee-mimxrt1170-evk\\nxpvee-mimxrt1170-evk")
```

- 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:



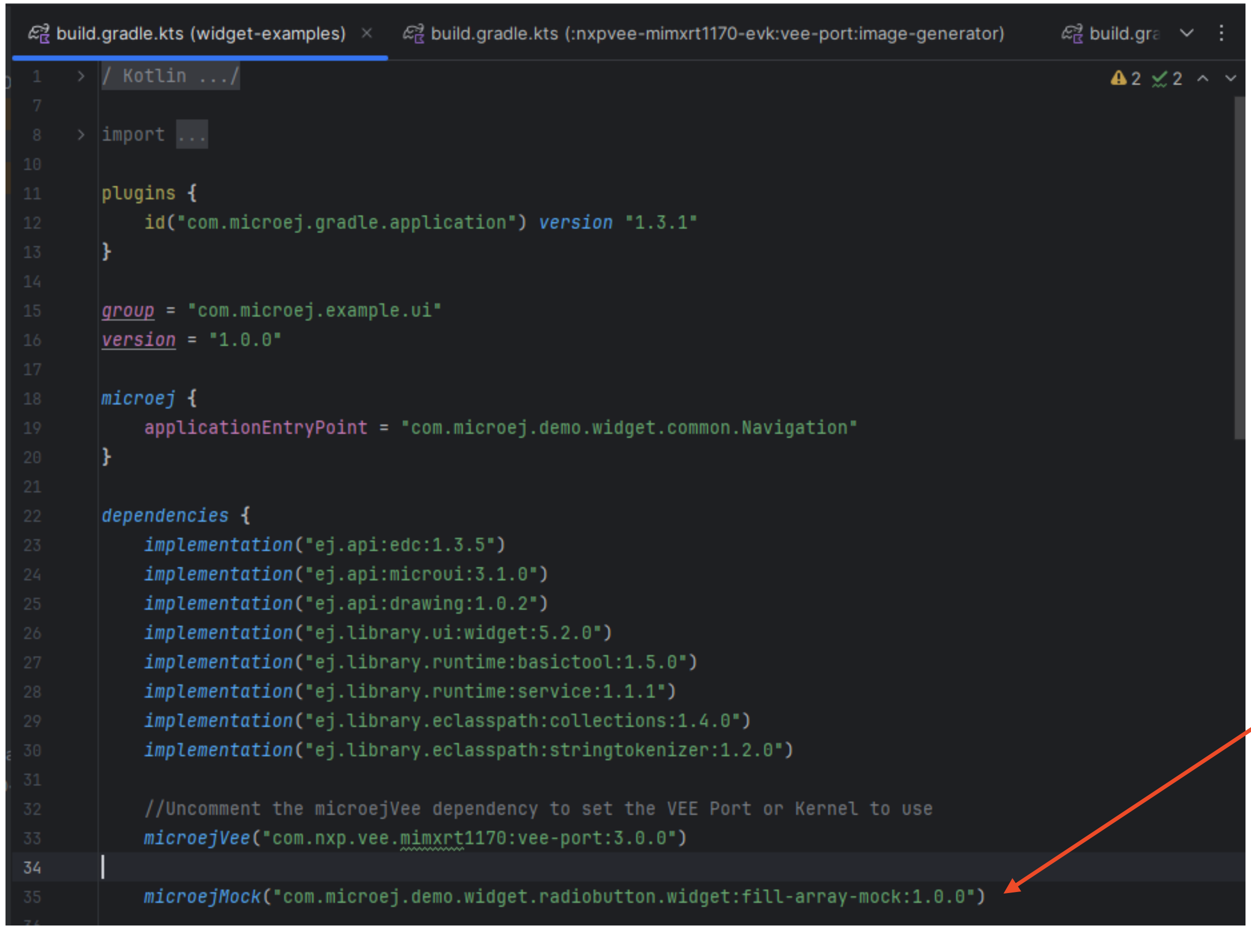


ENVIRONMENT SETUP (4/5)

ADD CUSTOM MOCK

A native function needs to be implemented in a mock to run the sample with the simulator:

- The mock is automatically built and added to the VEE port through a Gradle dependency in application project.


build.gradle.kts (widget-examples)
build.gradle.kts (:nxpvee-mimxrt1170-evk:vee-port:image-generator)
build.gra

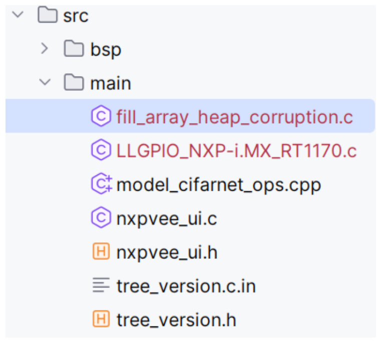
```
/ Kotlin .../

import ...

plugins {
    id("com.microej.gradle.application") version "1.3.1"
}

group = "com.microej.example.ui"
version = "1.0.0"

microej {
    applicationEntryPoint = "com.microej.demo.widget.common.Navigation"
}

dependencies {
    implementation("ej.api:edc:1.3.5")
    implementation("ej.api:microui:3.1.0")
    implementation("ej.api:drawing:1.0.2")
    implementation("ej.library.ui:widget:5.2.0")
    implementation("ej.library.runtime:basictool:1.5.0")
    implementation("ej.library.runtime:service:1.1.1")
    implementation("ej.library.eclasspath:collections:1.4.0")
    implementation("ej.library.eclasspath:stringtokenizer:1.2.0")

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

    microejMock("com.microej.demo.widget.radiobutton.widget:fill-array-mock:1.0.0")
```

ENVIRONMENT SETUP (5/5)

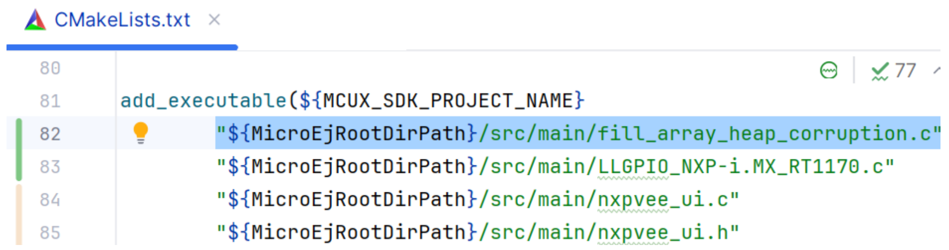
ADD CUSTOM NATIVE FUNCTION

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
```



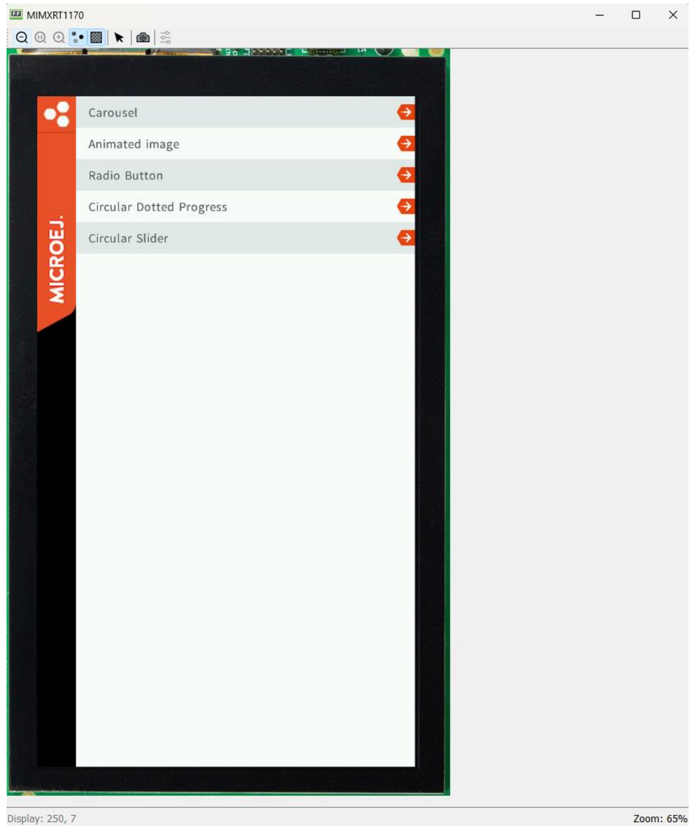
CMakeLists.txt

```
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"
```

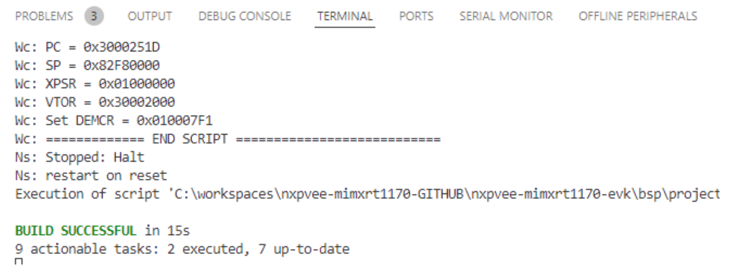
CHECK THE ENVIRONMENT SETUP

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
```

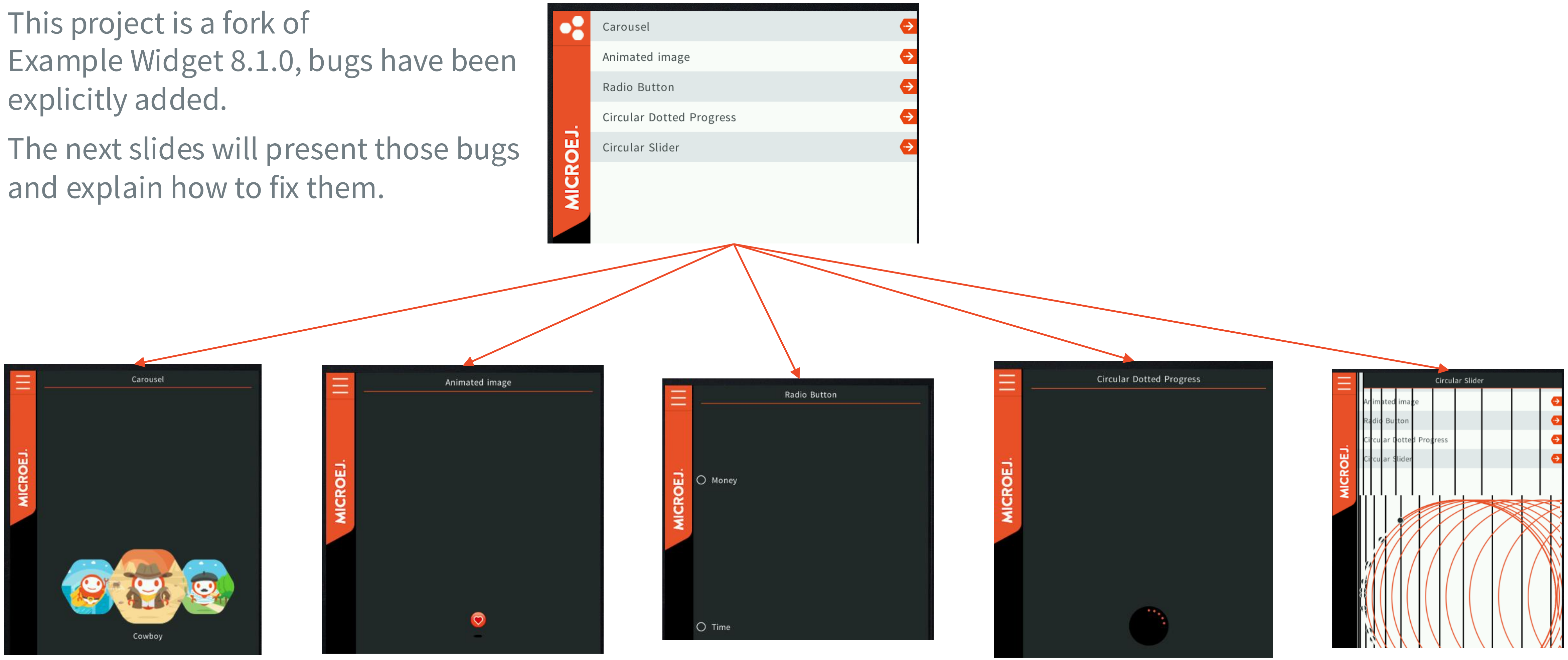
Example-Java-Widget Overview

Introduction to the Example-Java-Widget project

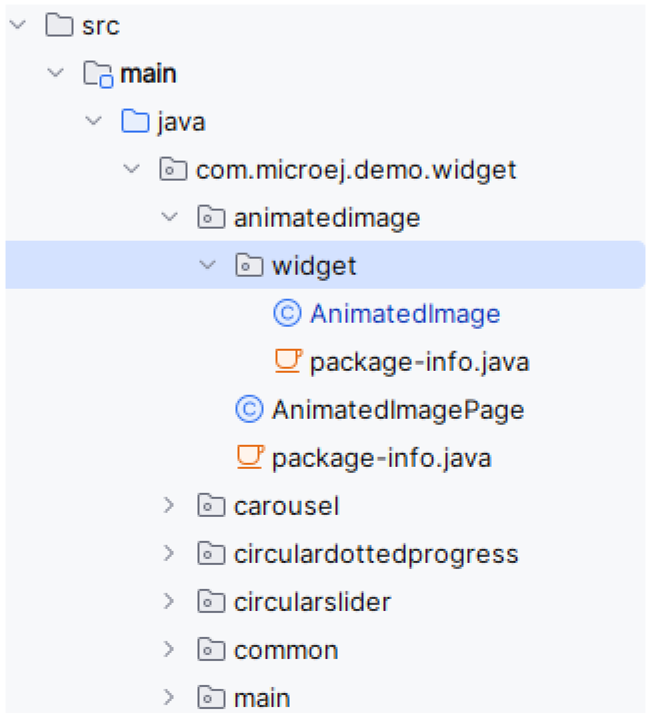
GUI OVERVIEW

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

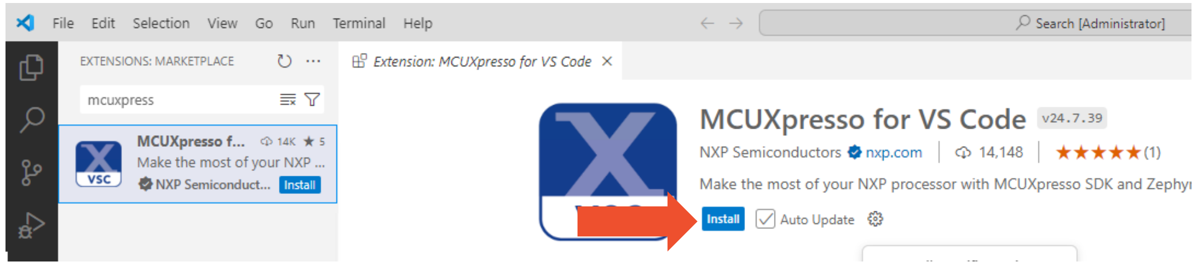
This section describes how to debug C code running on the NXP i.MXRT1170 using Visual Studio Code and GDB.

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.

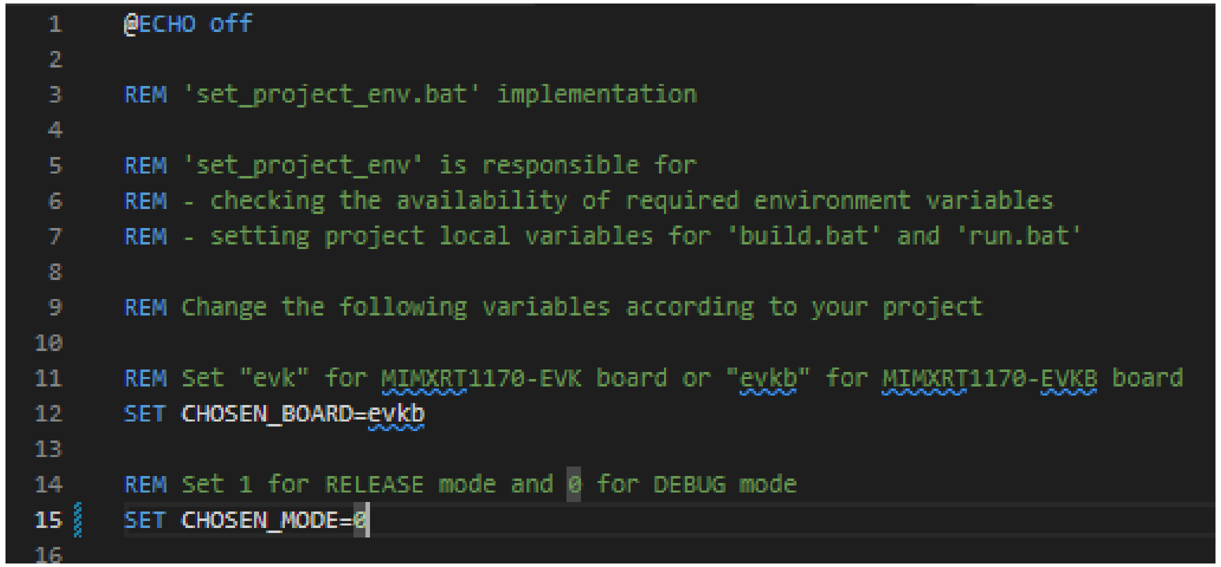
Requirements:

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


File Edit Selection View Go Run Terminal Help
Search [Administrator]
EXTENSIONS: MARKETPLACE
mcuxpress
MCUXpresso f... 14K 5
Make the most of your NXP ...
NXP Semiconduct... Install
Extension: MCUXpresso 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 Zephy
Install
Auto Update

ENABLE THE DEBUG MODE

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



```
@ECHO off

REM 'set_project_env.bat' implementation

REM 'set_project_env' is responsible for
REM - checking the availability of required environment variables
REM - setting project local variables for 'build.bat' and 'run.bat'

REM Change the following variables according to your project

REM Set "evk" for MIMXRT1170-EVK board or "evkb" for MIMXRT1170-EVKB board
SET CHOSEN_BOARD=evkb

