MICROEJ®

SOFTWARE-DEFINED EVERYTHING

Mastering UI Development Tools

Design and Improve GUI applications.

DISCLAIMER

TABLE OF CONTENT

OVERVIEW

- Goal:
  - Provide an overview of the development tools provided to developers to debug specifically UI application.
  - Illustrate the use of these development tools.
- Development tools categories:
  - GUI Performances Improvements (bottlenecks identification)
  - GUI Rendering Issues Debug

- 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.

PREREQUISITES

- It is mandatory to have done the following section of DEV-M0127-PRE-MicroEJ-Development-Tools-SDK6-NXP-i.MXRT1170:
    - Environment Setup
    - Debug the BSP C Code
- Optionally it would be useful to also read and follow this other section:
    - Runtime & Post Mortem Debugging Tools

DEVELOPMENT TOOLS OVERVIEW

MICROEJ

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				✕

Identify & Debug Performance Bottlenecks

Study done on a UI Application

IDENTIFYING & DEBUGGING BOTTLENECKS

Tools:

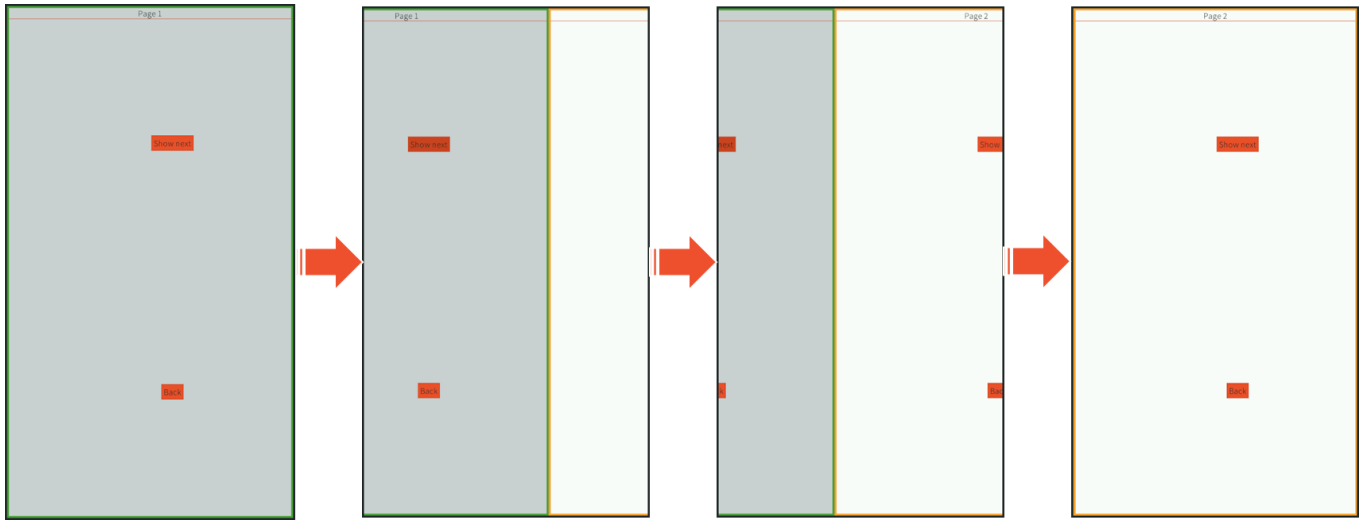
- Flush Visualizer
- Refresh Strategy Highlighting
- SystemView
- MicroUI Event Buffer Dump

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

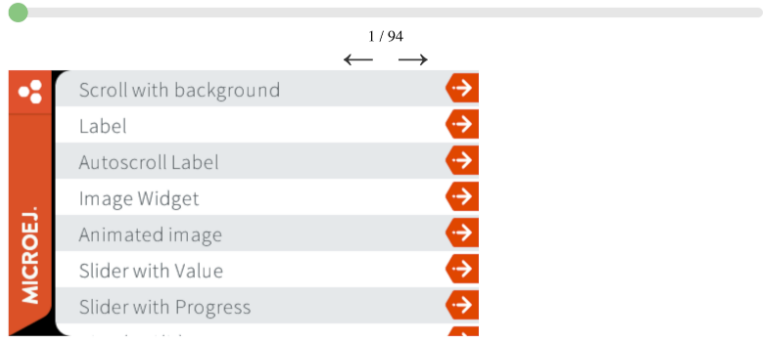
Building smooth and visually appealing UI applications requires a keen focus on performance. To achieve efficient UI rendering, minimizing unnecessary work that consumes valuable CPU time is essential.

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):

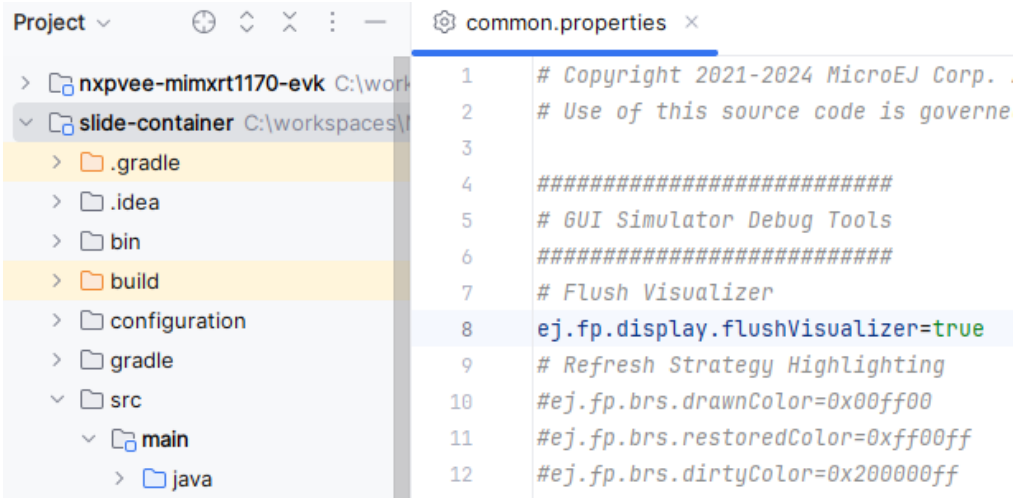
- VEE Port using UI Pack 14.0.0 or later.
- Frontpanel using the Display widget module version 4.+.

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.**



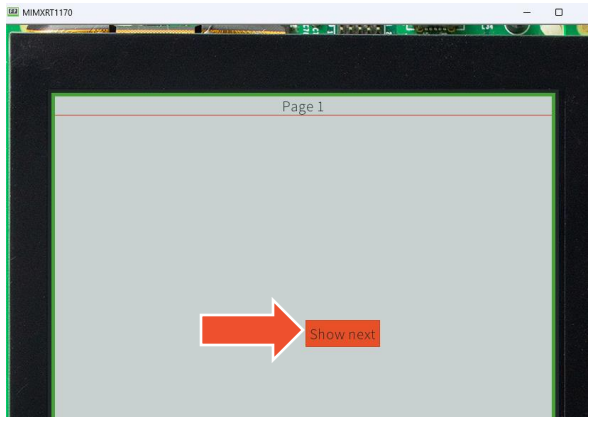
Application options of the slide-container sample

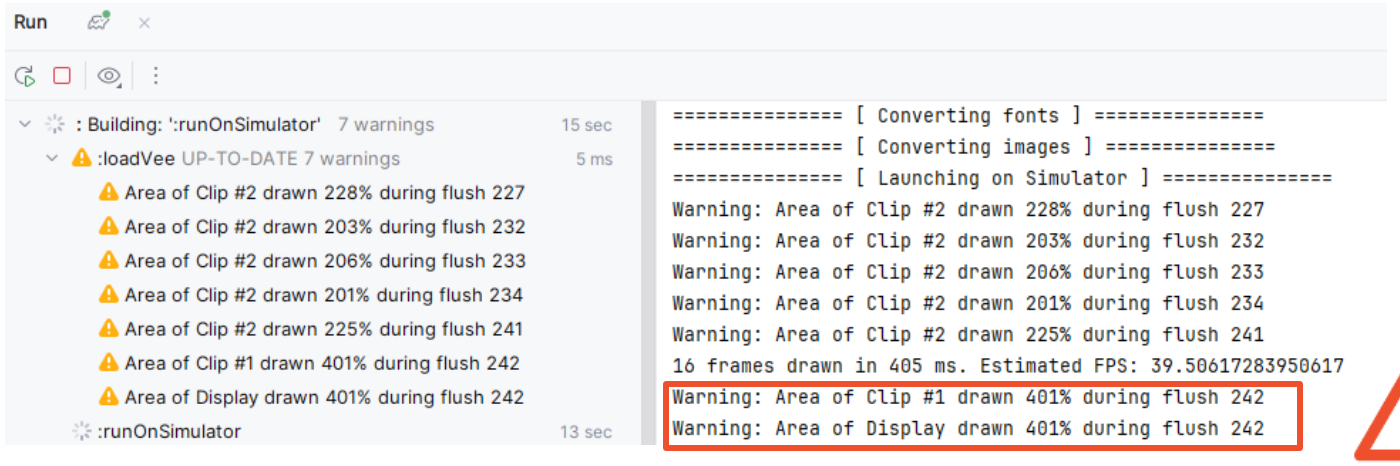
⚠ Starting with UI packs 14.4.1, properties to enabled the Flush Visualizer changed:

- **Set the property core.trace.enabled to true to enable the Flush Visualizer.**
- **Set the property core.trace.autostart to true to start the recording on startup.**

REPRODUCE THE ISSUE

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





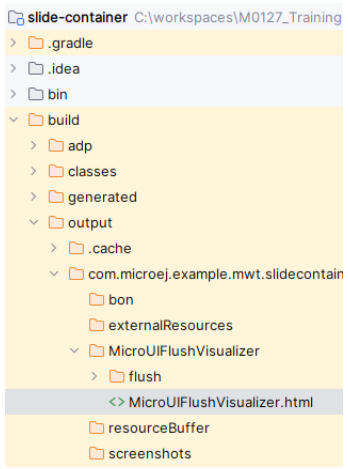
A value of 100% indicates that the area drawn is equivalent to the surface of the region.
A value of 200% indicates that the area drawn is equivalent to twice the surface of the region.
A perfect application has 100% of its root region drawn but its very unlikely for an application that draws anything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

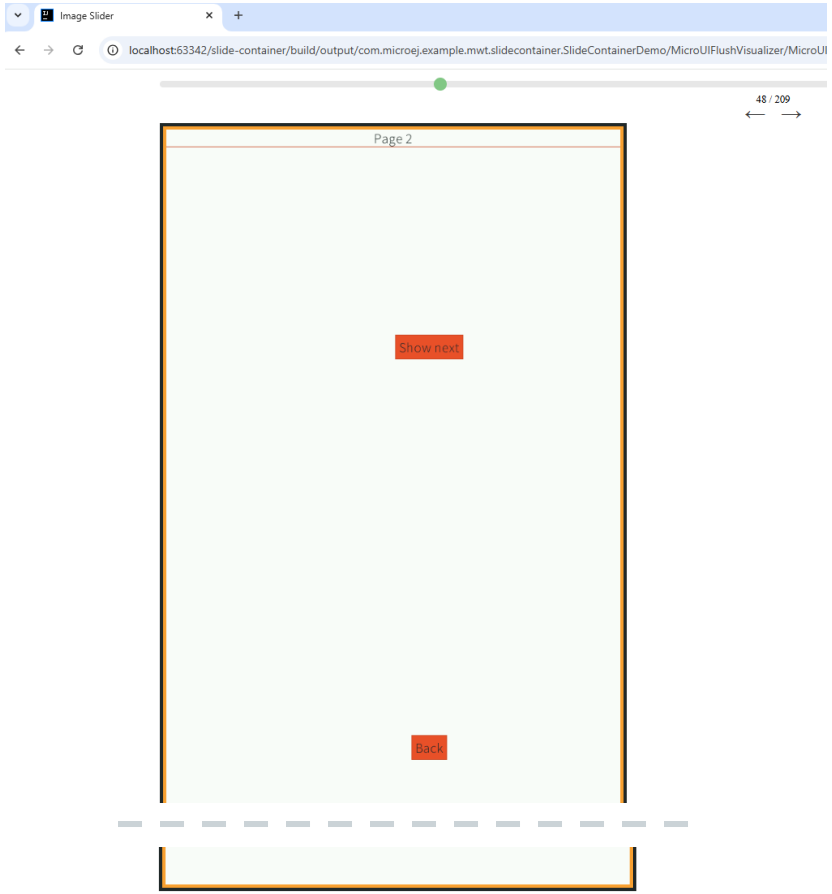
However, if the total area drawn is **bigger than 200%,** that means that the total **surface of the region was drawn more than twice**. Probably meaning that a **lot of drawings are done above others**.