REM Set 1 for RELEASE mode and 0 for DEBUG mode
SET CHOSEN_MODE=0
```

BSP DEBUGGING IN VS CODE (1/2)

SELECT THE DEBUG BUILD MODE

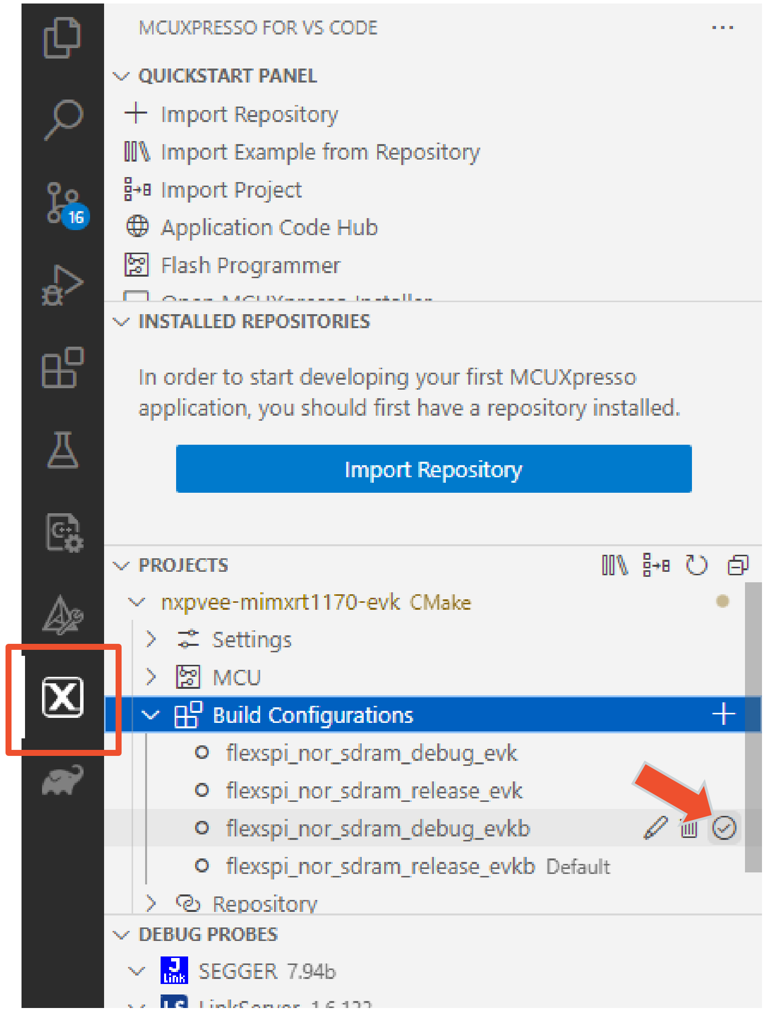
VS Code allows to build, flash and debug embedded projects.

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

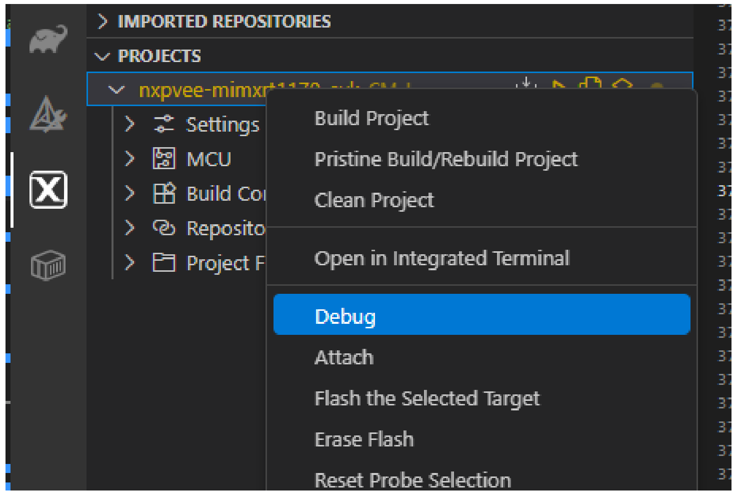
PROJECTS
- nxpvee-mimxrt1170-evk CMake
  - Settings
  - MCU
  - Build Configurations
    - flexspi_nor_sdram_debug_evk
    - flexspi_nor_sdram_release_evk
    - flexspi_nor_sdram_debug_evkb
    - flexspi_nor_sdram_release_evkb Default
  - Repository

DEBUG PROBES
- SEGGER 7.94b

BSP DEBUGGING IN VS CODE (2/2)

With your project still in VSCode 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).

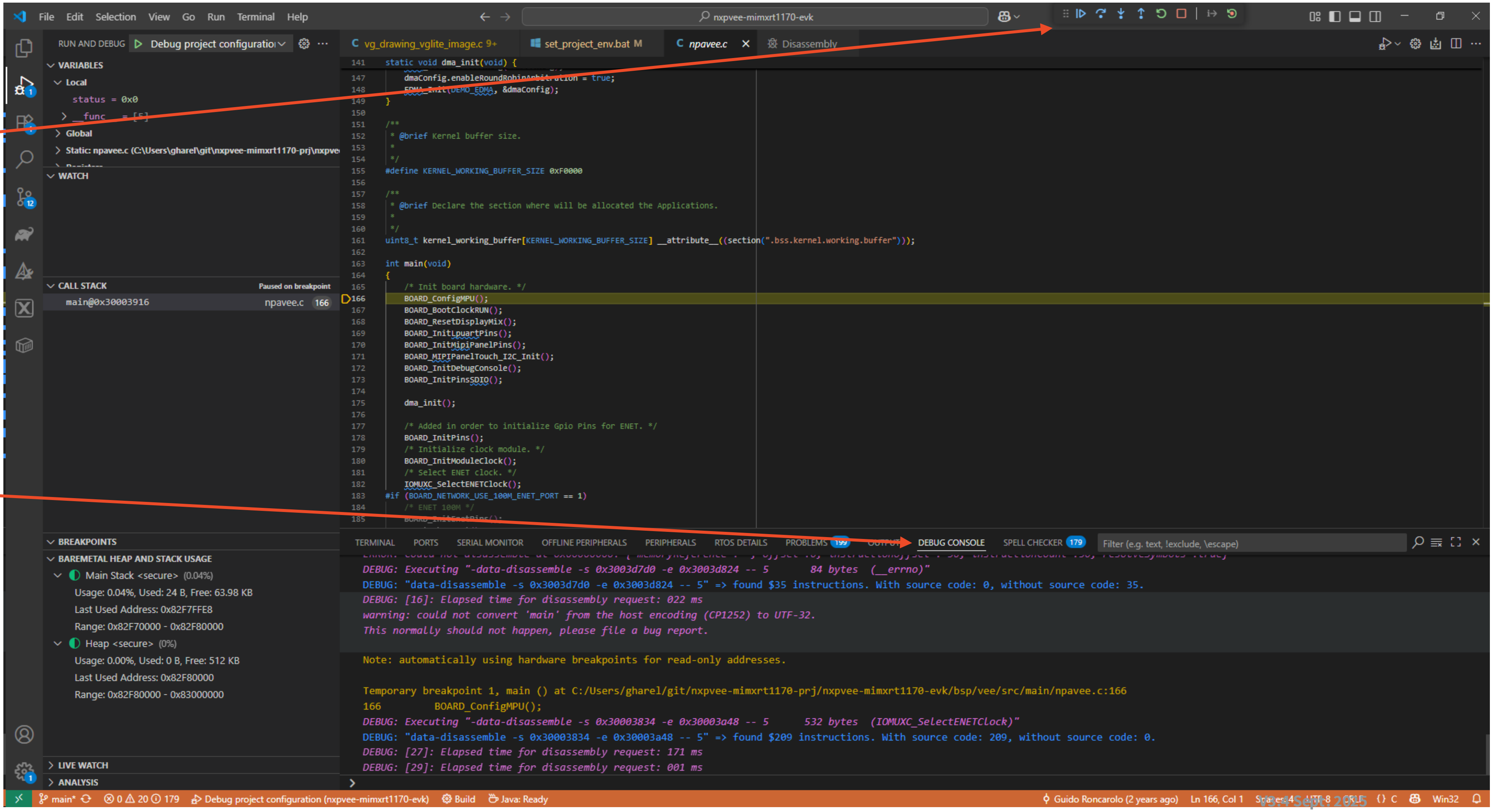


DEBUG VIEW IN VS CODE

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.



```
status = 0x0
__func__ = [5]
```

```
main@0x30003916          npavee.c  166
```

```
Main Stack <secure>  (0.04%)
    Usage: 0.04%, Used: 24 B, Free: 63.98 KB
    Last Used Address: 0x82F7FFE8
    Range: 0x82F70000 - 0x82F80000
Heap <secure>  (0%)
    Usage: 0.00%, Used: 0 B, Free: 512 KB
    Last Used Address: 0x82F80000
    Range: 0x82F80000 - 0x83000000
```

```c
static void dma_init(void) {
    dmaConfig.enableRoundRobinArbitration = true;
    EDMA_Init(DEMO_EDMA, &dmaConfig);
}

/**
 * @brief Kernel buffer size.
 *
 */
#define KERNEL_WORKING_BUFFER_SIZE 0xF0000

/**
 * @brief Declare the section where will be allocated the Applications.
 *
 */
uint8_t kernel_working_buffer[KERNEL_WORKING_BUFFER_SIZE] __attribute__((section(".bss.kernel.working.buffer")));

int main(void)
{
    /* Init board hardware. */
    BOARD_ConfigMPU();
    BOARD_BootClockRUN();
    BOARD_ResetDisplayMix();
    BOARD_InitLpuartPins();
    BOARD_InitMipiPanelPins();
    BOARD_MIPIPanelTouch_I2C_Init();
    BOARD_InitDebugConsole();
    BOARD_InitPinsSDIO();

    dma_init();

    /* Added in order to initialize Gpio Pins for ENET. */
    BOARD_InitPins();
    /* Initialize clock module. */
    BOARD_InitModuleClock();
    /* Select ENET clock. */
    IOMUXC_SelectENETClock();
#if (BOARD_NETWORK_USE_100M_ENET_PORT == 1)
    /* ENET 100M */
    BOARD_InitEnetPins();
```

```
DEBUG: Executing "-data-disassemble -s 0x3003d7d0 -e 0x3003d824 -- 5       84 bytes  (__errno)"
DEBUG: "data-disassemble -s 0x3003d7d0 -e 0x3003d824 -- 5" => found $35 instructions. With source code: 0, without source code: 35.
DEBUG: [16]: Elapsed time for disassembly request: 022 ms
warning: could not convert 'main' from the host encoding (CP1252) to UTF-32.
This normally should not happen, please file a bug report.

Note: automatically using hardware breakpoints for read-only addresses.

Temporary breakpoint 1, main () at C:/Users/gharel/git/nxpvee-mimxrt1170-prj/nxpvee-mimxrt1170-evk/bsp/vee/src/main/npavee.c:166
166         BOARD_ConfigMPU();
DEBUG: Executing "-data-disassemble -s 0x30003834 -e 0x30003a48 -- 5      532 bytes  (IOMUXC_SelectENETClock)"
DEBUG: "data-disassemble -s 0x30003834 -e 0x30003a48 -- 5" => found $209 instructions. With source code: 209, without source code: 0.
DEBUG: [27]: Elapsed time for disassembly request: 171 ms
DEBUG: [29]: Elapsed time for disassembly request: 001 ms
```

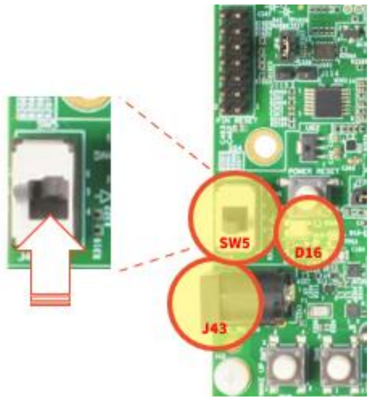
```
main*  ⊗ 0 ⚠ 20 ⓘ 179  Debug project configuration (nxpvee-mimxrt1170-evk)  Build  Java: Ready    Guido Roncarolo (2 years ago)  Ln 166, Col 1  Spaces: 4  UTF-8  CRLF  {} C  Win32
```

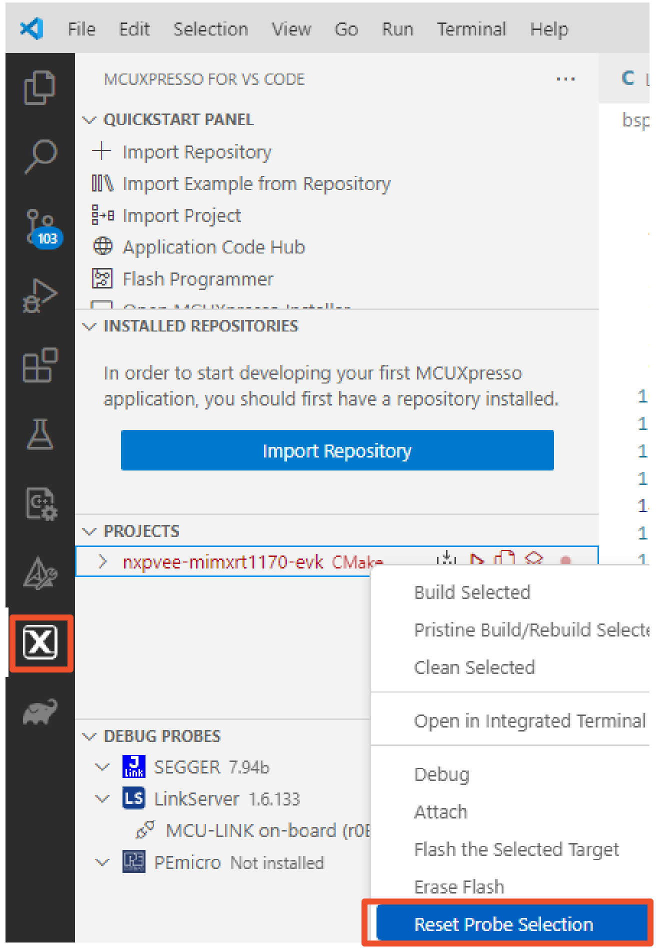
TROUBLESHOOTING

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.

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





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

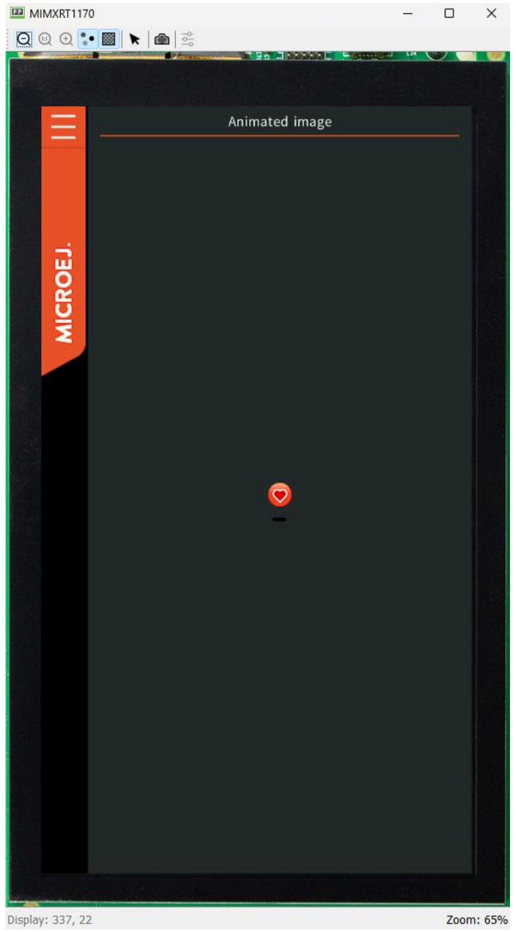
Runtime & Post-Mortem Debugging Tools

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



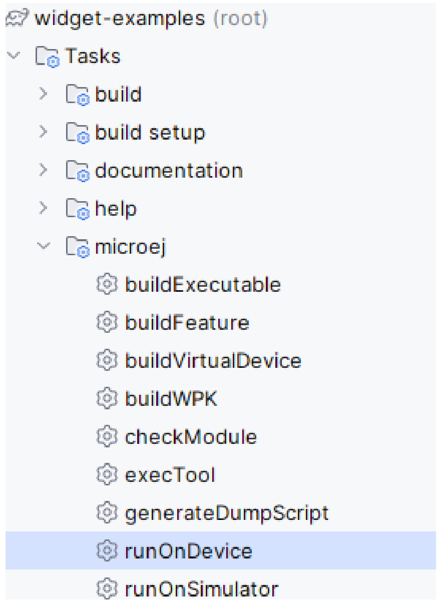
GUI freeze when entering a page

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

DEBUG A DEADLOCK IN AN APPLICATION

REPRODUCE THE ISSUE

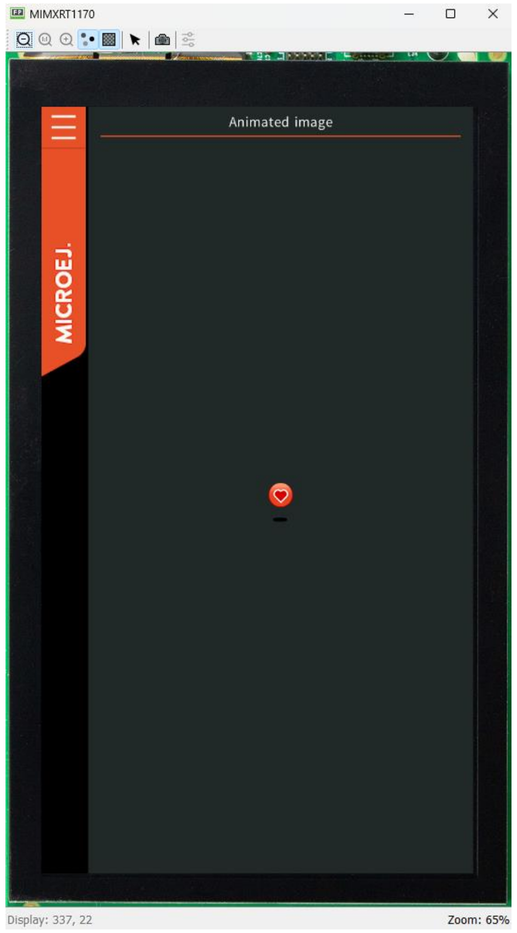
- Run the **example-java-widget** project the on Device:



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.

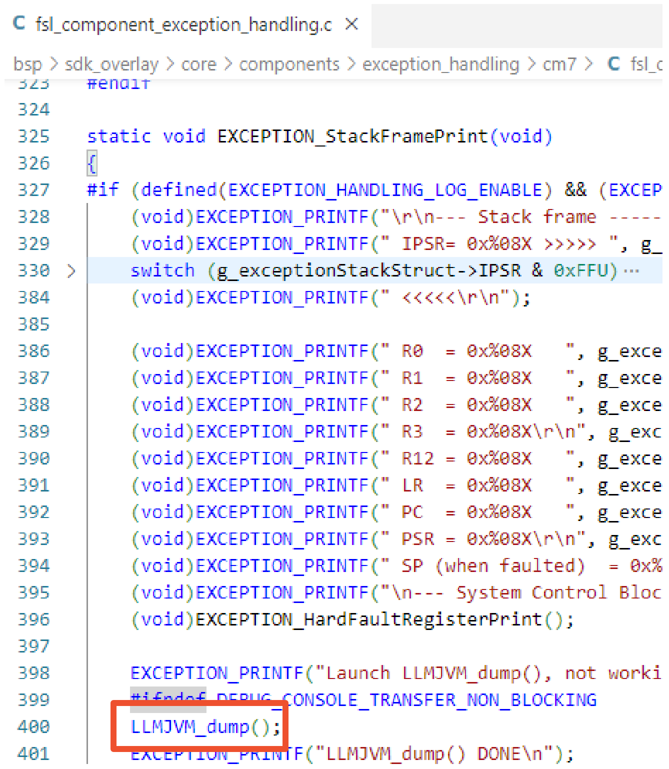
- What?
  - Prints the state of the MicroEJ Core Engine to the standard output stream.
  - For each Java thread, the Java stack trace, the name, the state and the priority are printed.

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

CORE ENGINE VM DUMP (2/4)

HOW-TO?

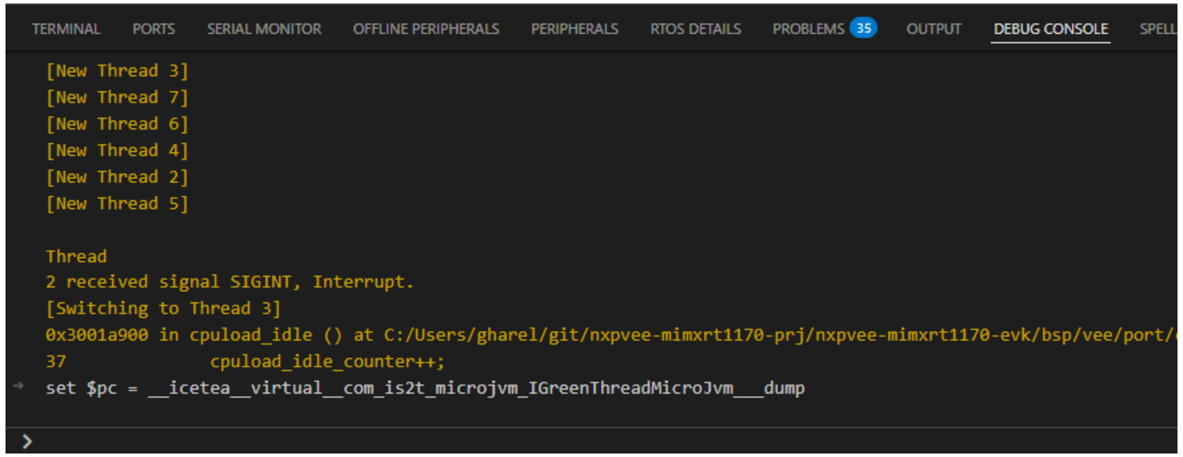
- Example of LLMJVM Dump triggered from a fault handler:



C fsl_component_exception_handling.c ×
bsp > sdk_overlay > core > components > exception_handling > cm7 > C 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):



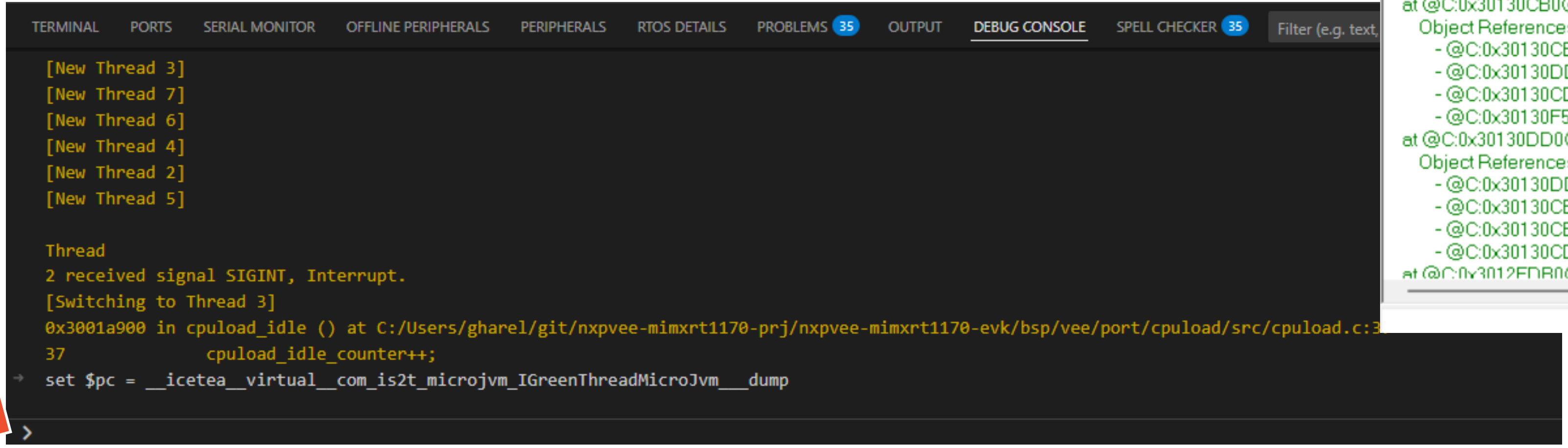
```
[New Thread 3]
[New Thread 7]
[New Thread 6]
[New Thread 4]
[New Thread 2]
[New Thread 5]

Thread
2 received signal SIGINT, Interrupt.
[Switching to Thread 3]
0x3001a900 in cpuload_idle () at C:/Users/gharel/git/nxpvee-mimxrt1170-prj/nxpvee-mimxrt1170-evk/bsp/vee/port/
37                cpuload_idle_counter++;
set $pc = __icetea__virtual__com_is2t_microjvm_IGreenThreadMicroJvm___dump
>
```

CORE ENGINE VM DUMP (3/4)

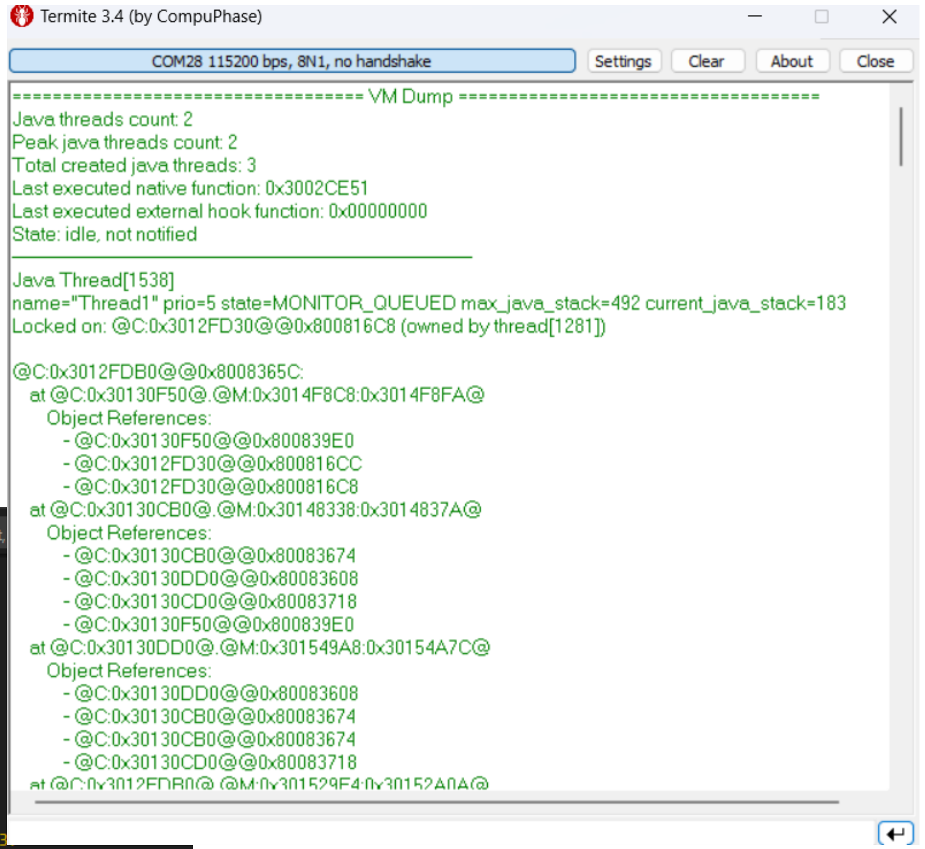
TRIGGER THE LLMJVM DUMP FROM THE DEBUGGER (VS CODE / GDB)

- 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 $pc = __icetea__virtual__com_is2t_microjvm_IGreenThreadMicroJvm___dump
```



```
[New Thread 3]
[New Thread 7]
[New Thread 6]
[New Thread 4]
[New Thread 2]
[New Thread 5]

Thread
2 received signal SIGINT, Interrupt.
[Switching to Thread 3]
0x3001a900 in cpuload_idle () at C:/Users/gharel/git/nxpvee-mimxrt1170-prj/nxpvee-mimxrt1170-evk/bsp/vee/port/cpuload/src/cpuload.c:3
37                cpuload_idle_counter++;
set $pc = __icetea__virtual__com_is2t_microjvm_IGreenThreadMicroJvm___dump
>
```


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@
```

- Click the “Play” button to resume the execution at the set symbol, the VM dump can be see in the serial terminal

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.

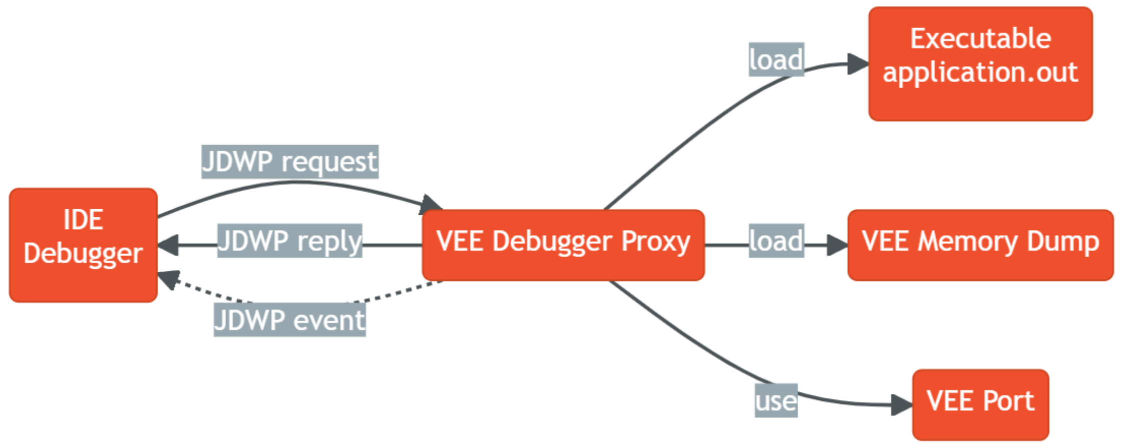
Debug on Device

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:

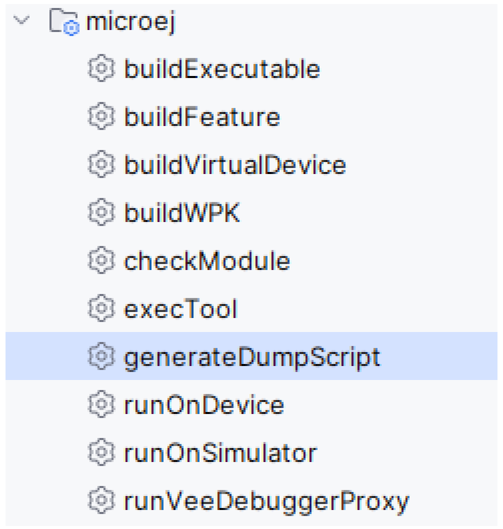


- Available since Architecture 8.1

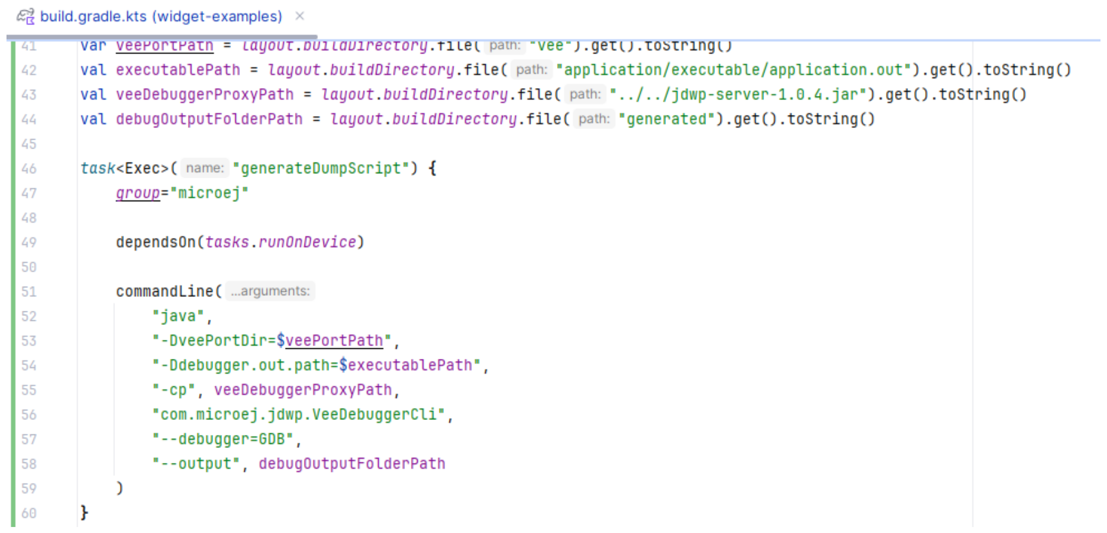
- No VEE Port update required
- Steps:
  1. Generate a VEE memory dump script for the target / toolchain
  2. Run the application Executable on target
  3. Dump the memory of the running Executable using the C Debugger using the VEE memory dump script
  4. Run the VEE Debugger Proxy in a Command Prompt
  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:



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)

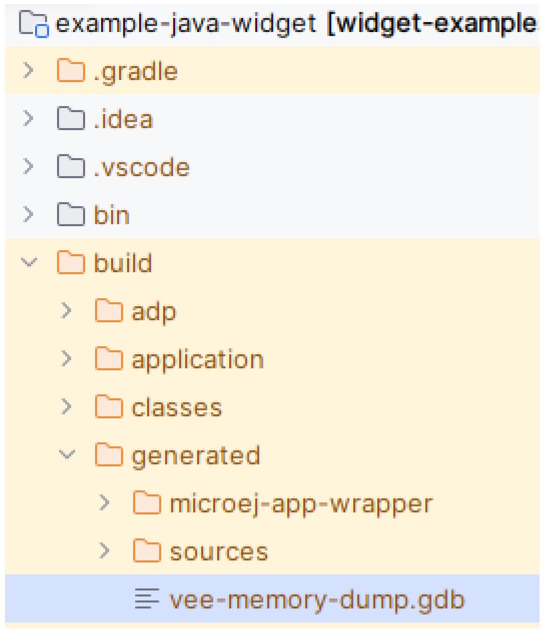
```
var veePortPath = layout.buildDirectory.file( path: "vee").get().toString()
val executablePath = layout.buildDirectory.file( path: "application/executable/application.out").get().toString()
val veeDebuggerProxyPath = layout.buildDirectory.file( path: "../../jdwp-server-1.0.4.jar").get().toString()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

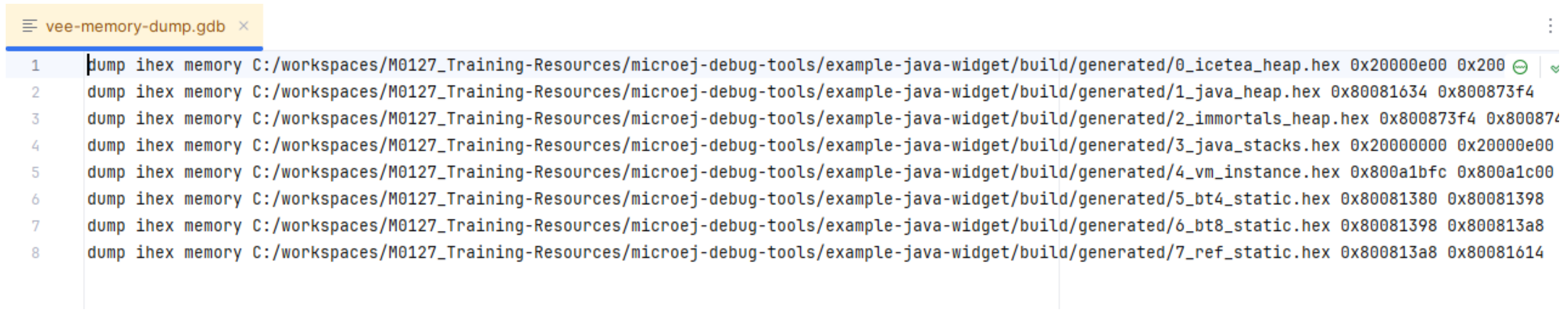
Note that this task is configured to generate a **GDB** dump script.