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

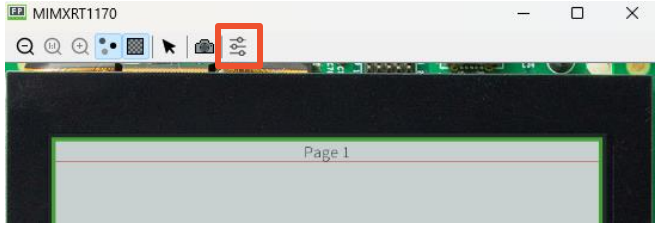
FLUSH ANALYZER REPORT

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





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



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):

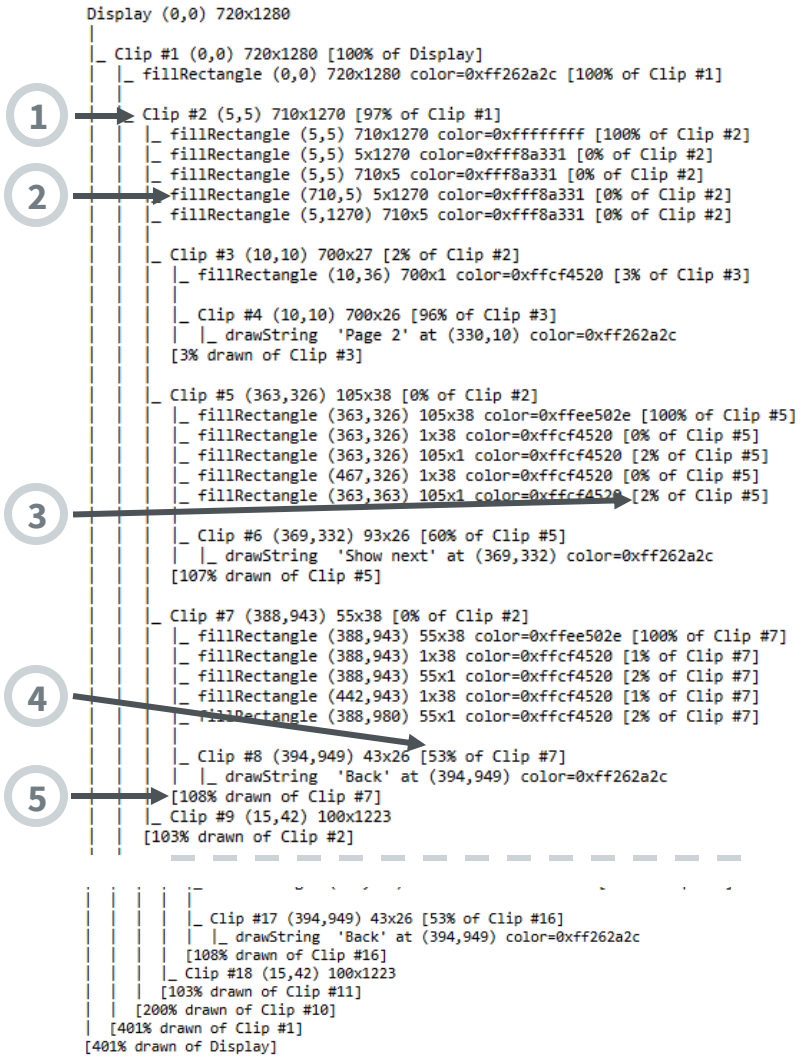
```
Warning: Area of Clip #1 drawn 401% during flush 242
Warning: Area of Display drawn 401% during flush 242
```

The report is available at the bottom of the page.

You should observe the report displayed beside.

The following information are provided:

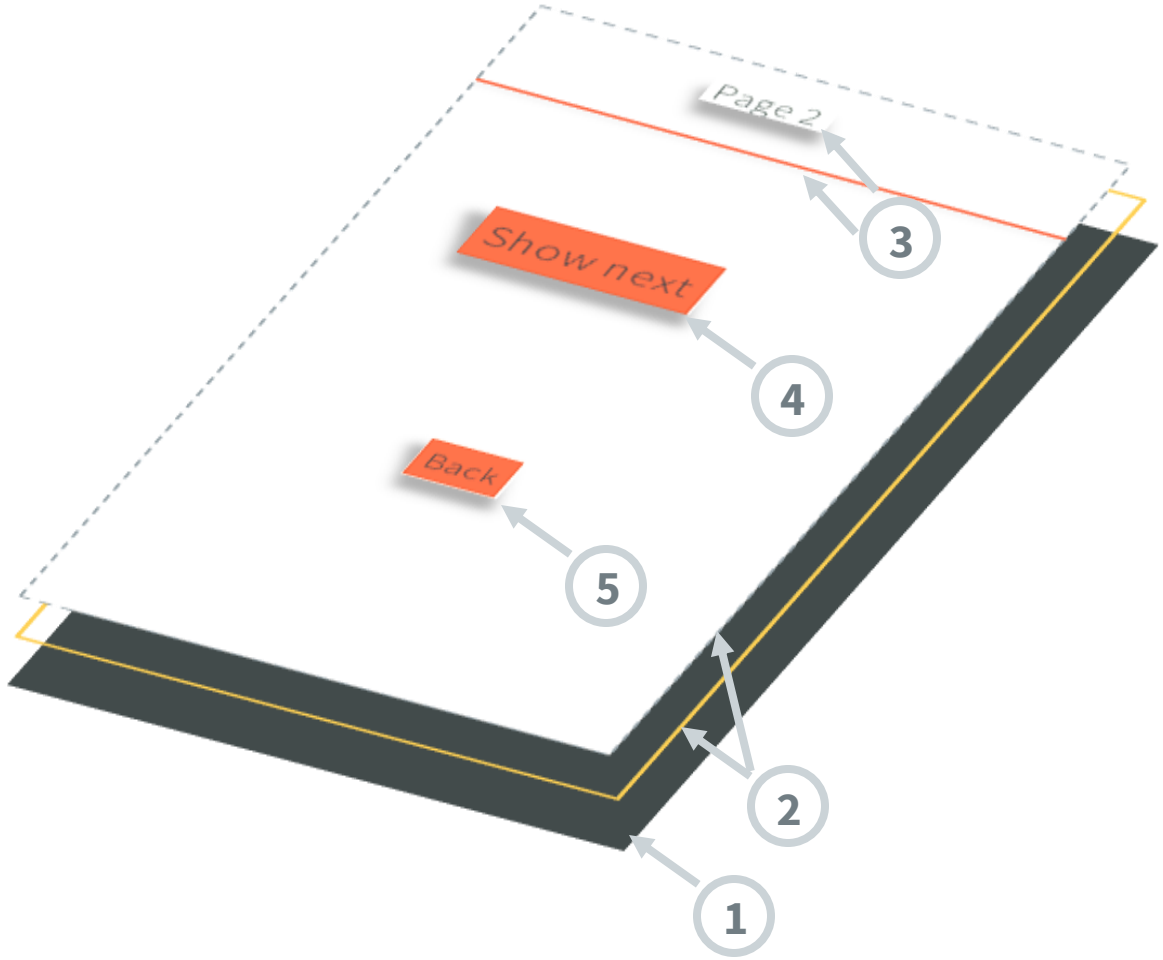
- The operations before a flush are structured as a tree, where nodes represent either:
  1. a region (display or clip)
  2. a drawing operation.
3. Each region has defined bounds, can contain other nodes, and displays the percentage of its parent region it covers.
4. Some drawings compute their coverage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



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:

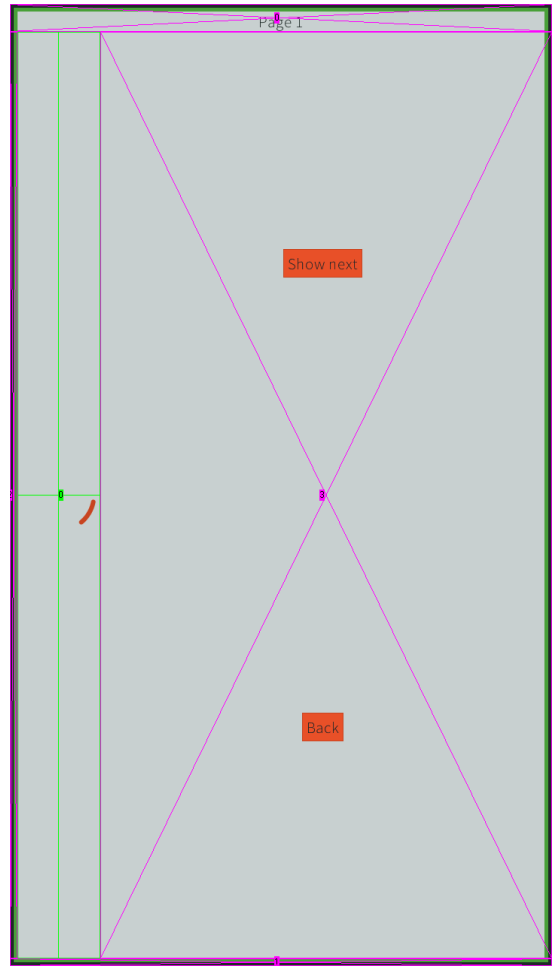
- Drawn Region(s)
- Restored Region(s)
- Dirty Region(s)

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:

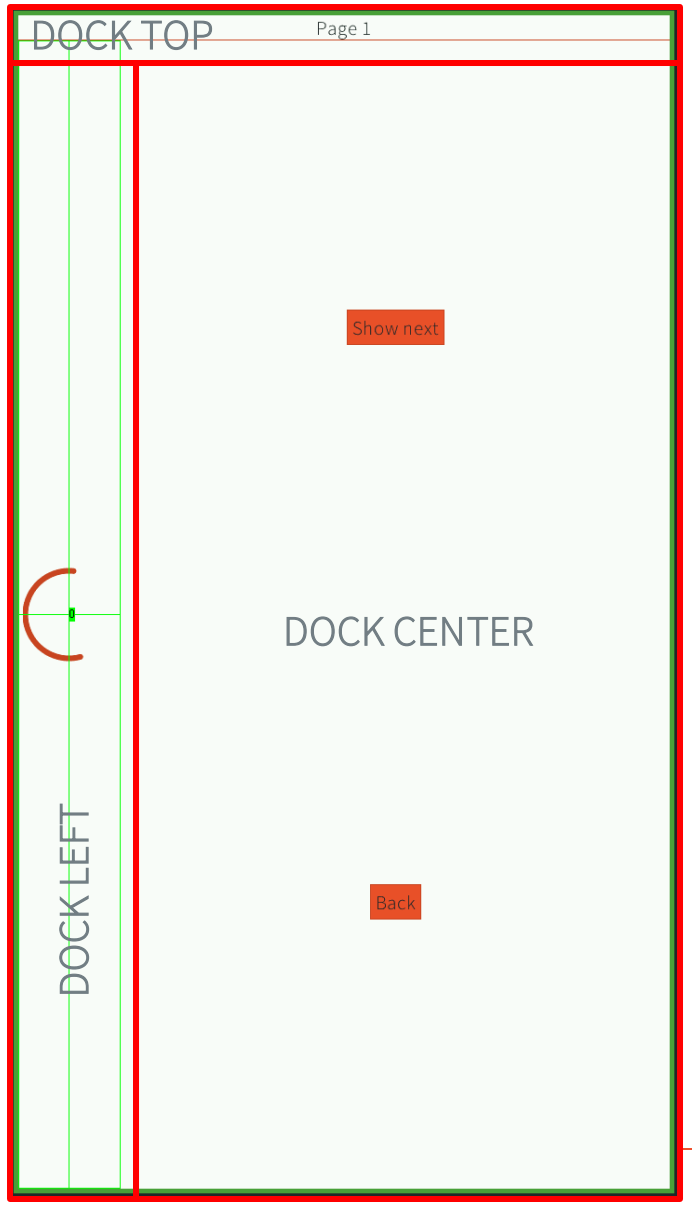
- The **CircularIndeterminateProgress** widget is included in a Dock Container (see **SlideContainerDemo** class):

```
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

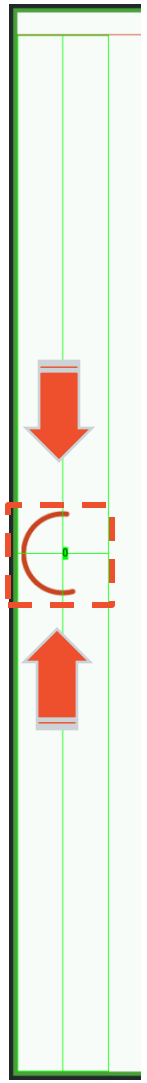
The refresh area could be reduced to fit the size of the **CircularIndeterminateProgress** widget.

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

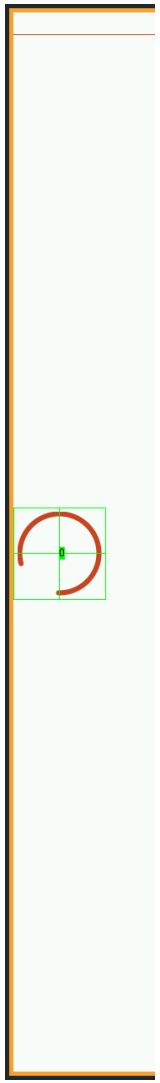
Update the style configuration of the example (*createStylesheet* method in the **SlideContainerDemo** class) to set an optimal dimension to the **CircularIndeterminateProgress** widget:

```
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



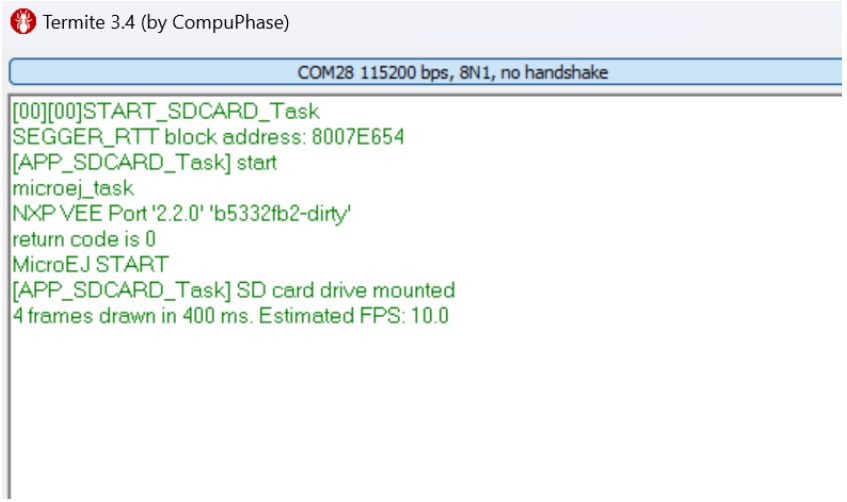
After widget style update

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:



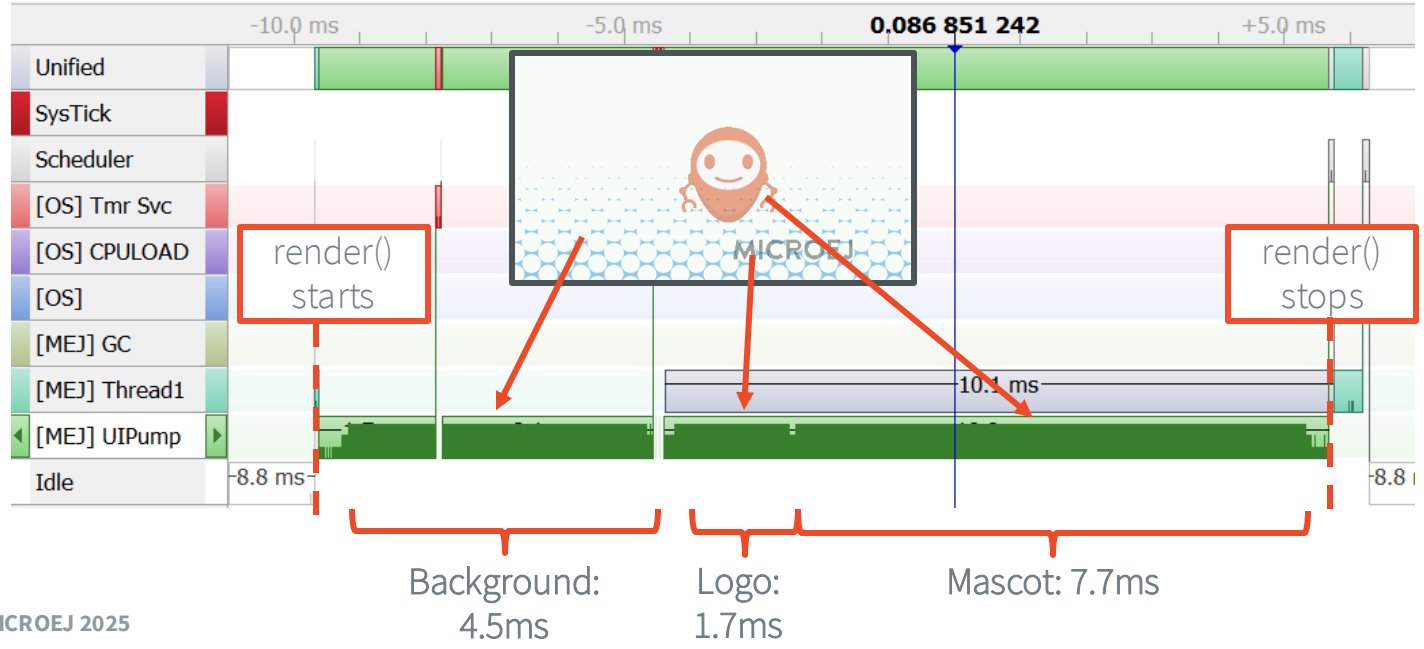
→ This board is theoretically able to run GUI applications near 60FPS. The next steps are about investigating the bottlenecks in the application and in the VEE Port that prevent having a smooth animation.

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:



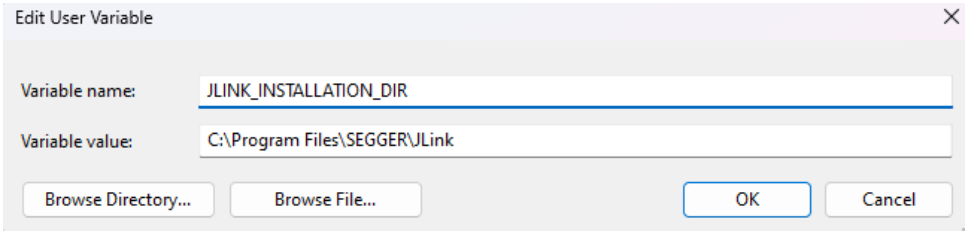

```
public void render(GraphicsContext gc) {
    drawTracer.recordEvent(MY_EVT_ID);
    int length = images.size();
    for(int i = -1; ++i<length;) {
        images.get(i).paint(gc);
    }
    drawTracer.recordEventEnd(MY_EVT_ID);
}
```

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

SETUP THE ENVIRONMENT FOR SYSTEMVIEW (1/2)

The following software are required:

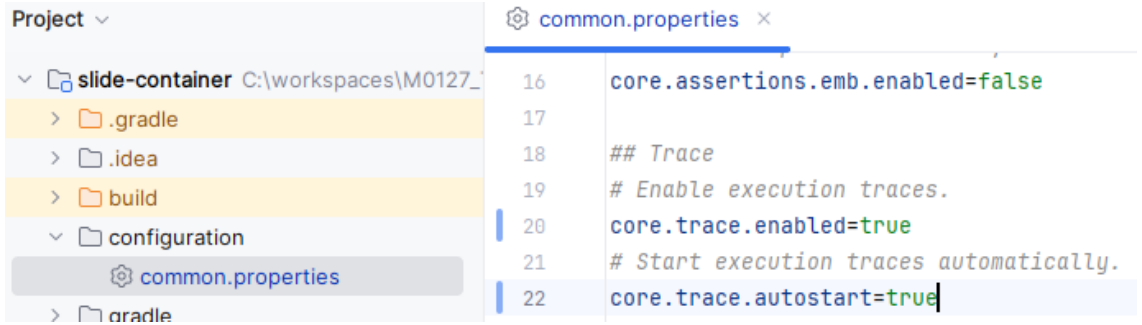
- Install SEGGER J-Link:
  - Create the **JLINK_INSTALLATION_DIR** environment variable that points to the SEGGER J-Link installation directory (e.g. C:\Program Files\SEGGER\Jlink):



- 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**



SETUP THE ENVIRONMENT FOR SYSTEMVIEW (2/2)

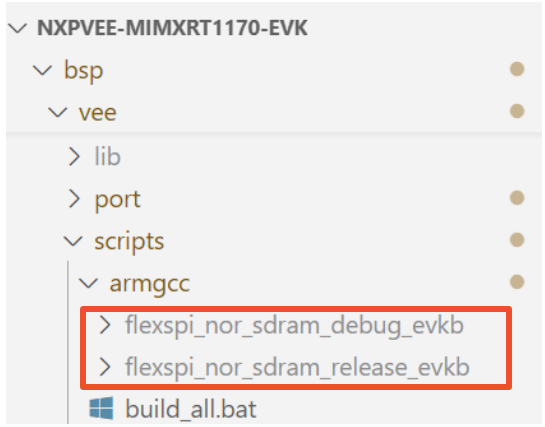
Enable SystemView in the VEE Port:

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

CMakePresets.json

```
19        },
20        "generator": "Ninja",
21        "binaryDir": "${sourceDir}/${presetName}",
22        "cacheVariables": {
23          "DEBUG_CONSOLE": "UART",
24          "LANGUAGE": "C",
25          "LIBRARY_TYPE": "NOLIB",
26          "ENABLE_AI": "0",
27          "ENABLE_SYSTEMVIEW": "1"
28        }
```

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/):



USE A J-LINK PROBE

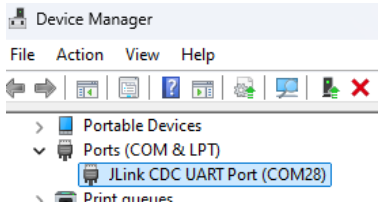
SystemView is a SEGGER tool, consequently it requires a J-Link probe to be used.

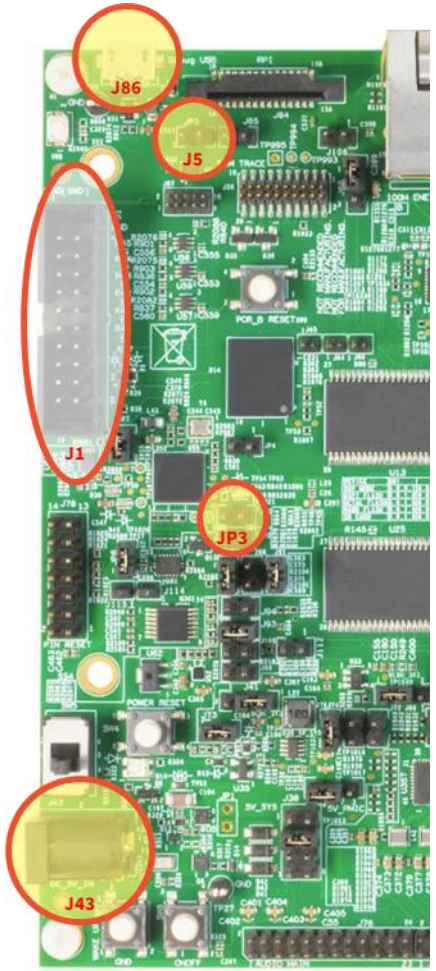
There are 2 ways to use a J-Link probe with the NXP i.MX RT1170 EVK:

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:





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

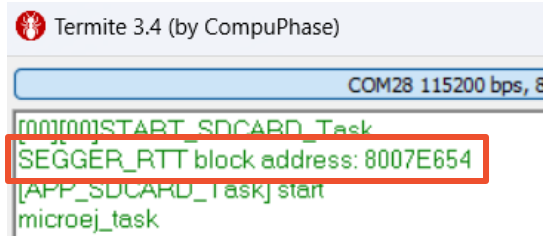
```
16
17  REM Set 1 for RELEASE mode and 0 for DEBUG mode
18  IF "%RELEASE%" == "" (
19      SET RELEASE=0
20  )
```

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

set_project_env.bat

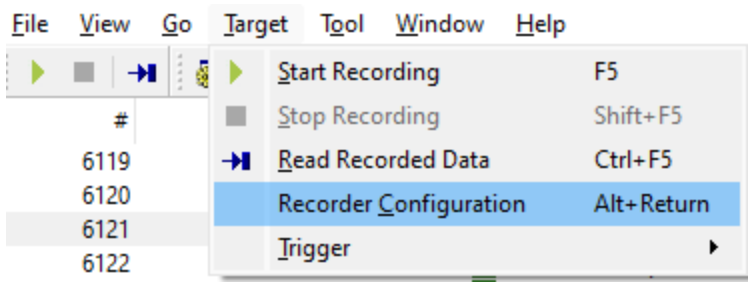
```
31  REM Set "flash_cmsisdap" for Linkserver probe or "flash" for J-Link probe
32  IF "%FLASH_CMD%" == "" (
33      SET FLASH_CMD=flash
34  )
```

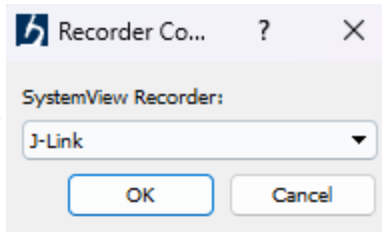
Once done, build and flash the application on the device using the **runOnDevice** task.
Once done, the following trace should appear in the console:

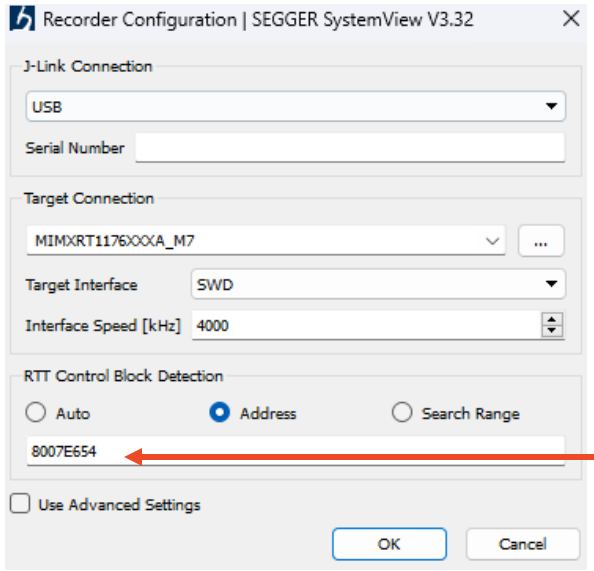


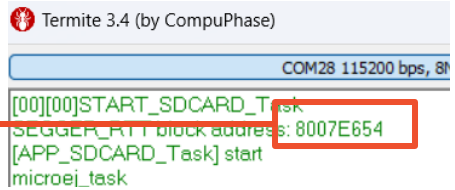
START THE ACQUISITION ON SYSTEMVIEW

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

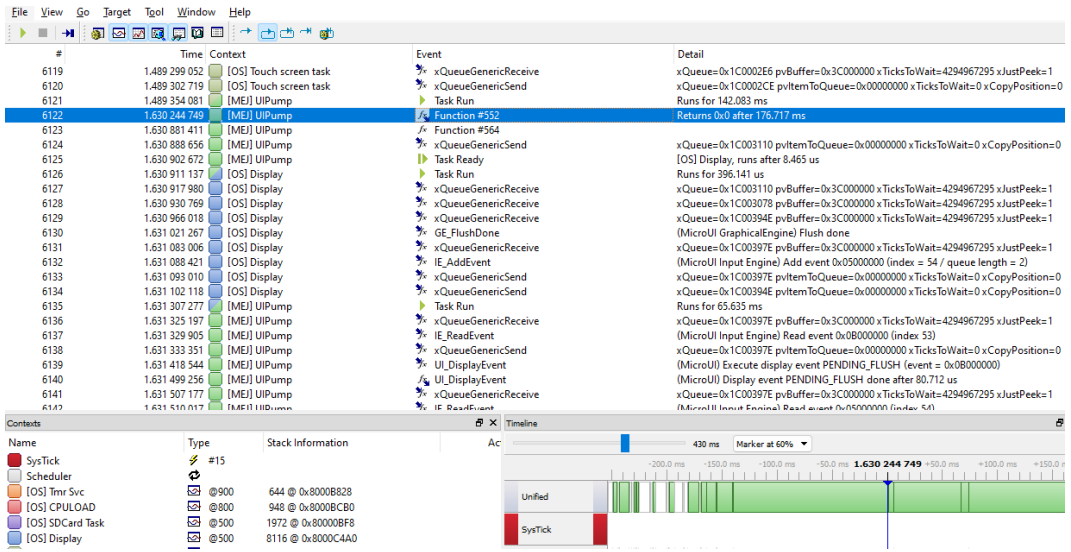








- Once done, click on the **Play** button.
  The acquisition starts, events can be see in SystemView:

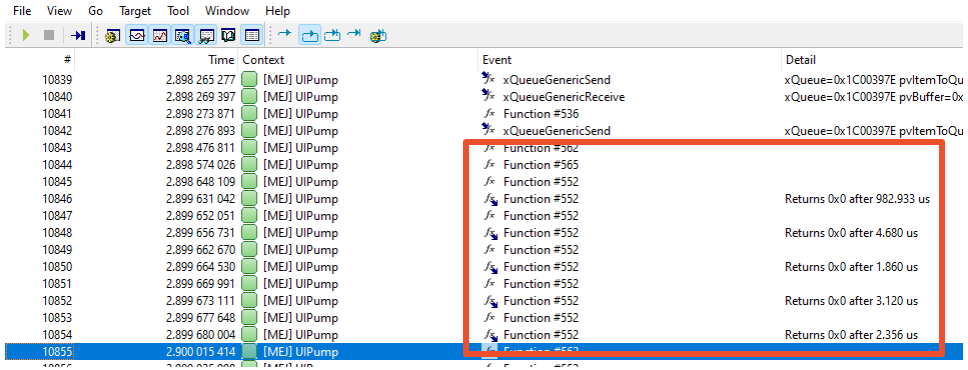


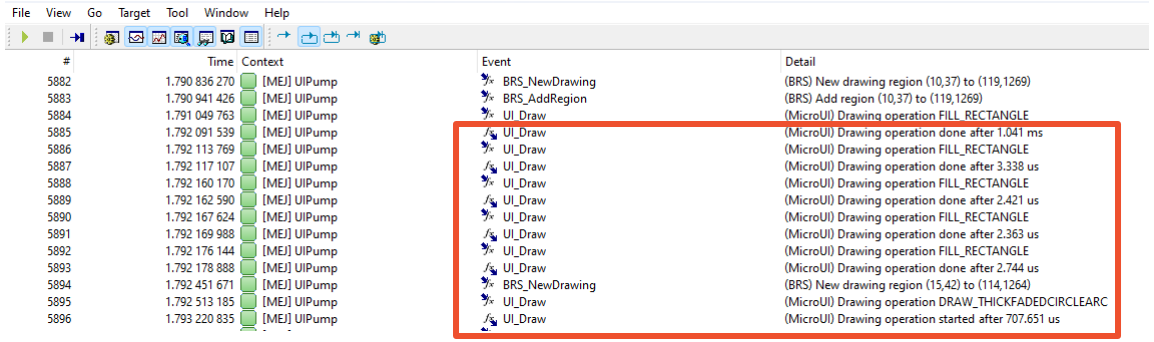
GET MICROUI DEBUG TRACES

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

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

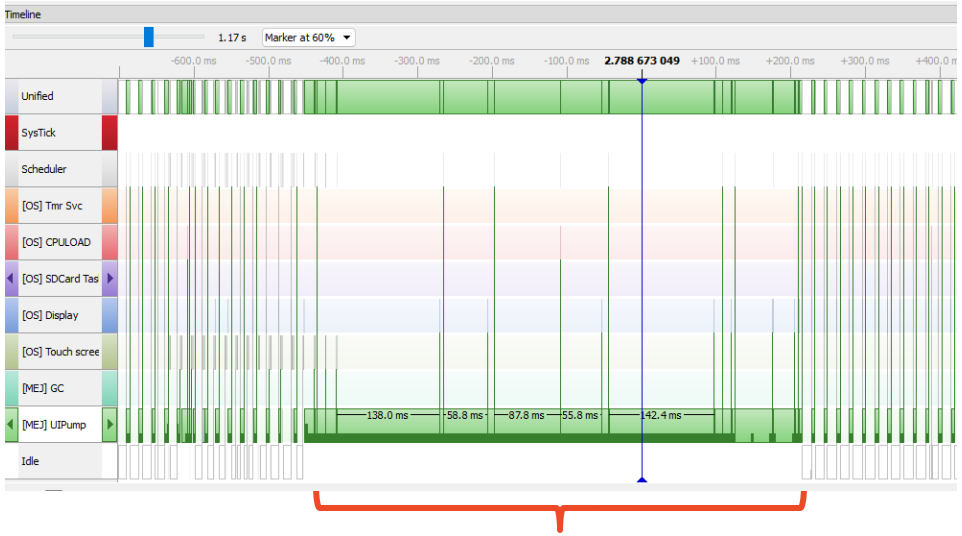
- Copy the traces provided in the following section: https://docs.microej.com/en/latest/ApplicationDeveloperGuide/UI/MicroUI/traces.html#systemview-integration
    - **Warning**: jump a line at the end of the line for it to be taken into account
- Save them in a **SYSVIEW_MicroUI.txt** file (the syntax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

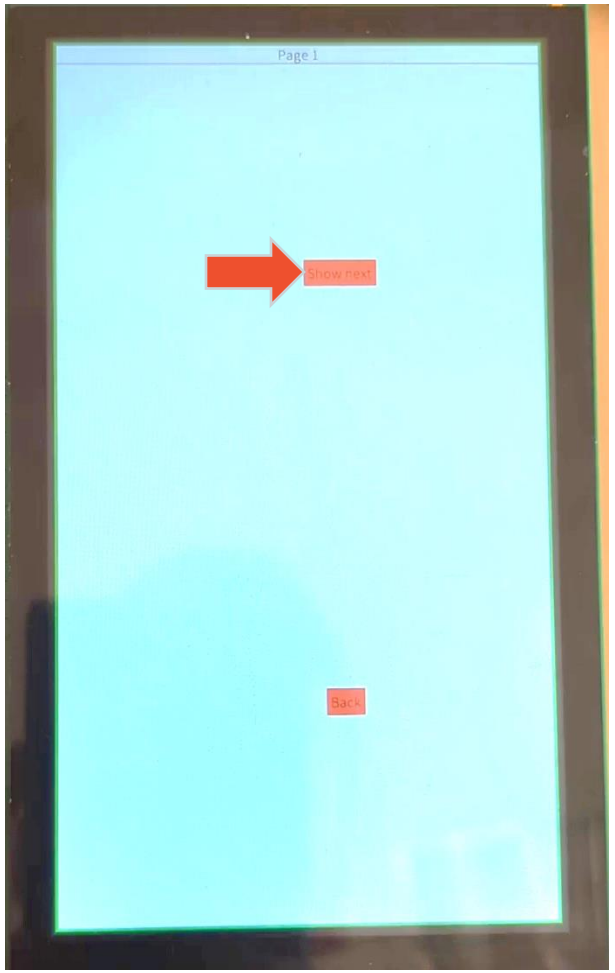




TRACE ANALYSIS

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





- 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:

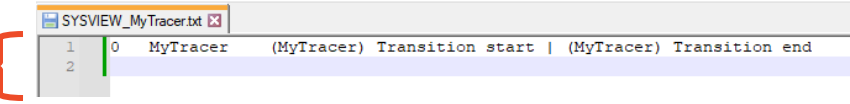
```
private void doAnimation(final Widget leftChild, final Widget rightChild, int startX, final int endX) {
  . . .
  this.releasedAnimation = new MotionAnimation(getAnimator(), motion, new MotionAnimationListener() {
    @Override
    public void tick(int value, boolean finished) {
      . . .
        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)

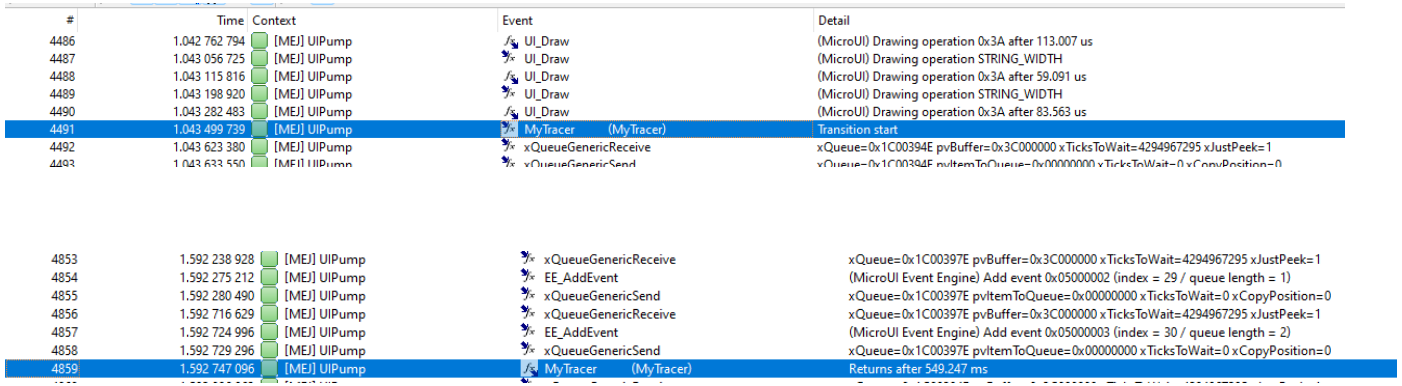
- The custom **MyTracer** event needs to be added to SystemView description files in order to be taken into account during the analysis.
- Create a **SYSVIEW_MyTracer.txt** file in **C:\Program Files\SEGGER\SystemView\Description**
- Add the following content in it:

```
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:



- 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:

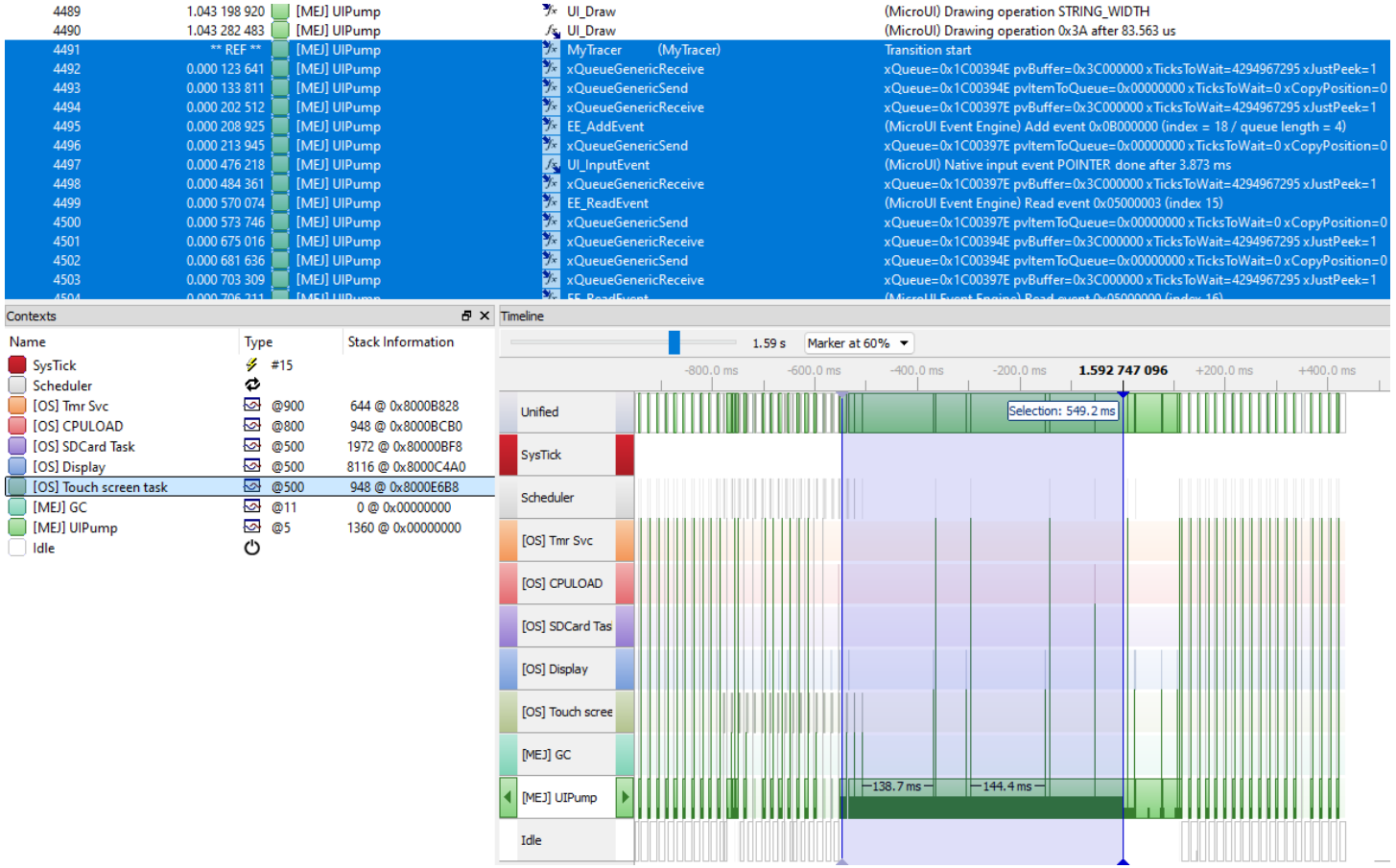


#	Time	Context	Event	Detail
4486	1.042 762 794	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation 0x3A after 113.007 us
4487	1.043 056 725	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation STRING_WIDTH
4488	1.043 115 816	[MEJ] UIPump	UI_Draw	(MicroUI) Drawing operation 0x3A after 59.091 us
4489	1.043 198 920	[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
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:



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

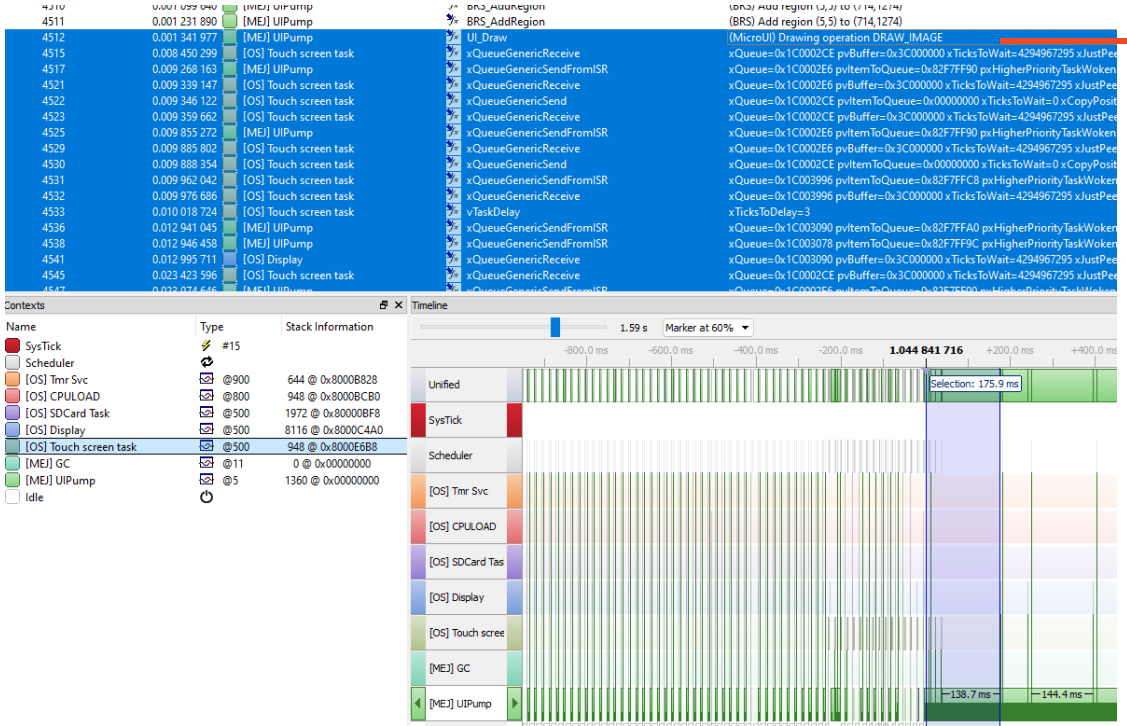
ANALYSIS (2/6)

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:



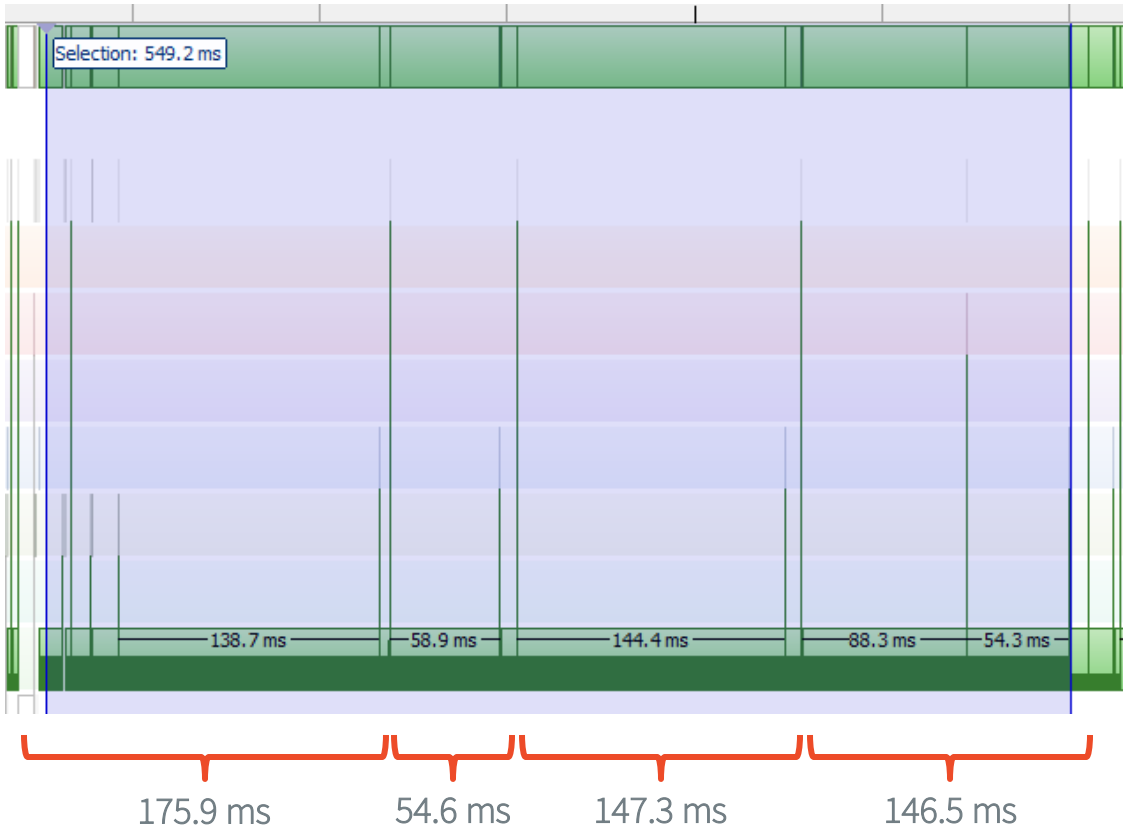
This is a **drawImage** operation

→ 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:

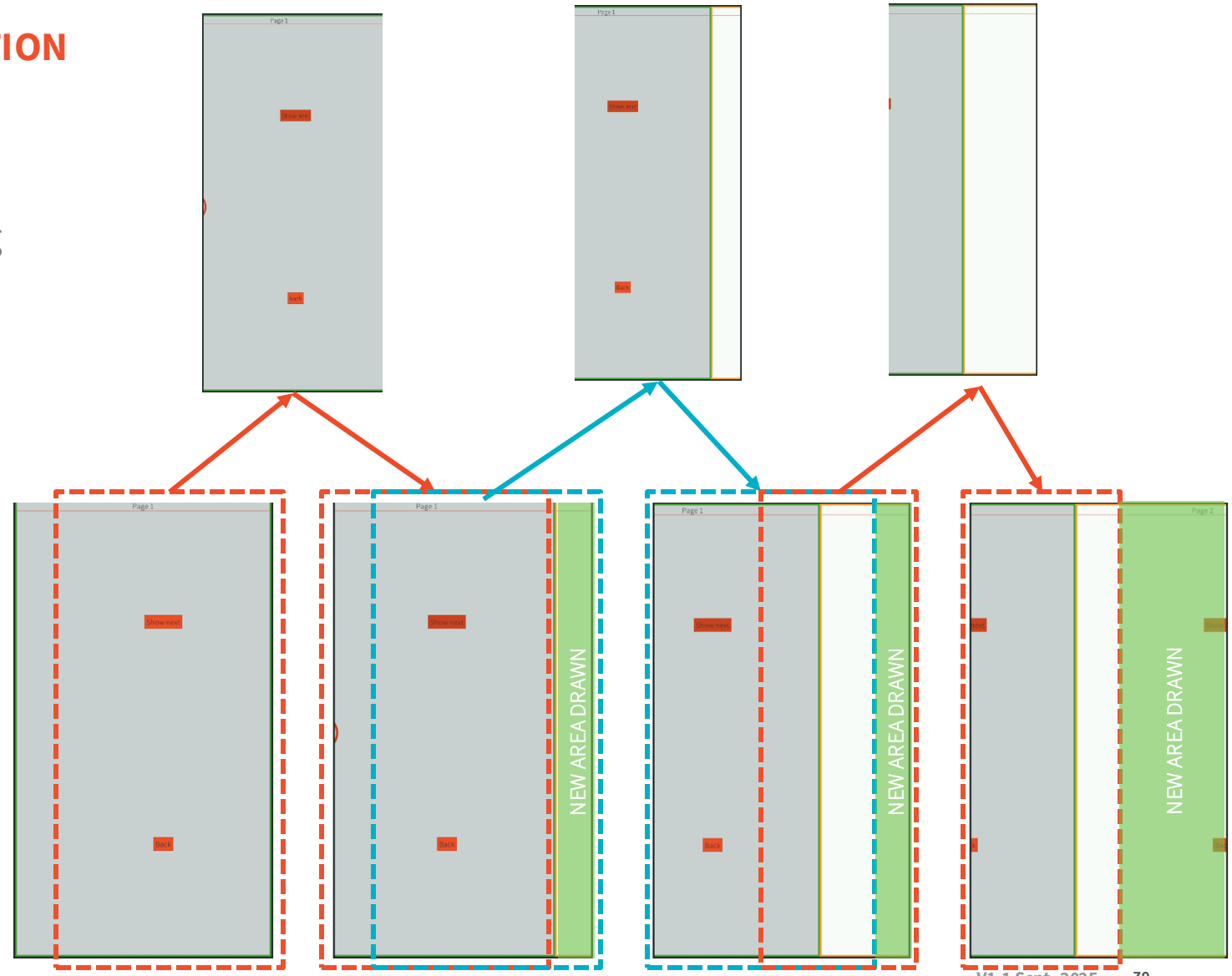
1. Check the application implementation to understand why / how **drawImage** operations are done.
2. Check that the **drawImage** LLAPI is properly implemented in the BSP (use hardware accelerator, front buffer located in a high speed memory, memory cache enabled, …)

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)

```
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);
}
```

BSP (C code) (MicroUI C Module)

LLUI_PAINTER_impl.c

```
419  }
420
421  // See the header file for the function documentation
422  void LLUI_PAINTER_IMPL_drawImage(MICROUI_GraphicsContext *gc, MICROUI_Image *img, jint regionX, jint regionY,
423                                   jint width, jint height, jint x, jint y, jint alpha) {
424      if (LLUI_DISPLAY_requestDrawing(gc, (SNI_callback) & LLUI_PAINTER_IMPL_drawImage)) {
425          DRAWING_Status status = DRAWING_DONE;
426          LOG_DRAW_START(drawImage);
445                  if (gc->image.format == img->format) {
446                      // source & destination have got the same pixels memory representation
447
448                      MICROUI_Image *image = LLUI_DISPLAY_getSourceImage(img);
449
450                      if ((0xff /* fully opaque */ == l_alpha) && !LLUI_DISPLAY_isTransparent(img)) {
451                          // copy source on destination without applying an opacity (beware about the overlapping)
452                          status = UI_DRAWING_copyImage(gc, image, regionX, regionY, width, height, x, y);
453                      } else if (LLUI_DISPLAY_getBufferAddress(img) == LLUI_DISPLAY_getBufferAddress(&gc->image)) {
454                          // blend source on itself applying an opacity (beware about the overlapping)
455                          status = UI_DRAWING_drawRegion(gc, regionX, regionY, width, height, x, y, l_alpha);
```

ui_drawing.c

```
1056
1057  // See the header file for the function documentation
1058  BSP_DECLARE_WEAK_FCNT DRAWING_Status UI_DRAWING_DEFAULT_copyImage(MICROUI_GraphicsContext *gc, MICROUI_Image *img,
1059                                                                    jint regionX, jint regionY, jint width, jint height,
1060                                                                    jint x, jint y) {
1061  #if !defined(LLUI_IMAGE_CUSTOM_FORMATS)
1062      return UI_DRAWING_SOFT_copyImage(gc, img, regionX, regionY, width, height, x, y);
1063  #else
1064      return UI_IMAGE_DRAWING_copy(gc, img, regionX, regionY, width, height, x, y);
1065  #endif
1066  }
```