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:

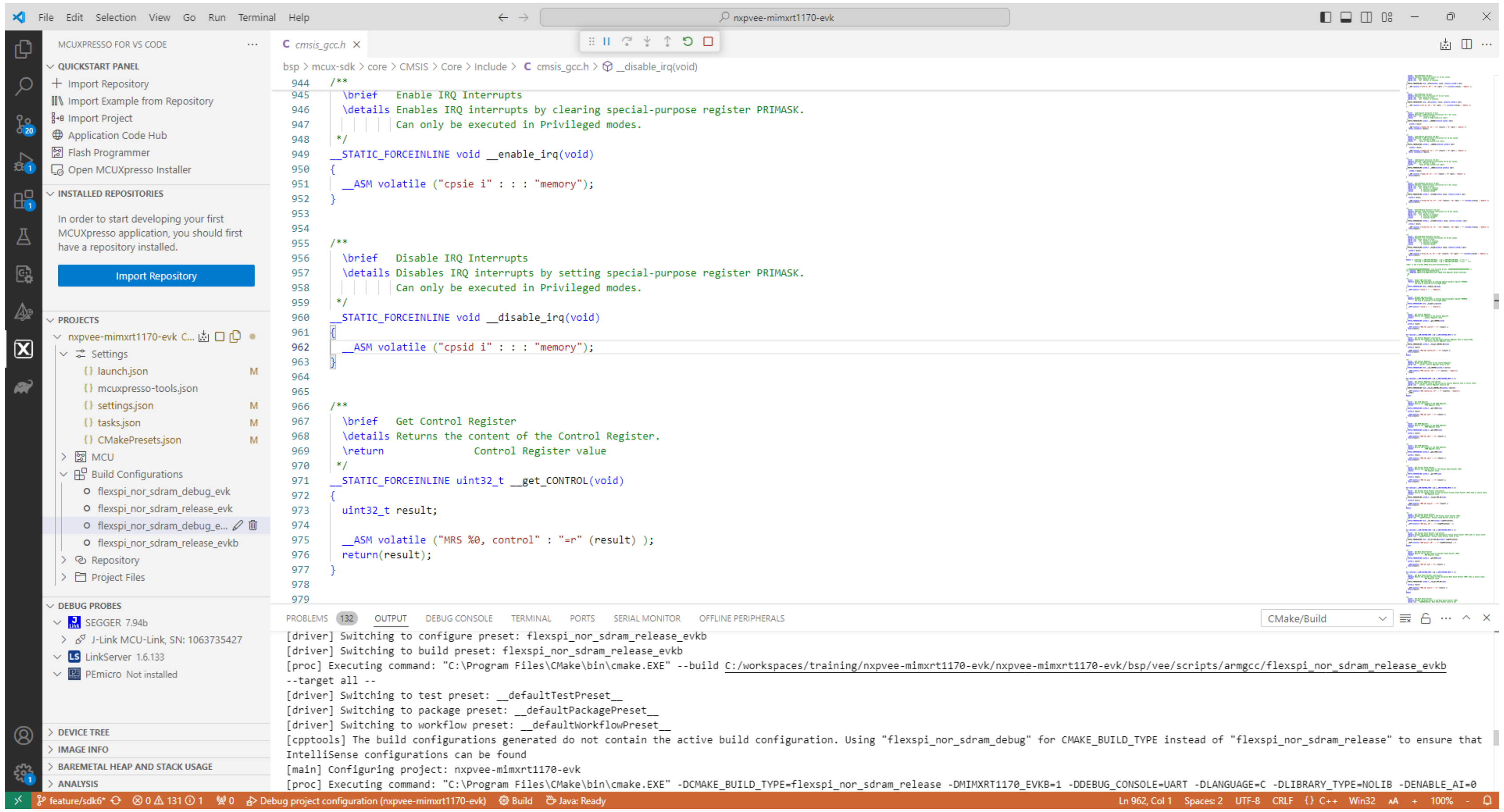




```
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/0_icetea_heap.hex 0x20000e00 0x200
dump ihex memory C:/workspaces/M0127_Training-Resources/microej-debug-tools/example-java-widget/build/generated/1_java_heap.hex 0x80081634 0x800873f4
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).



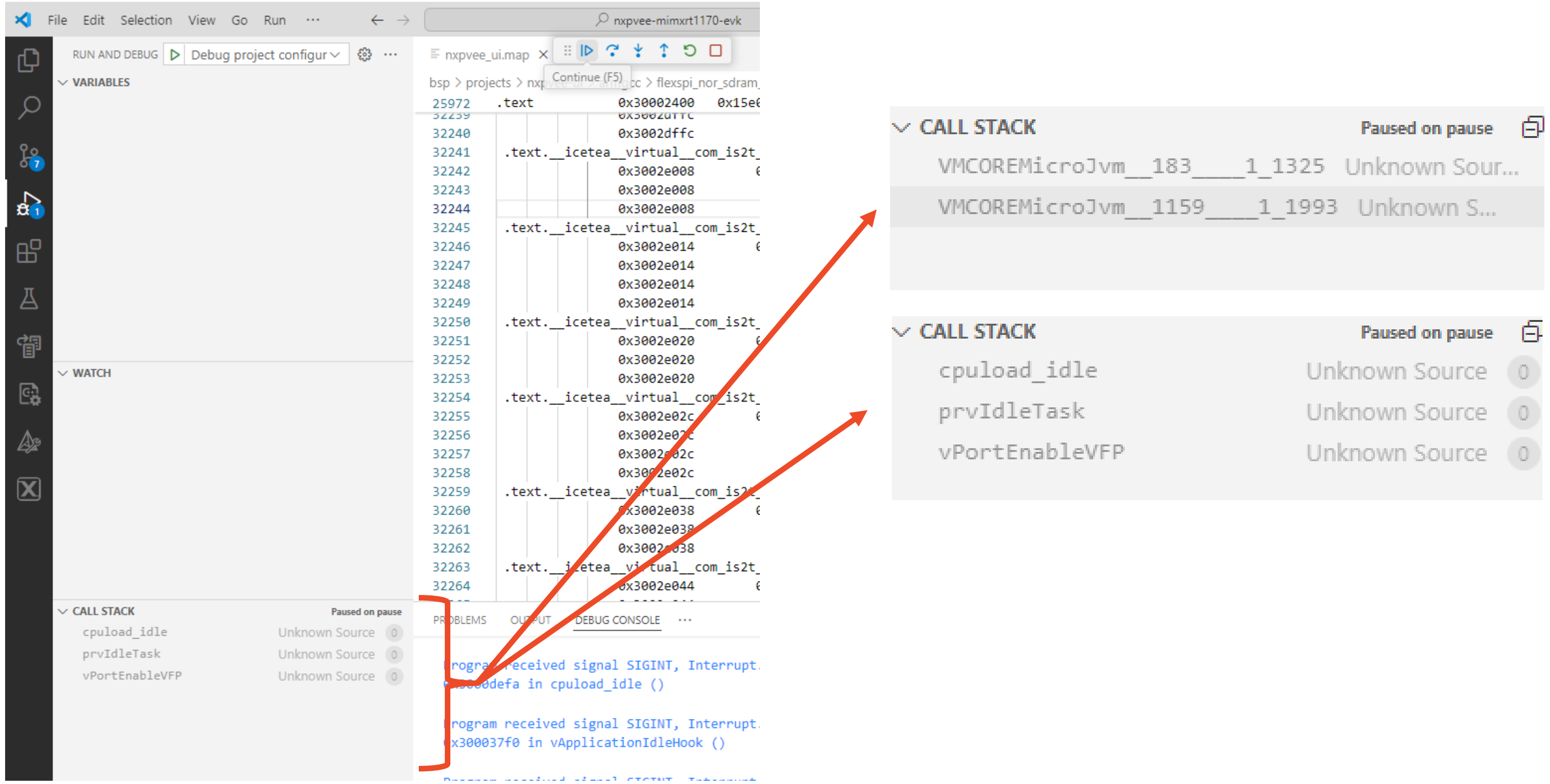
File Edit Selection View Go Run Terminal Help
nxpvee-mimxrt1170-evk
MCUXPRESSO FOR VS CODE
QUICKSTART PANEL
Import Repository
Import Example from Repository
Import Project
Application Code Hub
Flash Programmer
Open MCUXpresso Installer
INSTALLED REPOSITORIES
In order to start developing your first MCUXpresso application, you should first have a repository installed.
Import Repository
PROJECTS
nxpvee-mimxrt1170-evk
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...

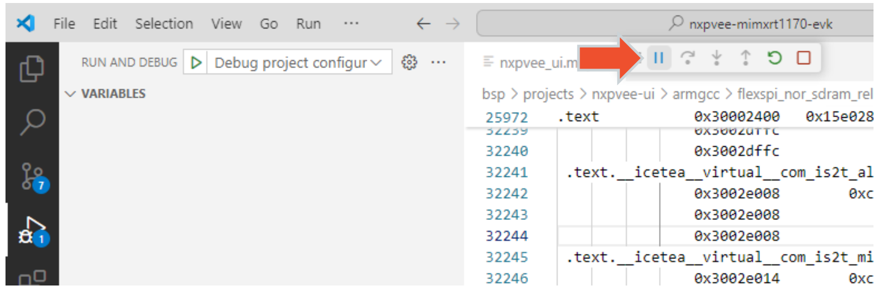
CALL STACK — Paused on pause

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

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:

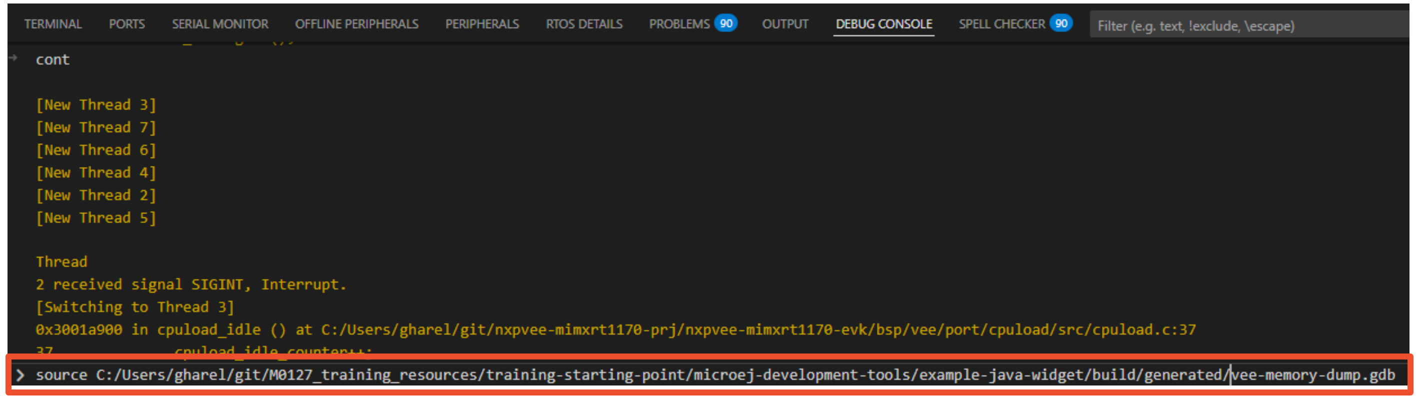


File Edit Selection View Go Run ···
nxpvee-mimxrt1170-evk
RUN AND DEBUG
Debug project configur
VARIABLES
nxpvee_ui.m
bsp > projects > nxpvee-ui > armgcc > flexspi_nor_sdram_rel

```
.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:

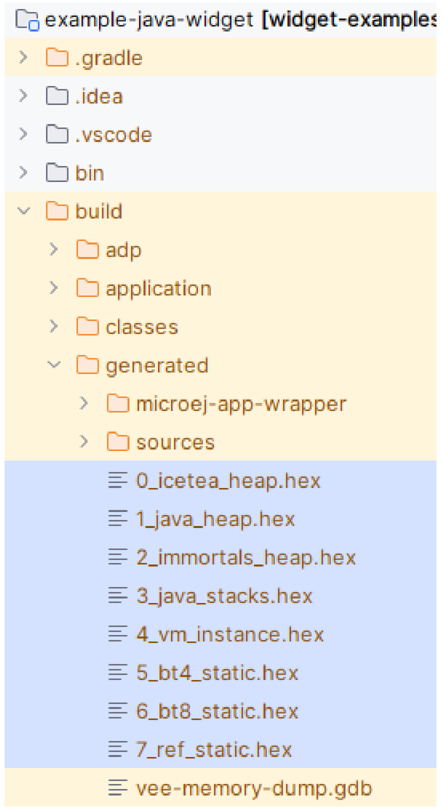
```
source C:/[YOUR_PATH]/vee-memory-dump.gdb
```



```
cont

[New Thread 3]
[New Thread 7]
[New Thread 6]
[New Thread 4]
[New Thread 2]
[New Thread 5]

Thread
2 received signal SIGINT, Interrupt.
[Switching to Thread 3]
0x3001a900 in cpuload_idle () at C:/Users/gharel/git/nxpvee-mimxrt1170-prj/nxpvee-mimxrt1170-evk/bsp/vee/port/cpuload/src/cpuload.c:37
37              cpuload_idle_counter++;
> source C:/Users/gharel/git/M0127_training_resources/training-starting-point/microej-development-tools/example-java-widget/build/generated/vee-memory-dump.gdb
```

⚠ Be aware of the separator used.

3. Heap dumps are generated in the output folder:

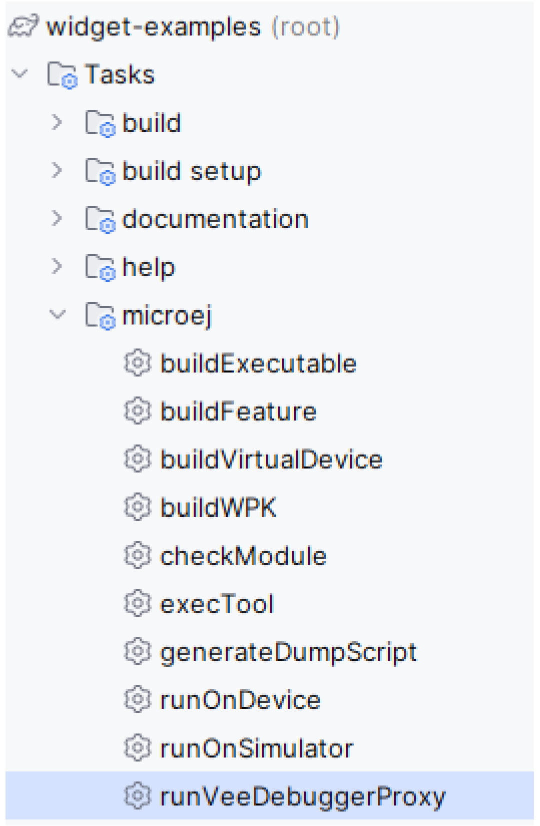


```
example-java-widget [widget-examples
> .gradle
> .idea
> .vscode
> bin
v build
  > adp
  > application
  > classes
  v generated
    > microej-app-wrapper
    > sources
      0_icetea_heap.hex
      1_java_heap.hex
      2_immortals_heap.hex
      3_java_stacks.hex
      4_vm_instance.hex
      5_bt4_static.hex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

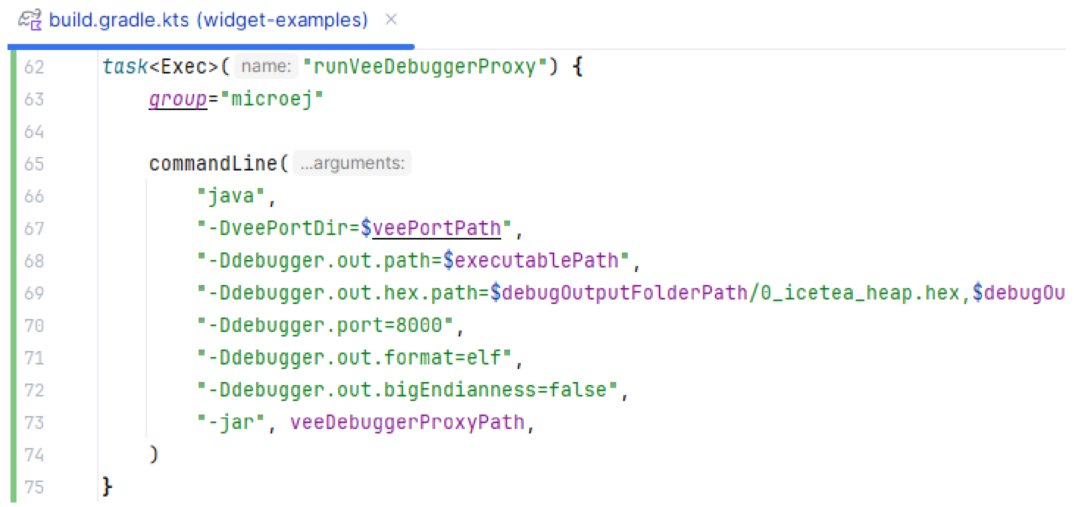
RUN THE VEE DEBUGGER PROXY (1/2)

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)

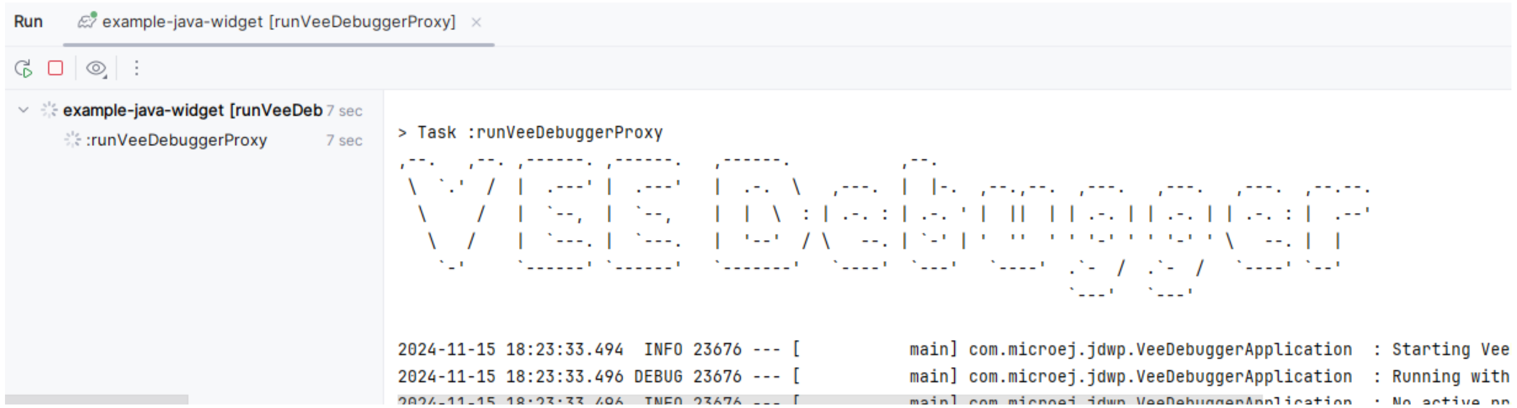
```
task<Exec>( name: "runVeeDebuggerProxy") {
    group="microej"