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

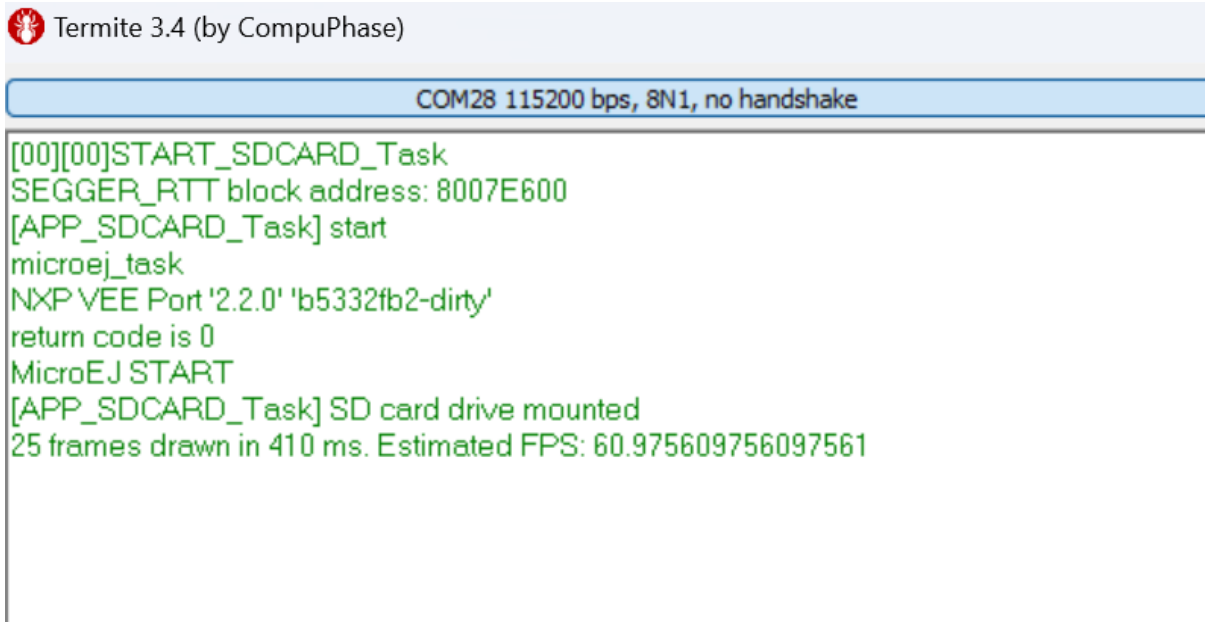
- Open the **SlideContainer** class of the **slide-container** example.
- Comment the **Render implementation 1** (render() and renderContent() methods).
- Look for the Render implementation 2, uncomment the render() content method:

```
294     /*
295      * Render implementation 2: fully render the 2 pages at each frame during the slide transition.
296      * This implementation is more CPU consuming but is not relying on the drawDisplayRegion API that performs a
297      * RAM to RAM copy.
298      *
299      * Warning: this implementation is drawing all the children hierarchy, even if they are not visible.
300      *
301      * Next step: override the renderContent method to only draw the 2 last children (the visible ones).
302      */
303     @Override
304     public void render(GraphicsContext g) {
305         if (this.moving) {
306             this.previousPosition = this.position;
307         }
308         super.render(g);
309     }
```

- Run the application on the device.

RUN THE APPLICATION ON THE DEVICE

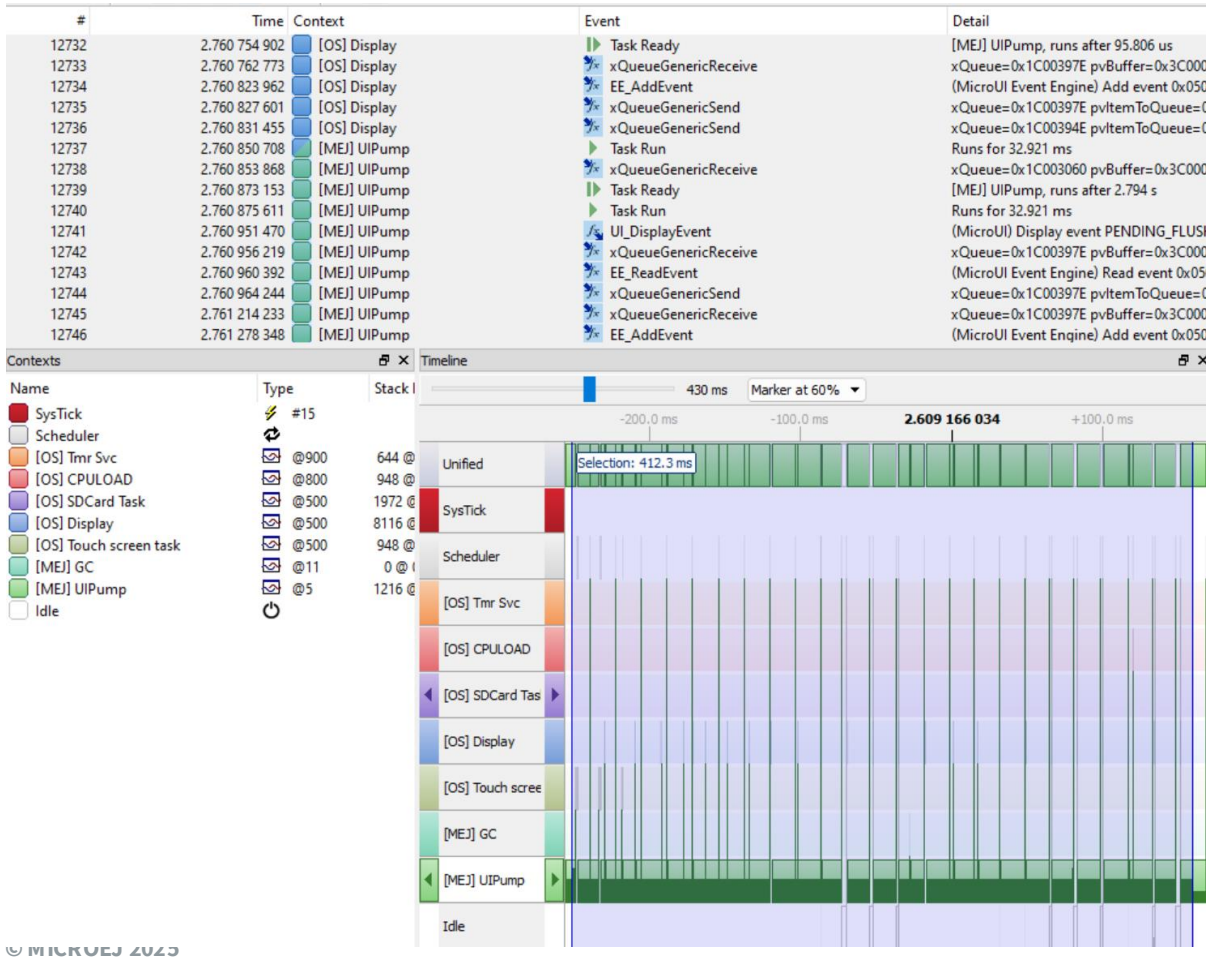
- Click on the **Show next** button on the screen.
- The implementation looks way smoother, see video in **slide-container/videos/slide_containrer_nxp_rt1170_optimized.m4v**
- The FPS have increased to 60 FPS:



- 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).



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

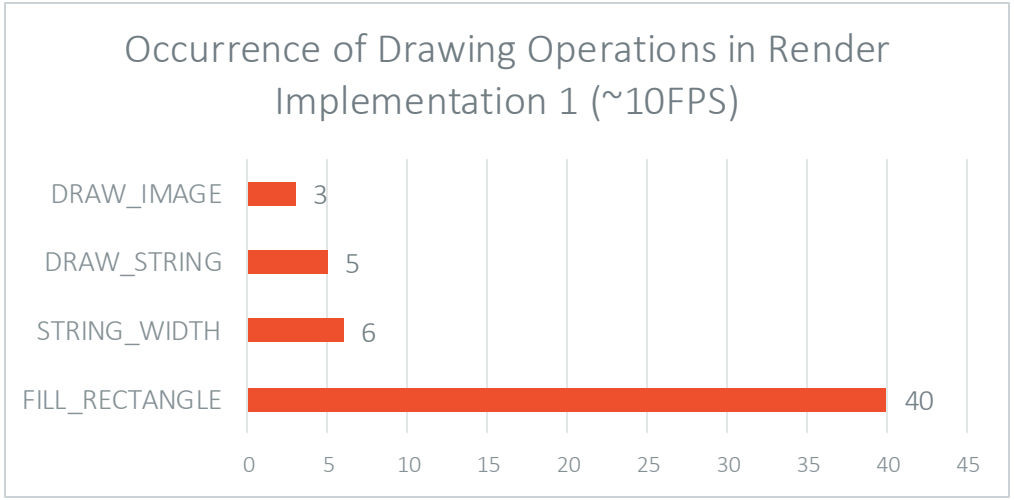
Note that the CPU load is still near 100% (almost no idle time in the timeline)

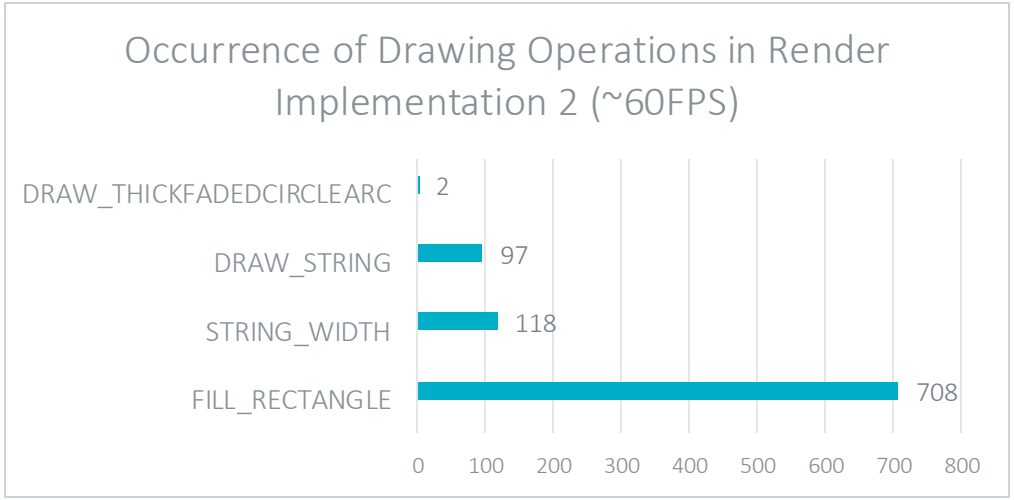
- This is due to the implementation of the slide animation. An Animator is used, it executes animations as fast as possible to get the best framerate.
- Check the Animations implementation documentation to learn more about the various implementations available.