    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:

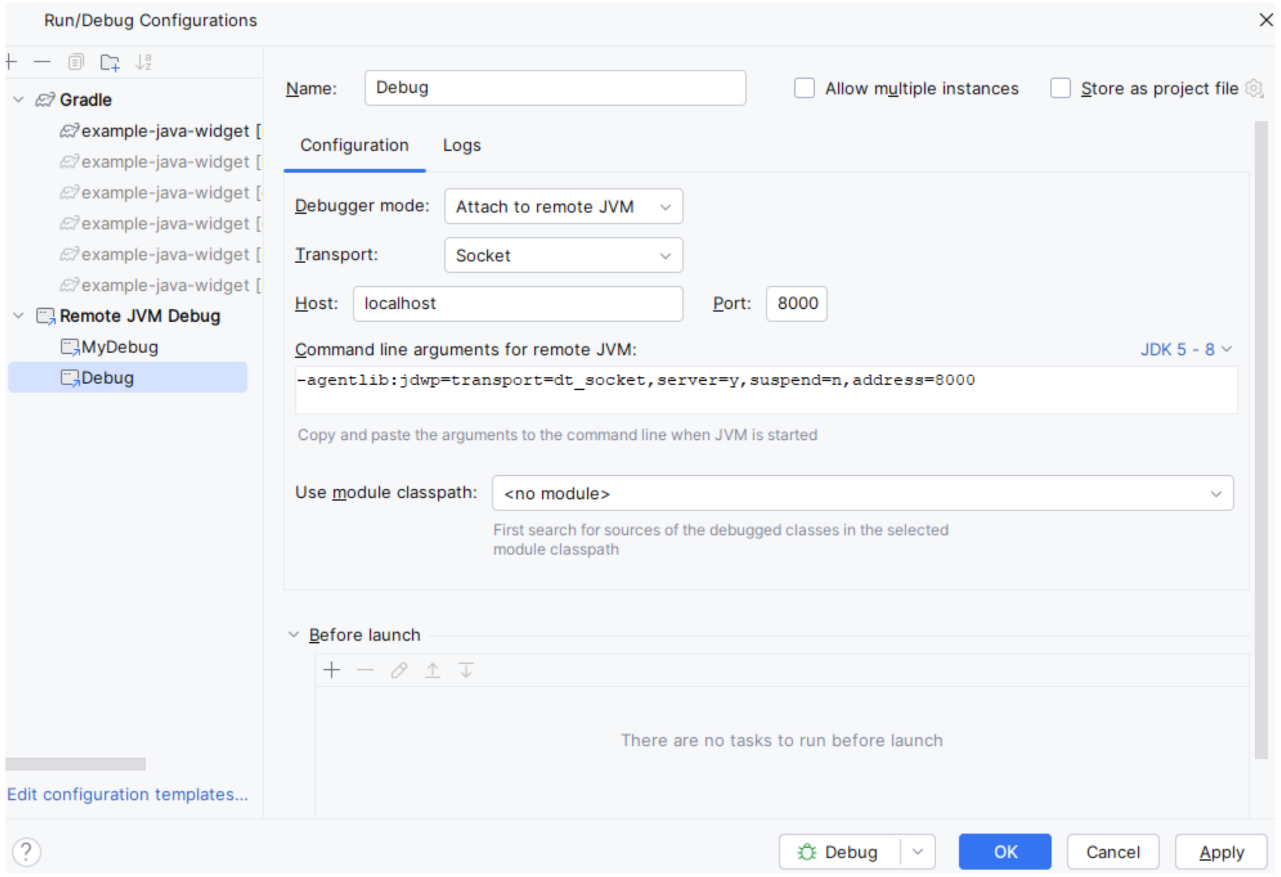

Run
example-java-widget [runVeeDebuggerProxy]
example-java-widget [runVeeDeb 7 sec
:runVeeDebuggerProxy 7 sec

```
> 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

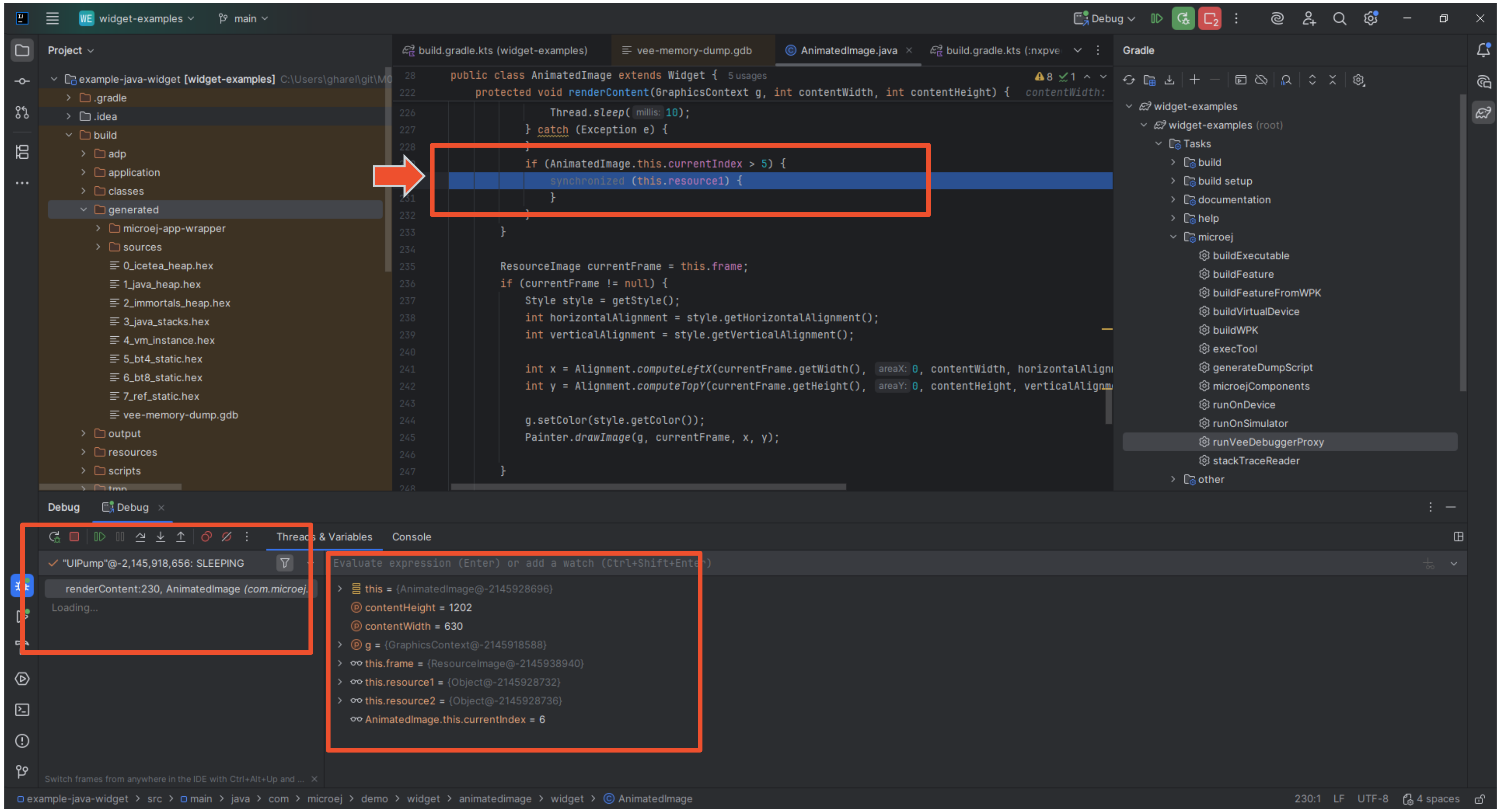
Use module classpath: <no module>

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)

Click on the pause button and the following debug state **and the thread call stack** can be seen:



```
example-java-widget [widget-examples] C:\Users\gharel\git\M
  .gradle
  .idea
  build
    adp
    application
    classes
    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
    output
    resources
    scripts
```

```java
public class AnimatedImage extends Widget {   5 usages
    protected void renderContent(GraphicsContext g, int contentWidth, int contentHeight) {   contentWidth:
                Thread.sleep( millis: 10);
            } catch (Exception e) {
            }
            if (AnimatedImage.this.currentIndex > 5) {
                synchronized (this.resource1) {
                }
            }
        }

        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, horizontalAlign
            int y = Alignment.computeTopY(currentFrame.getHeight(),  areaY: 0, contentHeight, verticalAlignm

            g.setColor(style.getColor());
            Painter.drawImage(g, currentFrame, x, y);

        }
```

```
widget-examples
  widget-examples (root)
    Tasks
      build
      build setup
      documentation
      help
      microej
        buildExecutable
        buildFeature
        buildFeatureFromWPK
        buildVirtualDevice
        buildWPK
        execTool
        generateDumpScript
        microejComponents
        runOnDevice
        runOnSimulator
        runVeeDebuggerProxy
        stackTraceReader
      other
```

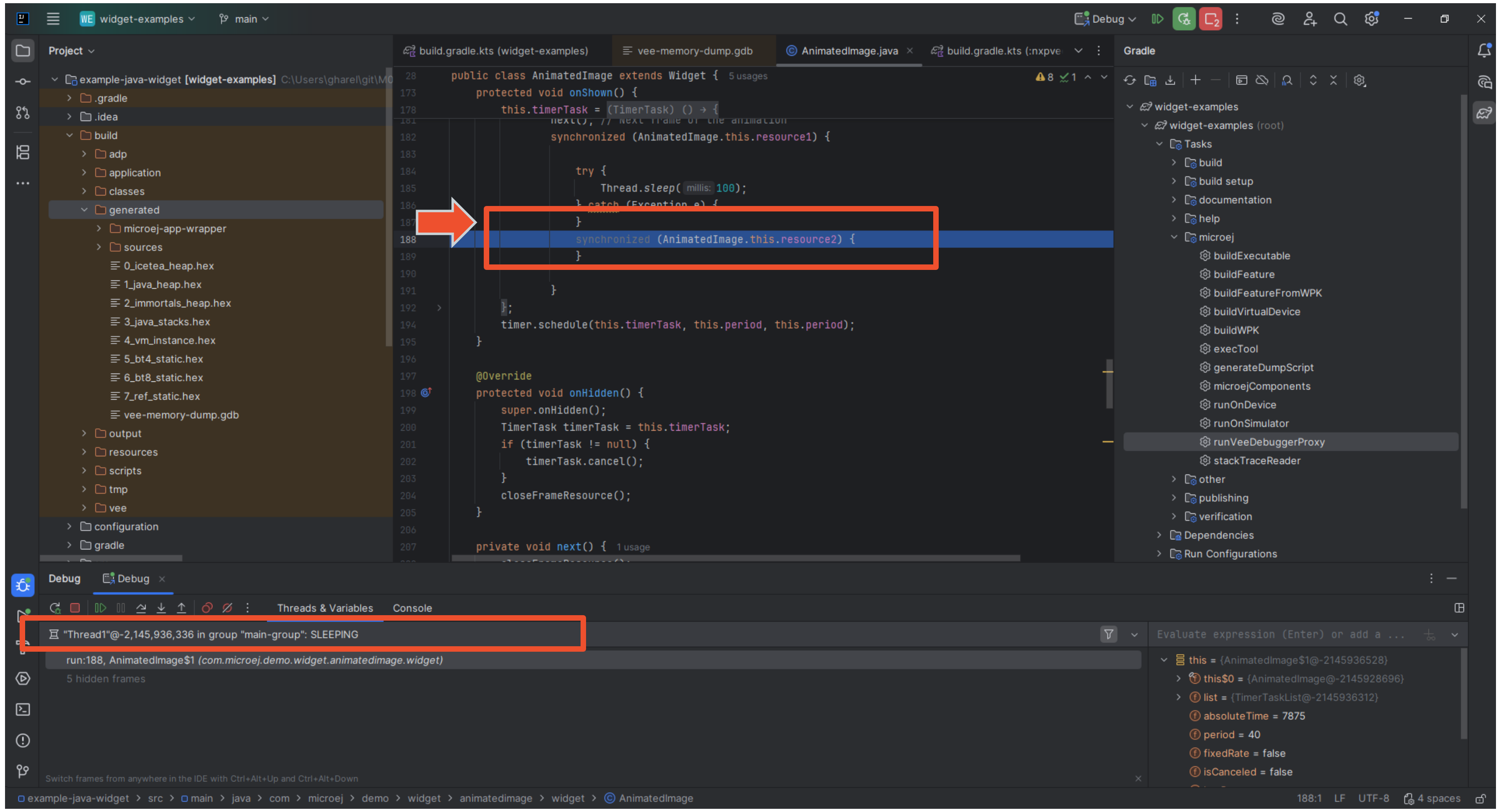
```
"UIPump"@-2,145,918,656: SLEEPING
renderContent:230, AnimatedImage (com.microej.
Loading...
```

```
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
this = {AnimatedImage@-2145928696}
contentHeight = 1202
contentWidth = 630
g = {GraphicsContext@-2145918588}
this.frame = {ResourceImage@-2145938940}
this.resource1 = {Object@-2145928732}
this.resource2 = {Object@-2145928736}
AnimatedImage.this.currentIndex = 6
```

```
example-java-widget > src > main > java > com > microej > demo > widget > animatedimage > widget > AnimatedImage        230:1  LF  UTF-8  4 spaces
```

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

The 2^{nd} Thread state can also be seen:



```java
public class AnimatedImage extends Widget {  5 usages
    protected void onShown() {
        this.timerTask = (TimerTask) () → {
                next(); // Next frame of the animation
                synchronized (AnimatedImage.this.resource1) {

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

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

    @Override
    protected void onHidden() {
        super.onHidden();
        TimerTask timerTask = this.timerTask;
        if (timerTask != null) {
            timerTask.cancel();
        }
        closeFrameResource();
    }

    private void next() {  1 usage
```

```
"Thread1"@-2,145,936,336 in group "main-group": SLEEPING
run:188, AnimatedImage$1 (com.microej.demo.widget.animatedimage.widget)
5 hidden frames
Switch frames from anywhere in the IDE with Ctrl+Alt+Up and Ctrl+Alt+Down
```

```
Evaluate expression (Enter) or add a ...
this = {AnimatedImage$1@-2145936528}
  this$0 = {AnimatedImage@-2145928696}
  list = {TimerTaskList@-2145936312}
  absoluteTime = 7875
  period = 40
  fixedRate = false
  isCanceled = false
```

example-java-widget > src > main > java > com > microej > demo > widget > animatedimage > widget > AnimatedImage   188:1 LF UTF-8 4 spaces

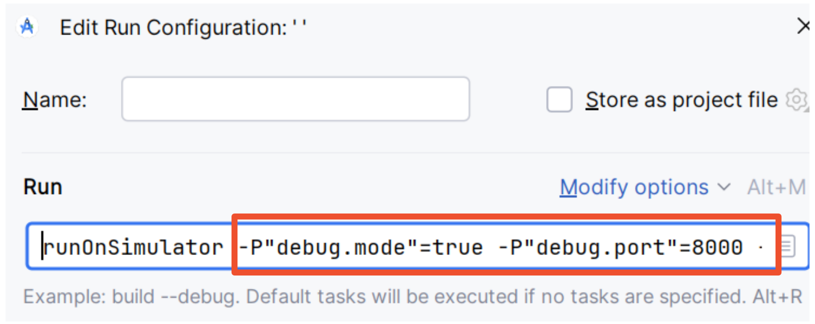
Debug on Simulator

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

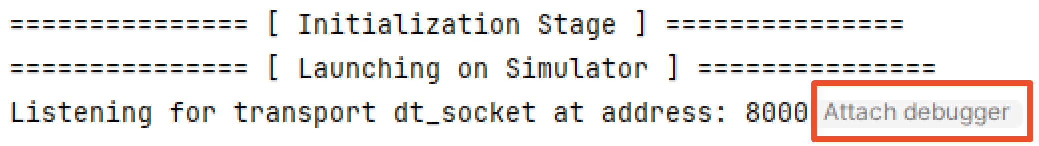
Run

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

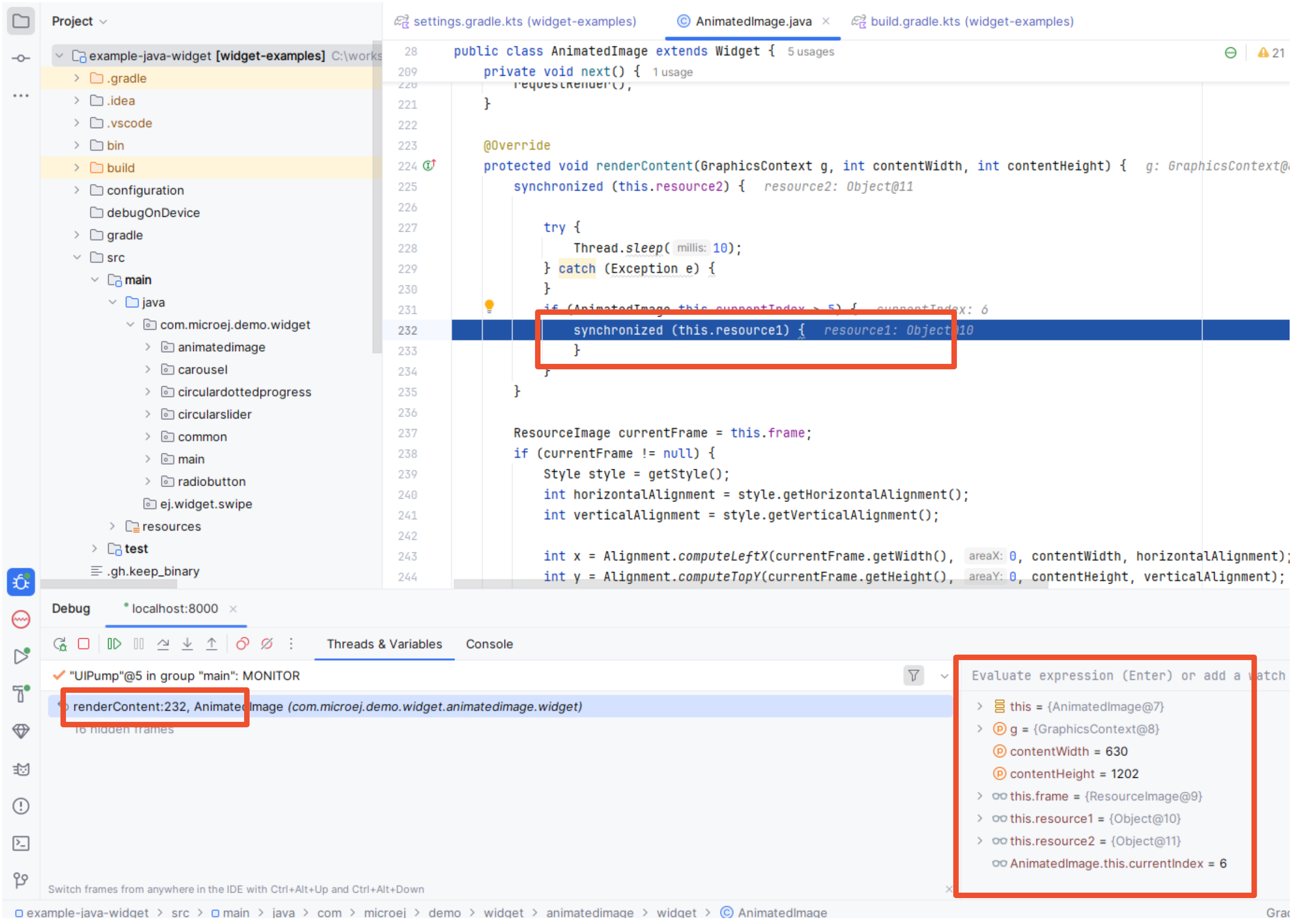
- The console opens with the following message, click on Attach debugger to start the debug session:



```
=============== [ 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



Project

- example-java-widget [widget-examples] C:\works
  - .gradle
  - .idea
  - .vscode
  - bin
  - build
  - configuration
  - debugOnDevice
  - gradle
  - src
    - main
      - java
        - com.microej.demo.widget
          - animatedimage
          - carousel
          - circulardottedprogress
          - circularslider
          - common
          - main
          - radiobutton
        - ej.widget.swipe
      - resources
    - test
  - .gh.keep_binary

settings.gradle.kts (widget-examples) | AnimatedImage.java | build.gradle.kts (widget-examples)

```
public class AnimatedImage extends Widget {   5 usages
    private void next() {   1 usage
        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);
```

Debug | localhost:8000

Threads & Variables | Console

```
"UIPump"@5 in group "main": MONITOR
renderContent:232, AnimatedImage (com.microej.demo.widget.animatedimage.widget)
16 hidden frames
```

```
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
```

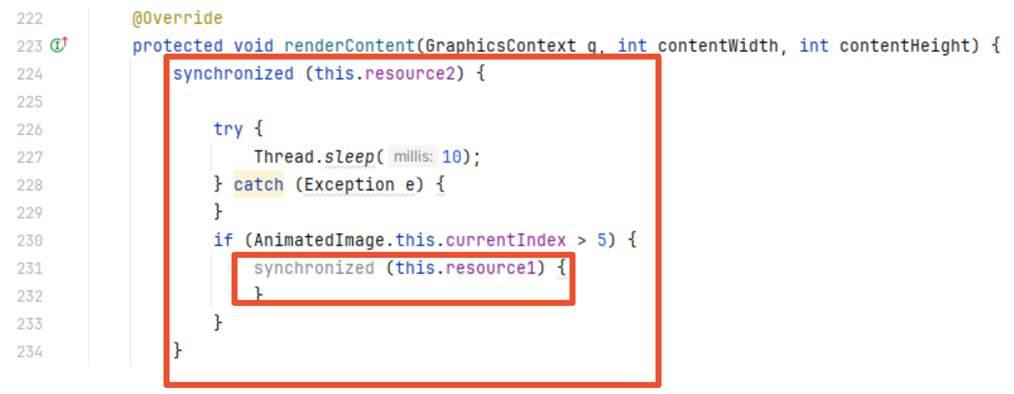
Switch frames from anywhere in the IDE with Ctrl+Alt+Up and Ctrl+Alt+Down

example-java-widget > src > main > java > com > microej > demo > widget > animatedimage > widget > AnimatedImage

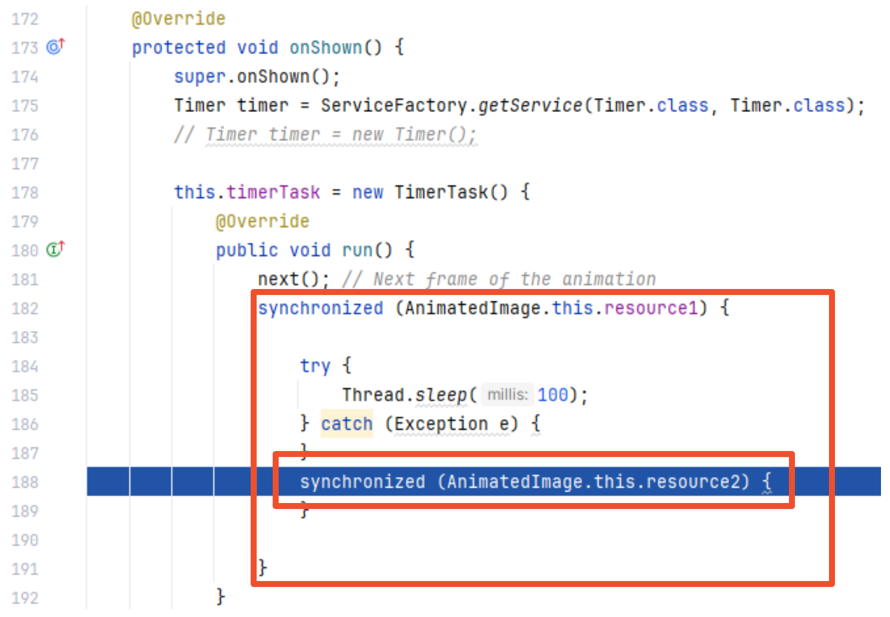
ISSUE ANALYSIS & FIX

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



```java
@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) {
            }
        }
    }
```



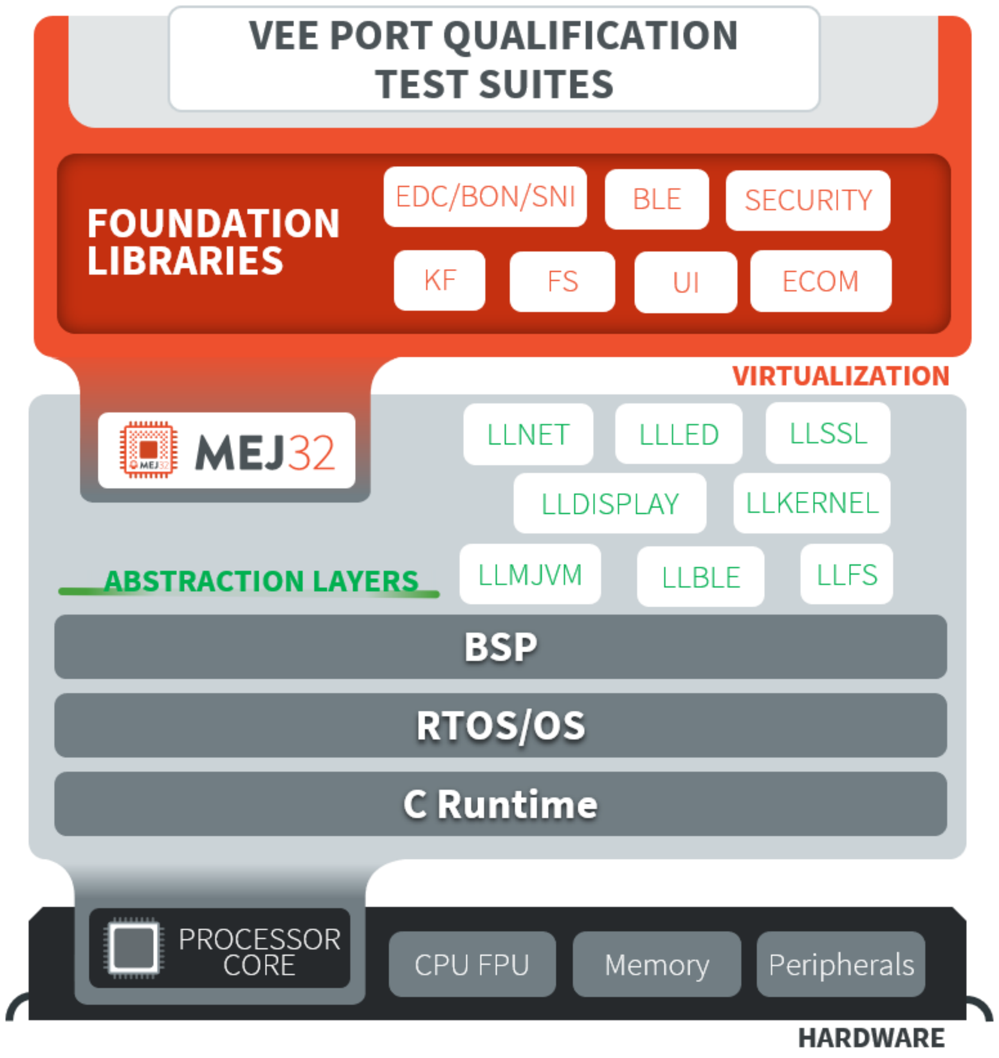
```java
@Override
protected void onShown() {
    super.onShown();
    Timer timer = ServiceFactory.getService(Timer.class, Timer.class);
    // Timer timer = new Timer();

    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.

VEE Port Qualification Tool

VEE PORT QUALIFICATION TOOL (1/3)

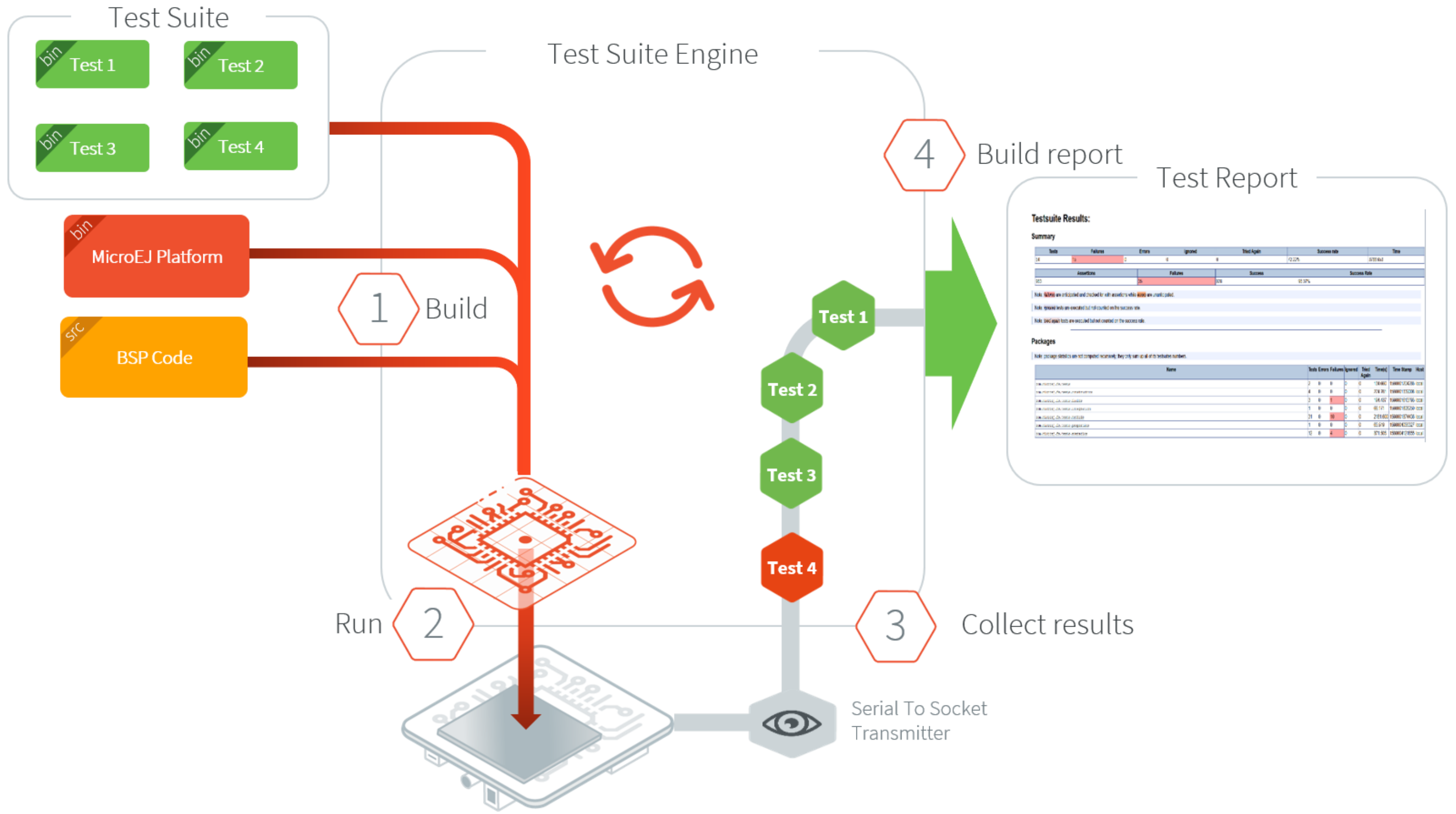
- The VEE Port Qualification Tool (PQT) project provides the tools required to validate each component of a MicroEJ VEE Port.
- After porting or adding a Foundation Library to a MicroEJ VEE Port, it is necessary to validate its integration.
- For each Low Level API, an Abstraction Layer implementation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



VEE PORT QUALIFICATION TOOL (2/3)

- 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

VEE PORT QUALIFICATION TOOL (3/3)

- The VEE Port Template GitHub repository holds the configuration necessary to pass tests on VEE ports:
  - https://github.com/MicroEJ/Tool-Project-Template-VEEPort/tree/master/vee-port/validation

- The validation folder provides configuration files for each test suite.

- Execute the **test** Gradle task on the Test Suite Project either in Command Line or via the IDE to launch the corresponding test suite.

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.)

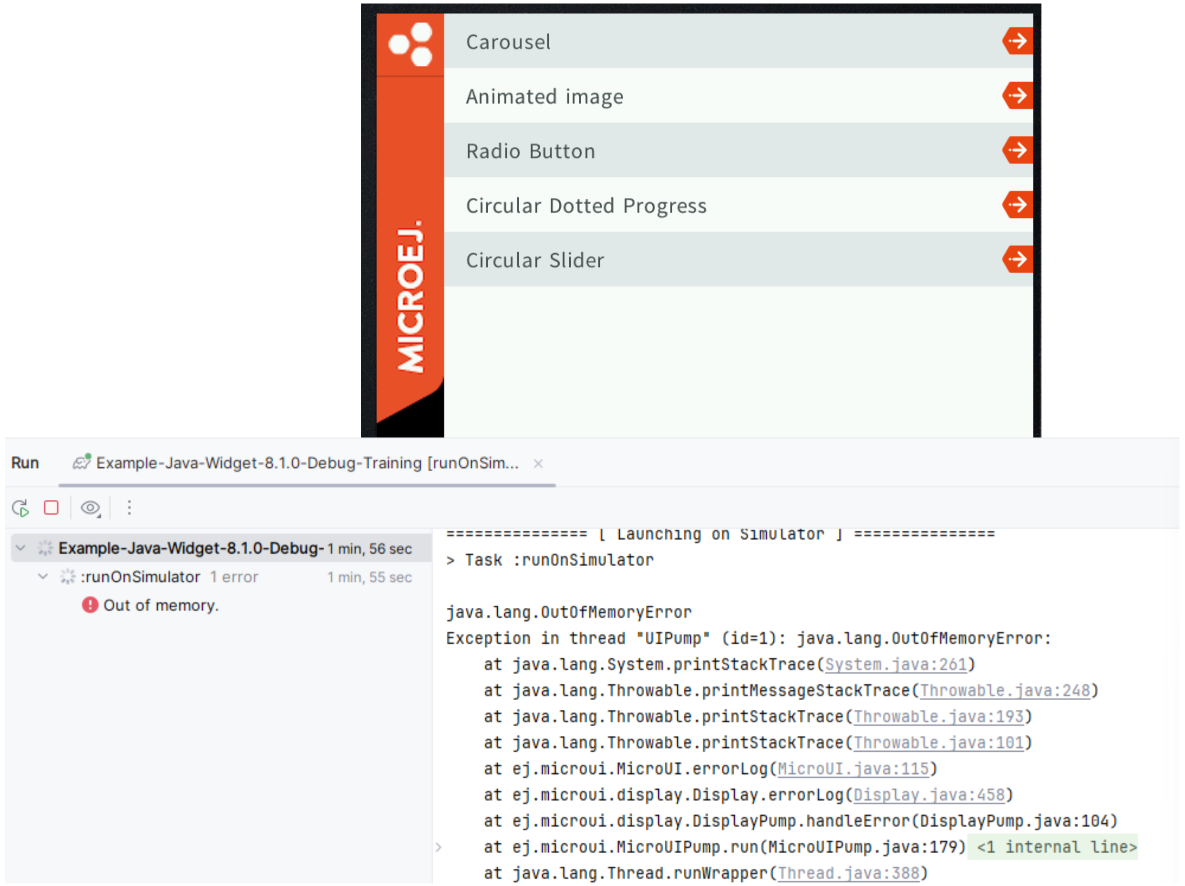
4. Debugger (on device & simulator): analysis of an applicative issue

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)
```

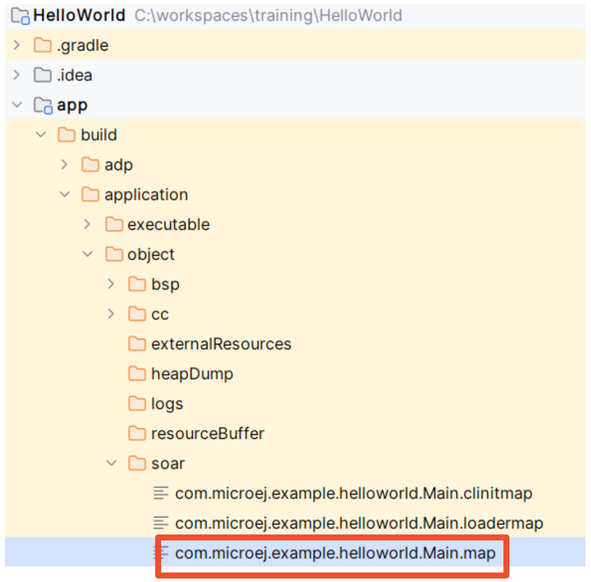
Out Of Memory exception in a GUI application

* 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:



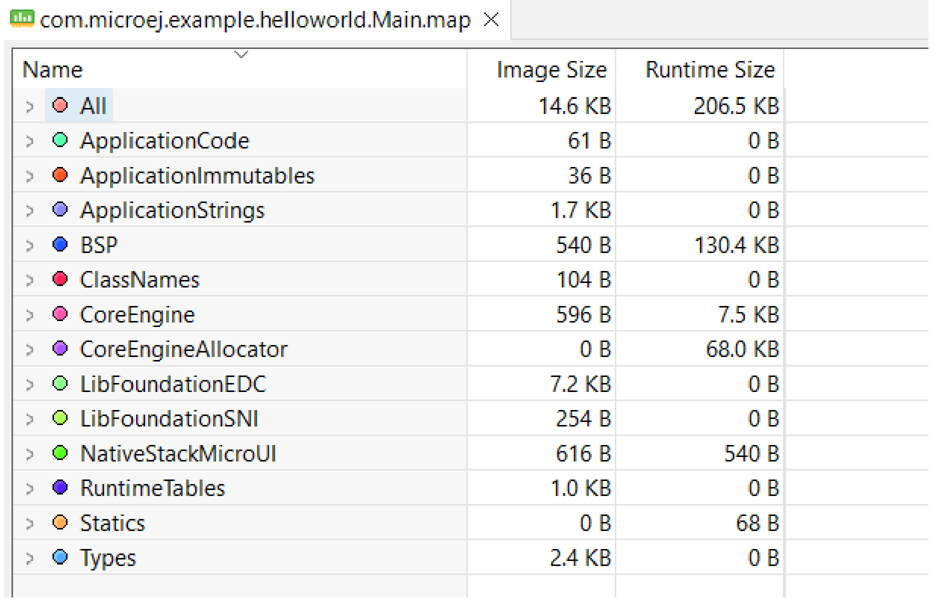
```
HelloWorld C:\workspaces\training\HelloWorld
> .gradle
> .idea
v app
  v build
    > adp
    v application
      > executable
      v object
        > bsp
        > cc
          externalResources
          heapDump
          logs
          resourceBuffer
        v soar
            com.microej.example.helloworld.Main.clinitmap
            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 (or MicroEJ native code). Only the content of microejapp.o is displayed.

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):

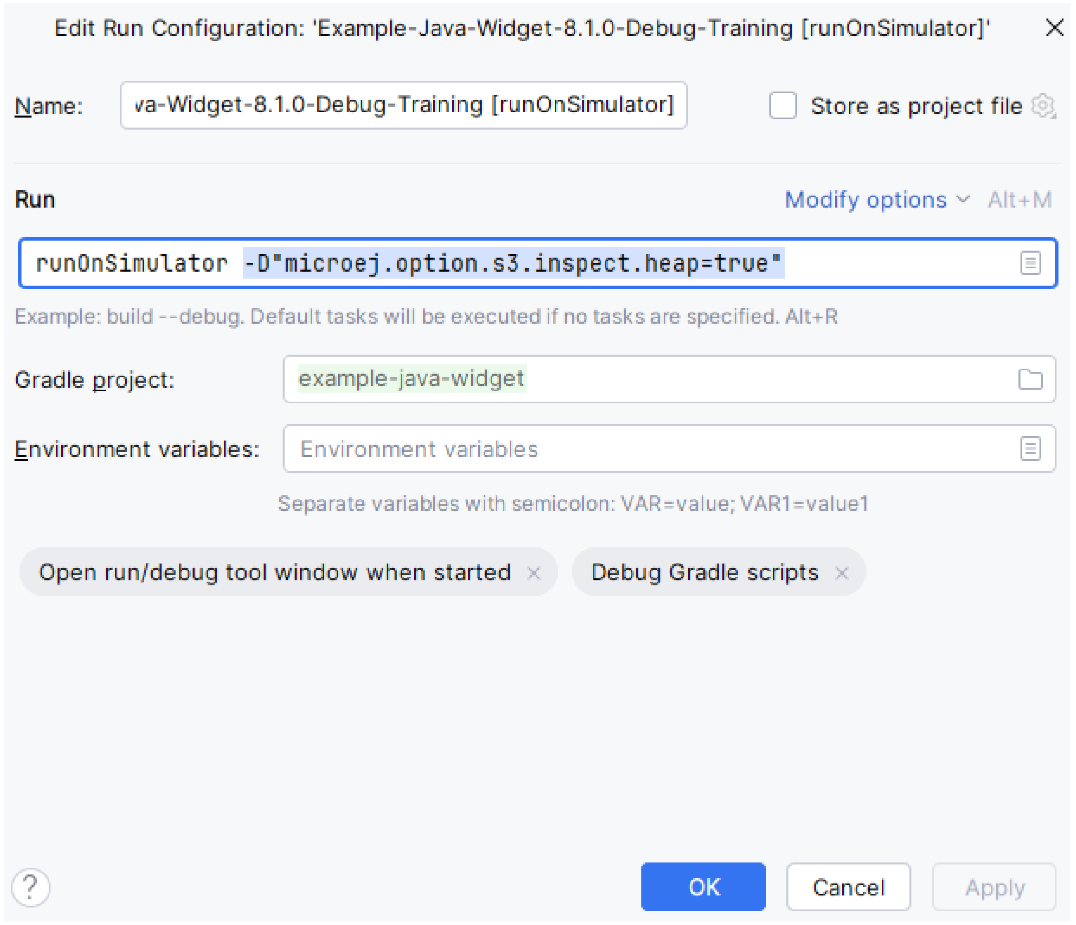
- Right-Click on the **runOnSimulator** task.
- Click on **Modify Run Configuration…**
- Add the following option:

```
-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   Modify options Alt+M

```
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

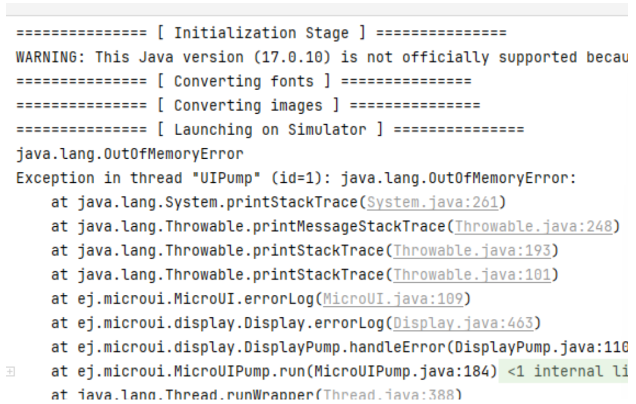
Separate variables with semicolon: VAR=value; VAR1=value1

Run Configuration options

HEAP DUMPER & HEAP ANALYZER (3/8)

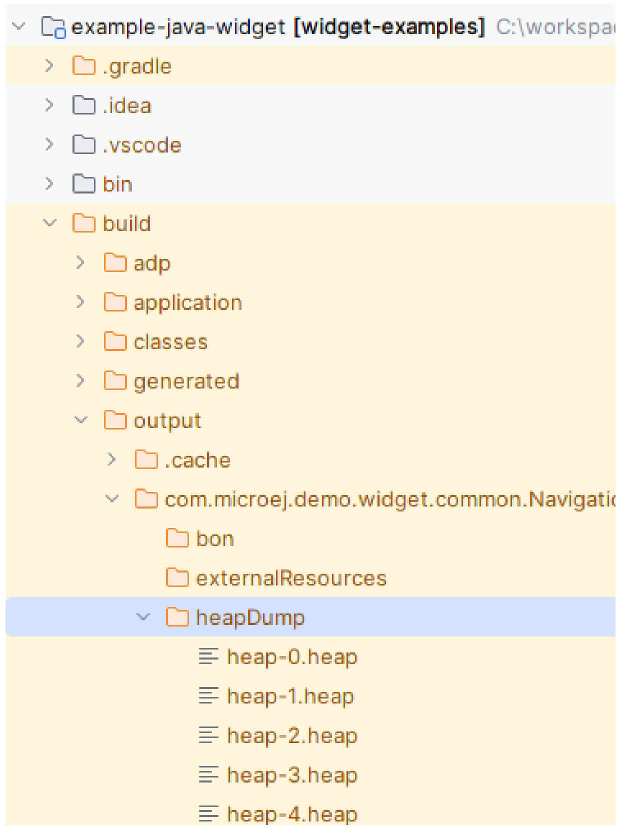
REPRODUCE THE ISSUE

- 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

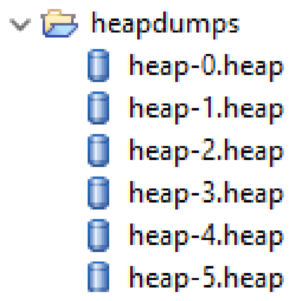
HEAP DUMPER & HEAP ANALYZER (4/8)

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

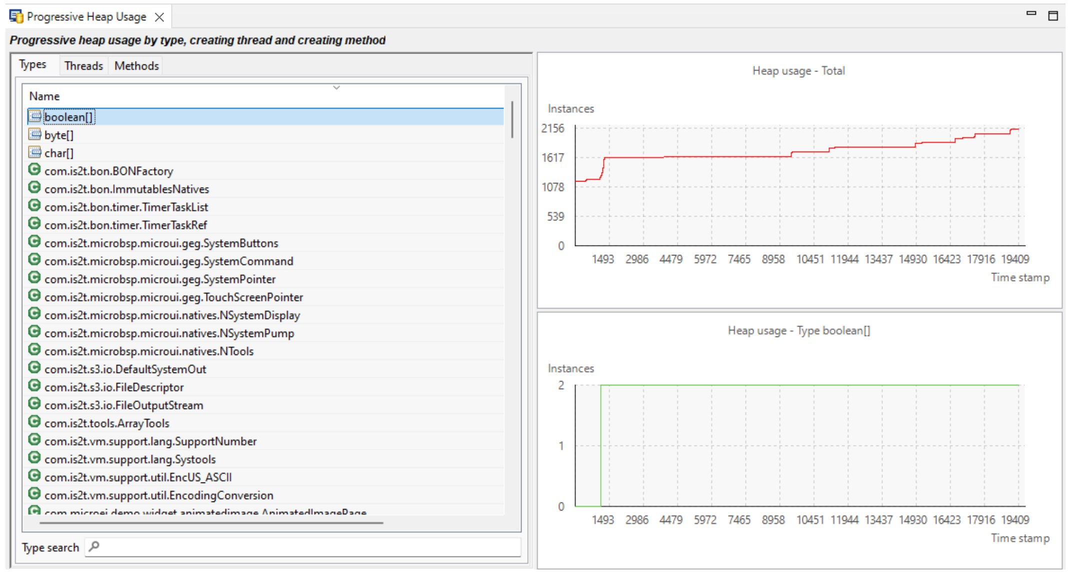
HEAP DUMPER & HEAP ANALYZER (5/8)

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

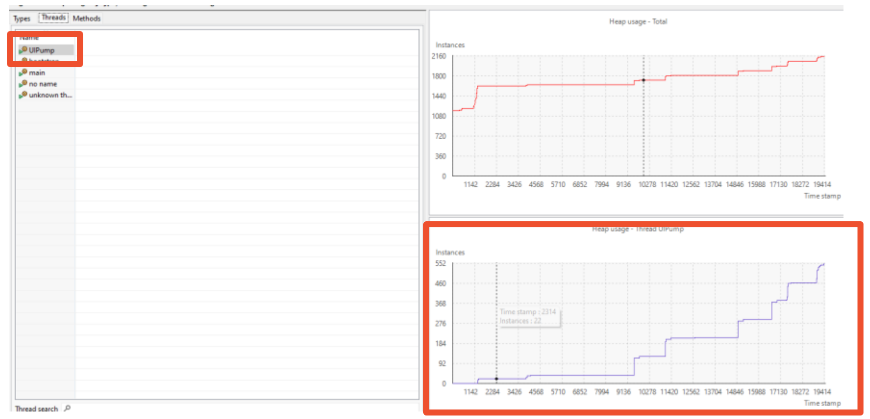
Progressive heap usage by type, creating thread and creating method

Types | Threads | Methods

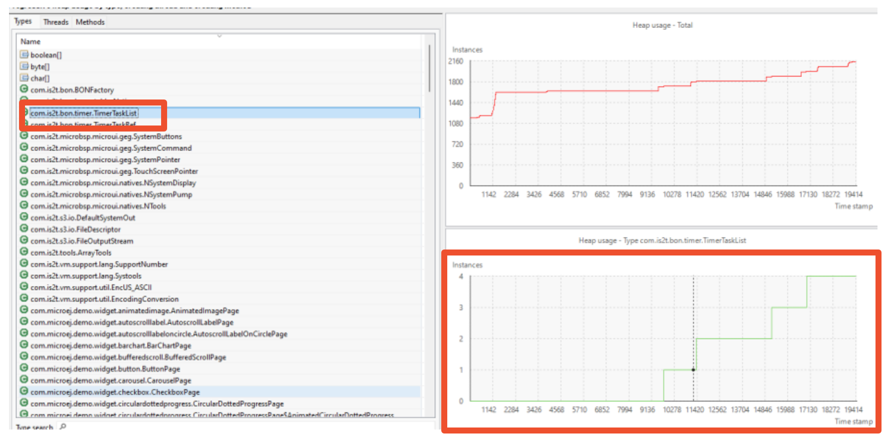
Name
- boolean[]
- byte[]
- char[]
- com.is2t.bon.BONFactory
- com.is2t.bon.ImmutablesNatives
- com.is2t.bon.timer.TimerTaskList
- com.is2t.bon.timer.TimerTaskRef
- com.is2t.microbsp.microui.geg.SystemButtons
- com.is2t.microbsp.microui.geg.SystemCommand
- com.is2t.microbsp.microui.geg.SystemPointer
- 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**:



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

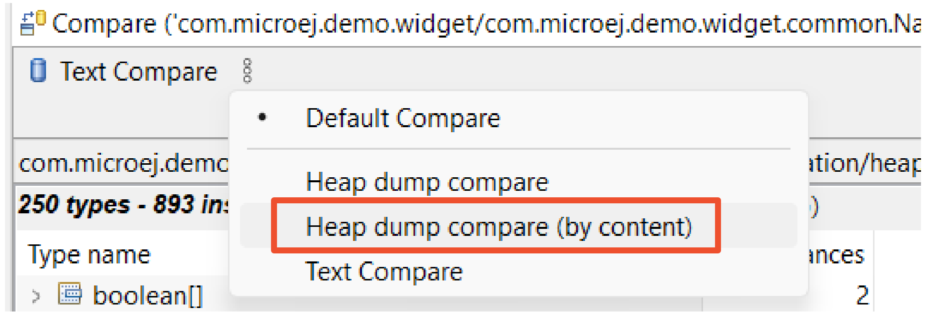


Next step: compare 2 consecutive heap dumps focusing the types that are growing continuously.

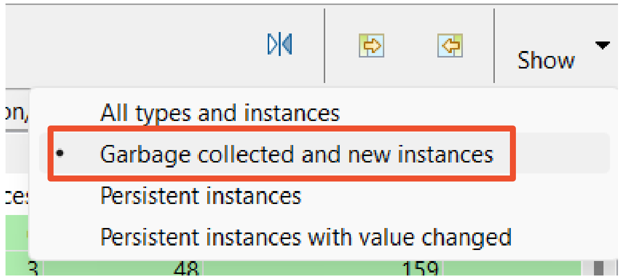
HEAP DUMPER & HEAP ANALYZER (6/8)

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:

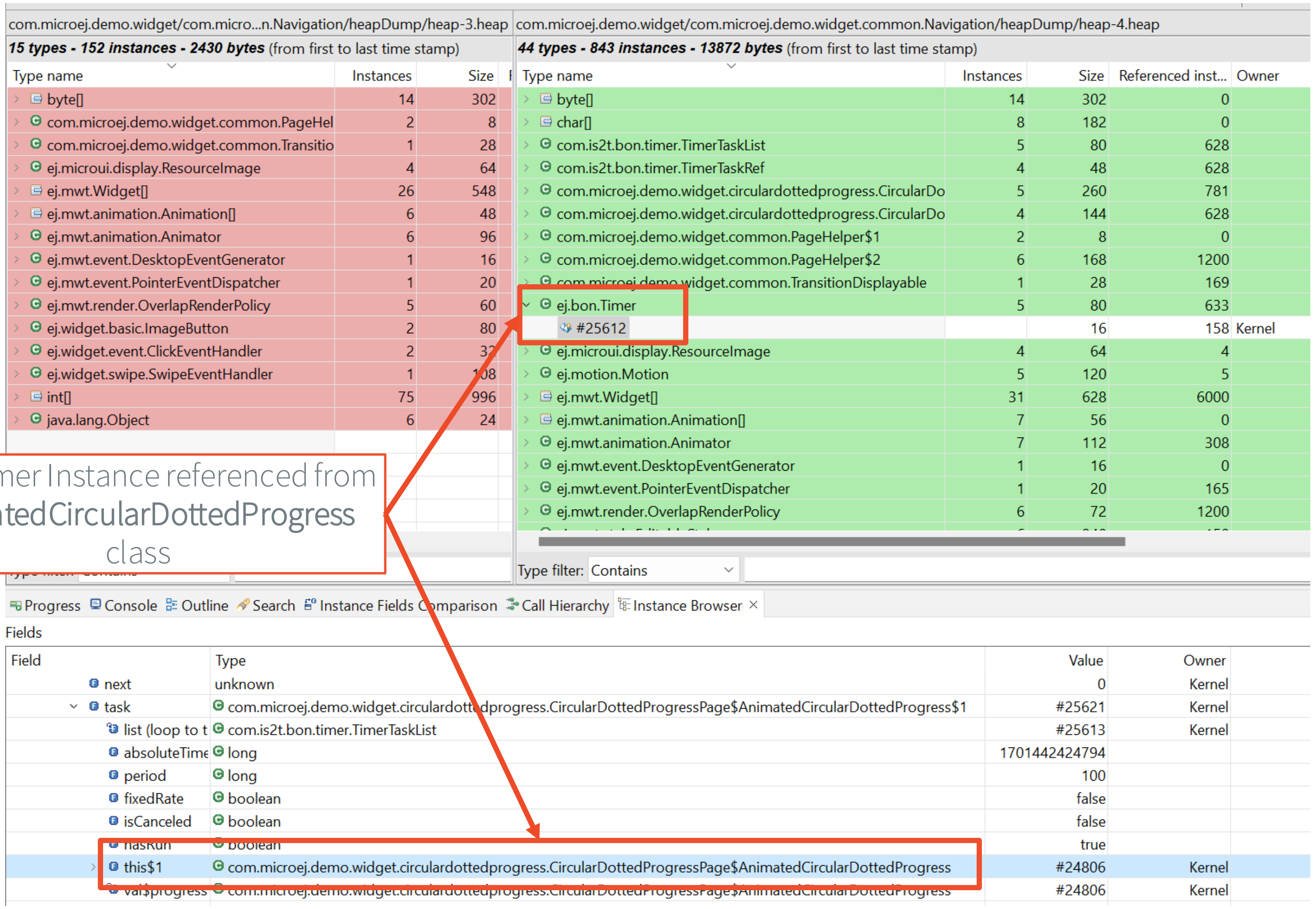

Compare ('com.microej.demo.widget/com.microej.demo.widget.common.Na
Text Compare
- Default Compare
- Heap dump compare
- Heap dump compare (by content)
- Text Compare

com.microej.demo
250 types - 893 ins
Type name
boolean[]


Show
- All types and instances
- • Garbage collected and new instances
- Persistent instances
- Persistent instances with value changed

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

Progress Console Outline Search Instance Fields Comparison Call Hierarchy Instance Browser

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:

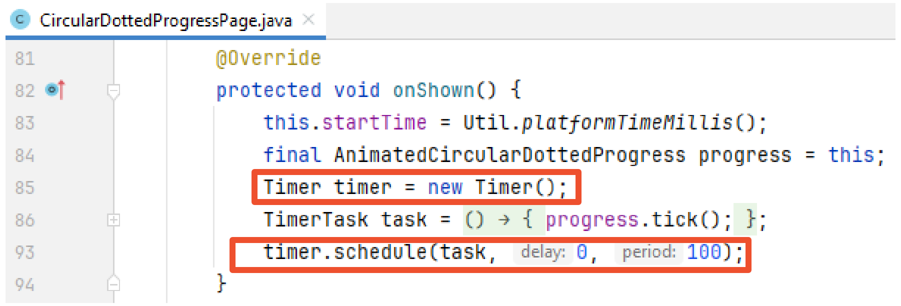
- Lots of new objects have been created (691 new instances)
- Use the **compare by content** option to discard objects that moved but have the same content
- Look for new objects that can have an impact (Thread, Timer, Page, Widget, StyleSheet)

  → 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:



```
@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.