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.

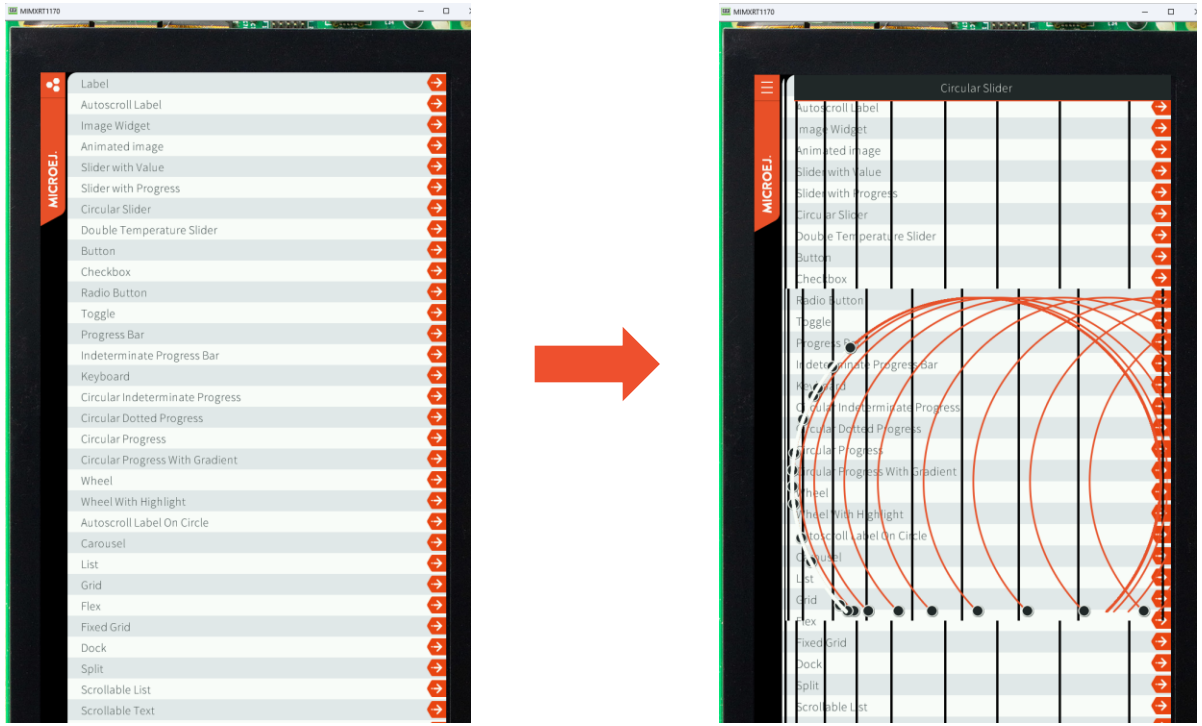
```
============================== MicroUI FIFO Dump ==============================
--------------------------------- Old Events ---------------------------------
[27: 0x00000000] garbage
[28: 0x00000000] garbage
[...]
[99: 0x00000000] garbage
[00: 0x08000000] Display SHOW Displayable (Displayable index = 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:

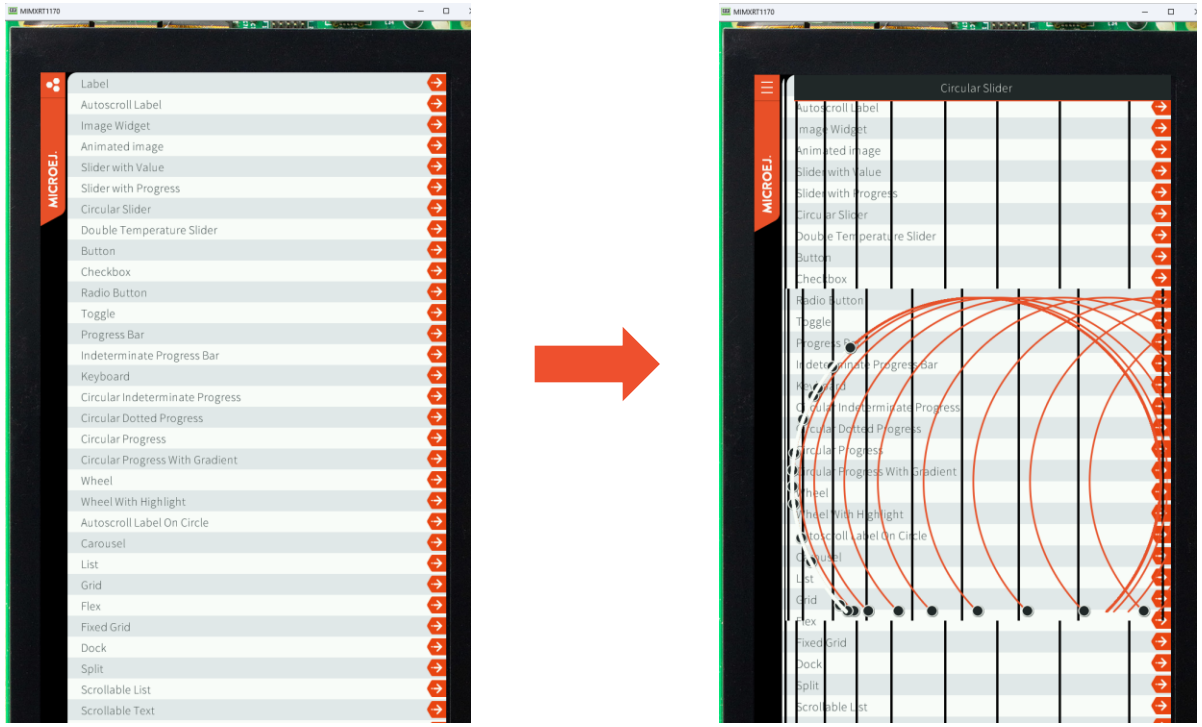
- Print the hierarchy of widgets and styles
- Print the path to a widget
- Count the number of widgets or containers
- Count the maximum depth of a hierarchy
- Print the bounds of a widget
- Print the bounds of all the widgets in a hierarchy

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, pa
|  |  +--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[SourceSansP
```

→ 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:

*CircularSliderPage.java

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

Fix proposals:

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

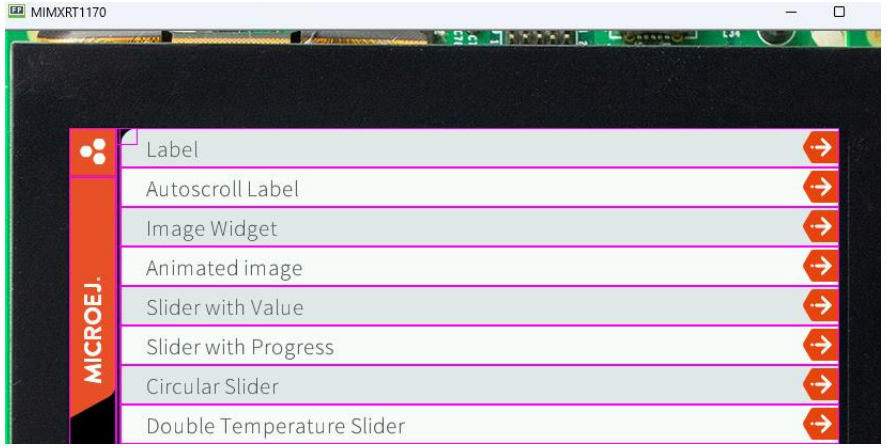
MWT Debug Utilities

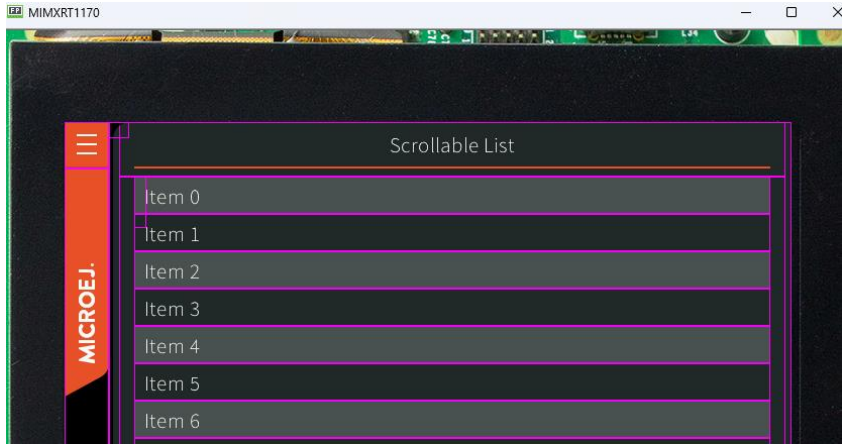
Debug tools provided in the MWT library

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}
```

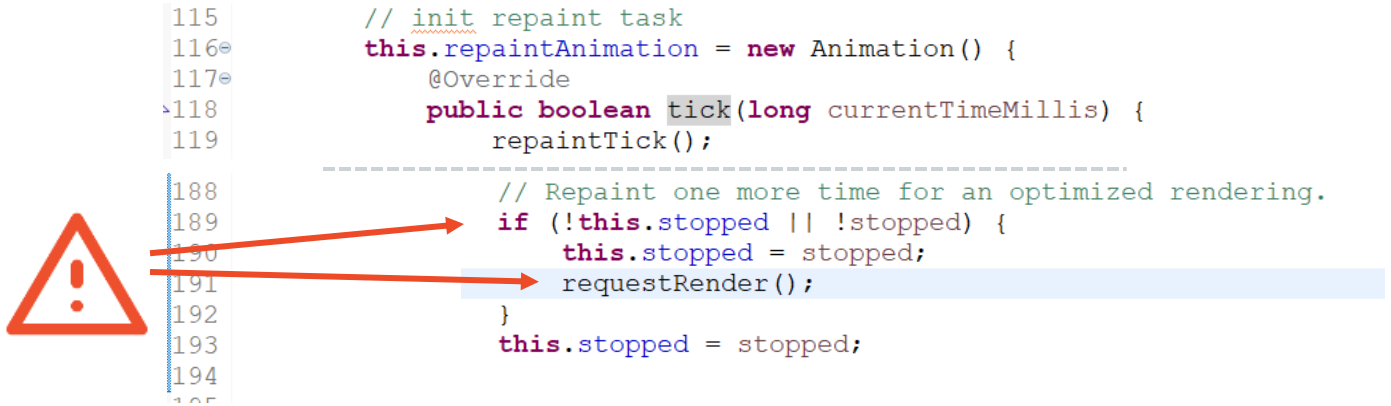
Check the Debug Utilities page for more information.

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]
```



- 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

GET IN TOUCH WITH US

MICROEJ USA
Boston

MICROEJ FRANCE
Nantes

MICROEJ GERMANY
Munich

MICROEJ ROMANIA
Sibiu

MICROEJ JAPAN
Tokyo

MICROEJ KOREA
Yongin-si

www.microej.com

developer.microej.com

MICROEJ®