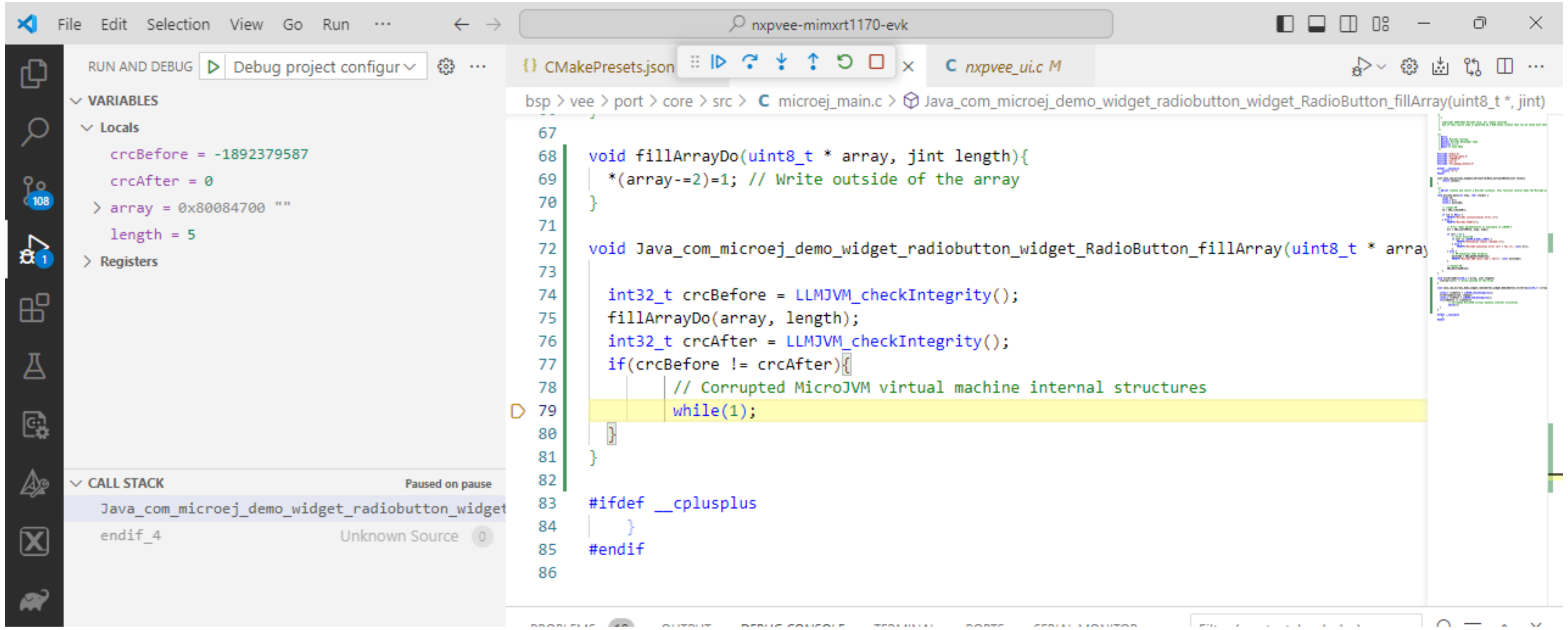
- This feature is for Applications deployed on hardware devices only:
  - If an integrity error is detected, the **LLMJVM_on_CheckIntegrity_error** hook is called and this method returns 0.
  - If no integrity error is detected, a non-zero checksum is returned.

- 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:



File Edit Selection View Go Run
nxpvee-mimxrt1170-evk
RUN AND DEBUG
Debug project configur
CMakePresets.json
nxpvee_ui.c M
bsp > vee > port > core > src > microej_main.c > Java_com_microej_demo_widget_radiobutton_widget_RadioButton_fillArray(uint8_t *, jint)

```
VARIABLES
  Locals
    crcBefore = -1892379587
    crcAfter = 0
  > array = 0x80084700 ""
    length = 5
  > Registers

CALL STACK                                  Paused on pause
  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.

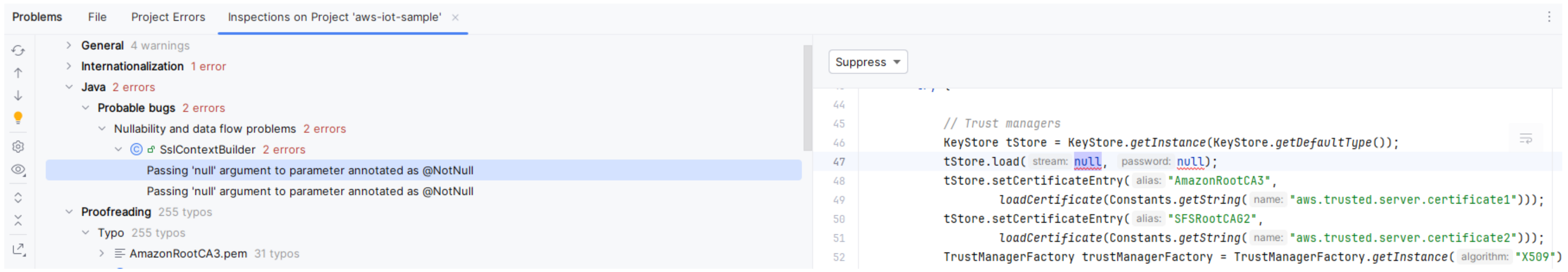
Static Analysis Tools

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.



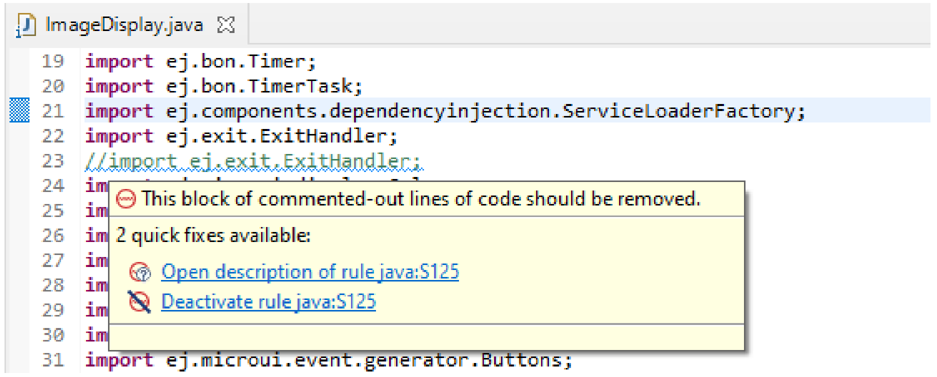
```
> General 4 warnings
> Internationalization 1 error
v Java 2 errors
  v Probable bugs 2 errors
    v Nullability and data flow problems 2 errors
      v SslContextBuilder 2 errors
          Passing 'null' argument to parameter annotated as @NotNull
          Passing 'null' argument to parameter annotated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

```java
        // 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;
```

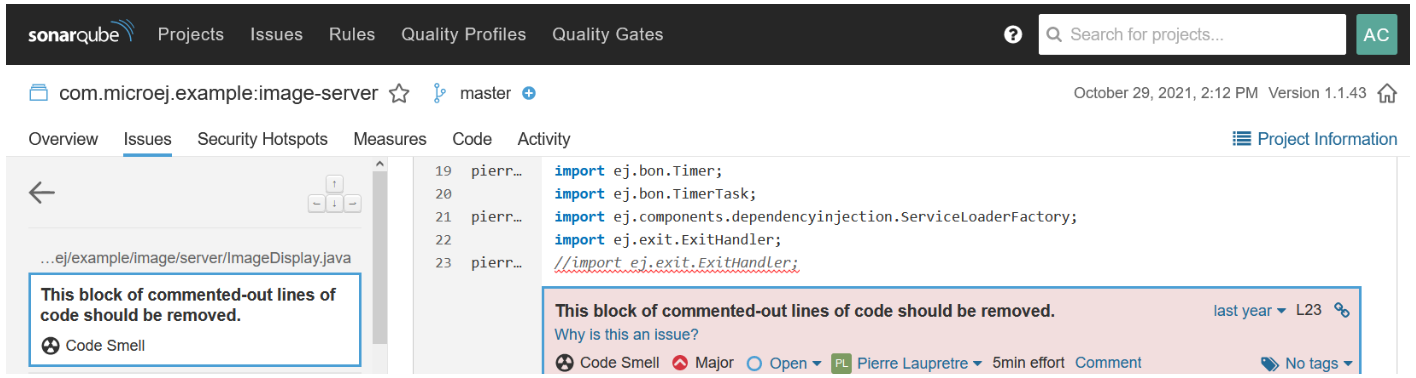
This block of commented-out lines of code should be removed.

2 quick fixes available:

Open description of rule java:S125

Deactivate rule java:S125

SonarQube code analysis performed inside Eclipse IDE



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

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

Code Smell

```
19  pierr…  import ej.bon.Timer;
20          import ej.bon.TimerTask;
21  pierr…  import ej.components.dependencyinjection.ServiceLoaderFactory;
22          import ej.exit.ExitHandler;
23  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

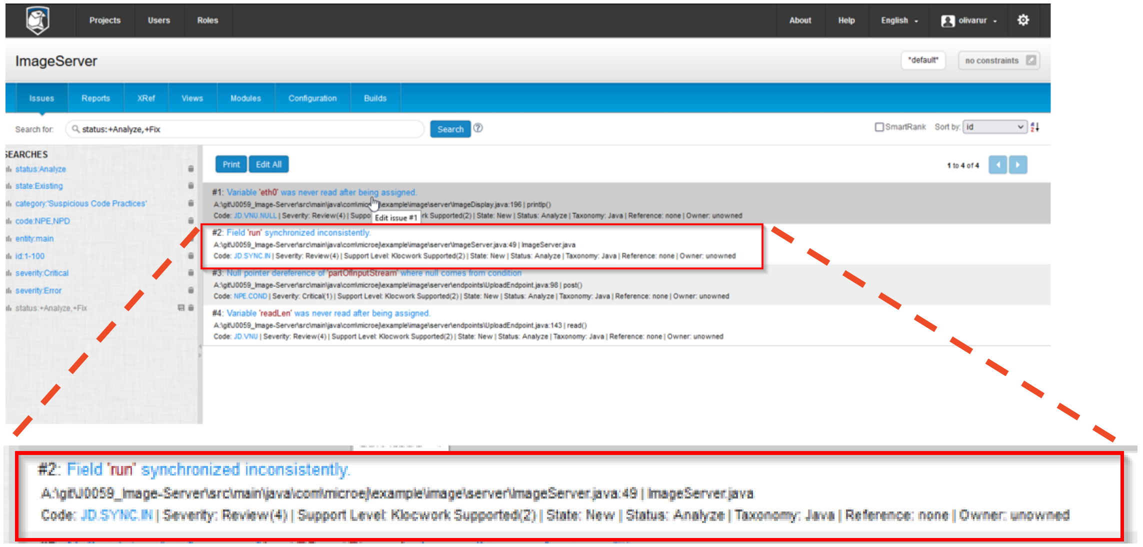
SonarQube code analysis performed on SonarQube server

STATIC ANALYSIS TOOLS (3/3)

MICROEJ

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.



Search for: status:+Analyze,+Fix Sort by: id

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

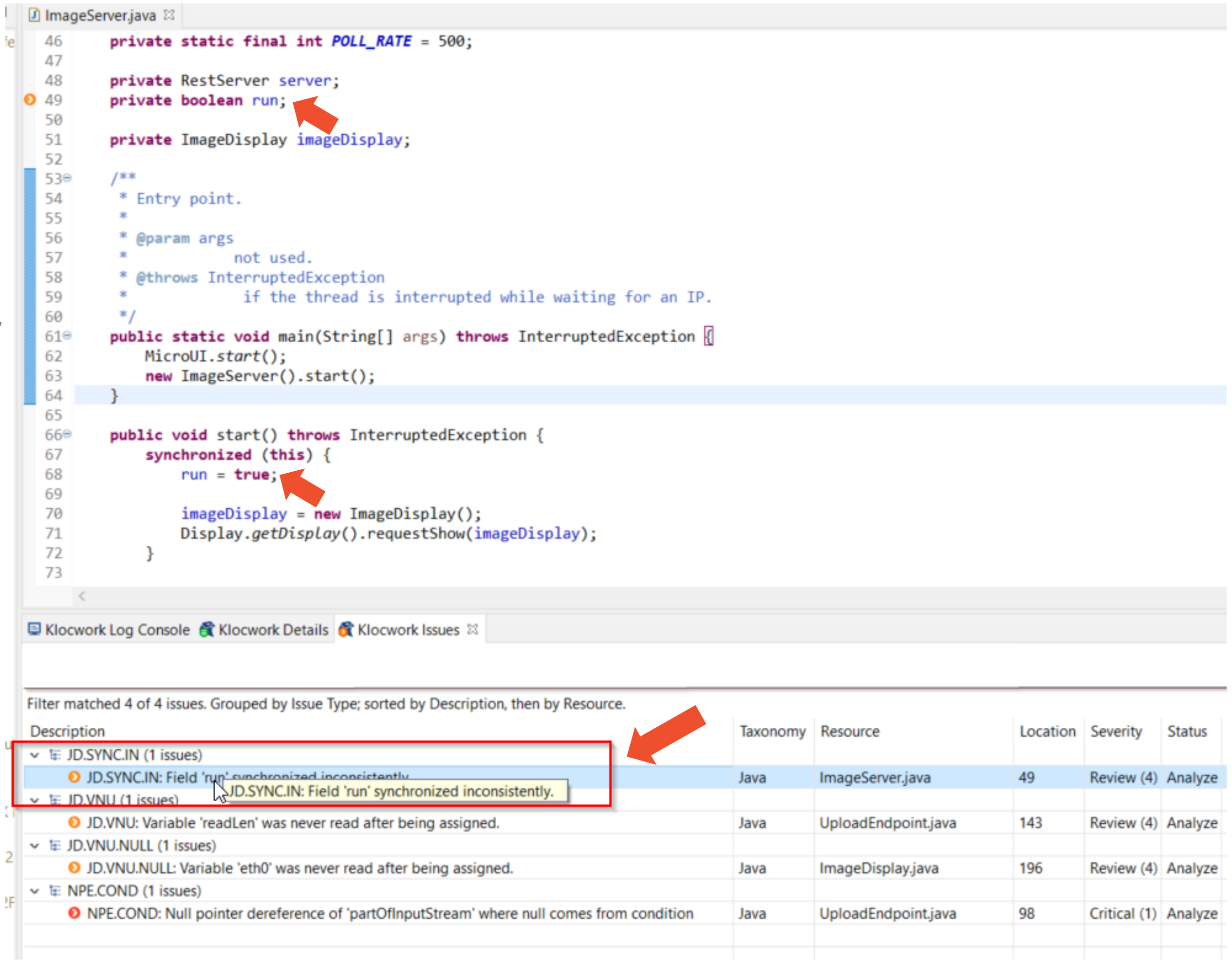
#1: Variable 'eth0' was never read after being assigned.
A:\git\J0059_Image-Server\src\main\java\com\microej\example\image\server\ImageDisplay.java:196 | println()
Code: JD.VNU.NULL | Severity: Review(4) | Support Level: Klocwork Supported(2) | State: New | Status: Analyze | Taxonomy: Java | Reference: 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

#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

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

Klocwork code analysis performed inside Eclipse IDE

FOLLOW-UP: UI SPECIFIC TOOLS AND SYSTEMVIEW

- UI specific tools are explained in **Mastering UI Development Tools** training course. It is tailored towards UI application debugging and improvements:
  - GUI Performances Improvements (bottlenecks identification with SystemView)
  - GUI Rendering Issues Debug
- While centered around a UI profiling use case, the SystemView training teaches skills that could be applied around debugging any performance issues.

THANK YOU

for your attention !

MICROEJ®