



JavaOne™

ORACLE®

Be in control of your JavaFX mission

Oleg Mazurov, Marcus Hirt

Oracle, Inc.

October 2, 2014

Outline

- JavaFX threading architecture
- PulseLogger
- Using Java Mission Control/Java Flight Recorder with JavaFX
- Implementation

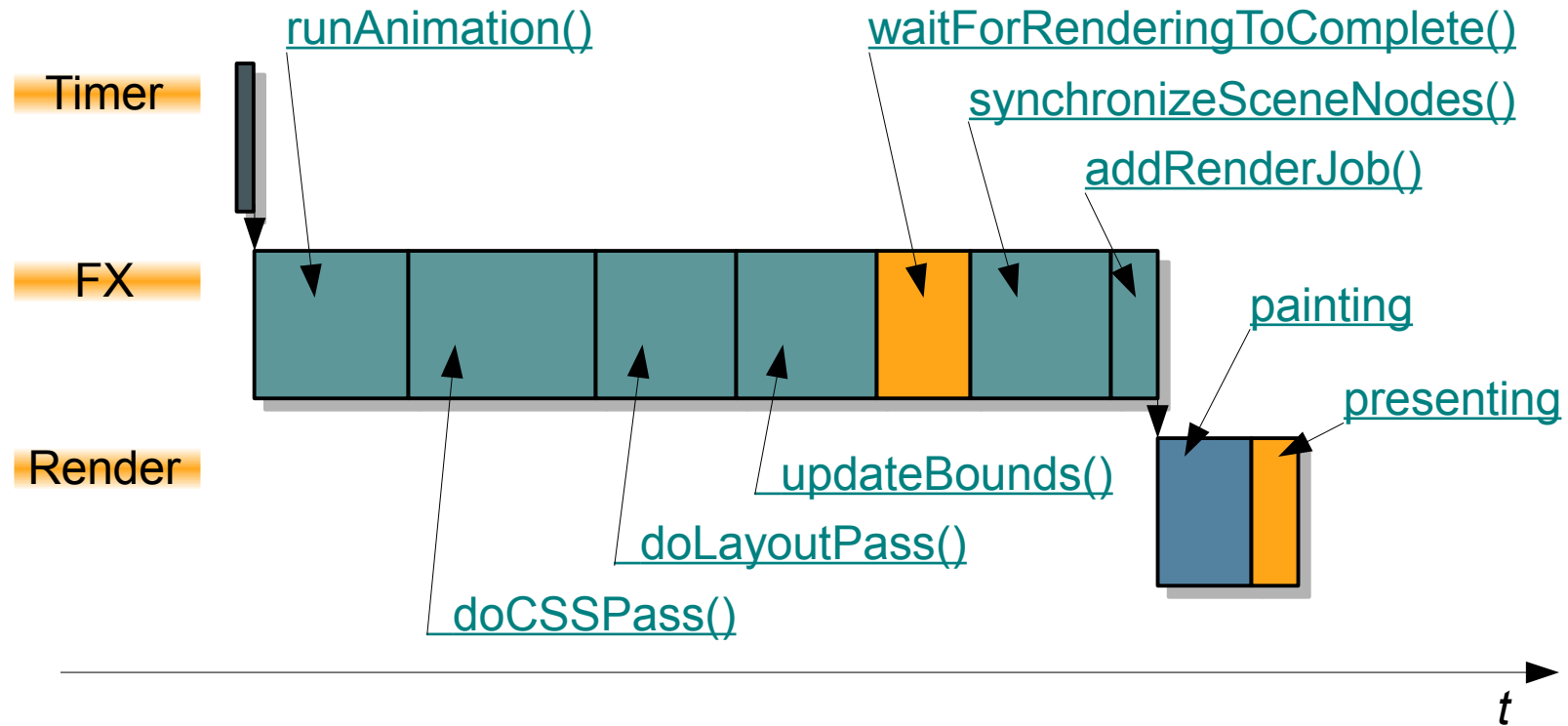
JavaFX threading architecture

- Scene graph updated in response to input events, animation, user tasks
- Updates are translated to render commands (SW or HW accelerated)
- JavaFX Application (FX) thread
 - input events
 - pulse
 - user tasks
- Render thread
 - render commands
 - presenting

JavaFX pulse

- 16ms timer
- New pulse is started if:
 - animation is running
 - OR there are dirty scenes
 - OR explicitly requested
 - AND NOT the previous pulse is still running

JavaFX pulse diagram



PulseLogger

- JVM system properties
 - -Djavafx.pulseLogger=true
 - -Djavafx.pulseLogger.threshold=17
- Output
 - Pulse based and inter-pulse data
 - Time intervals
 - Messages
 - Counters

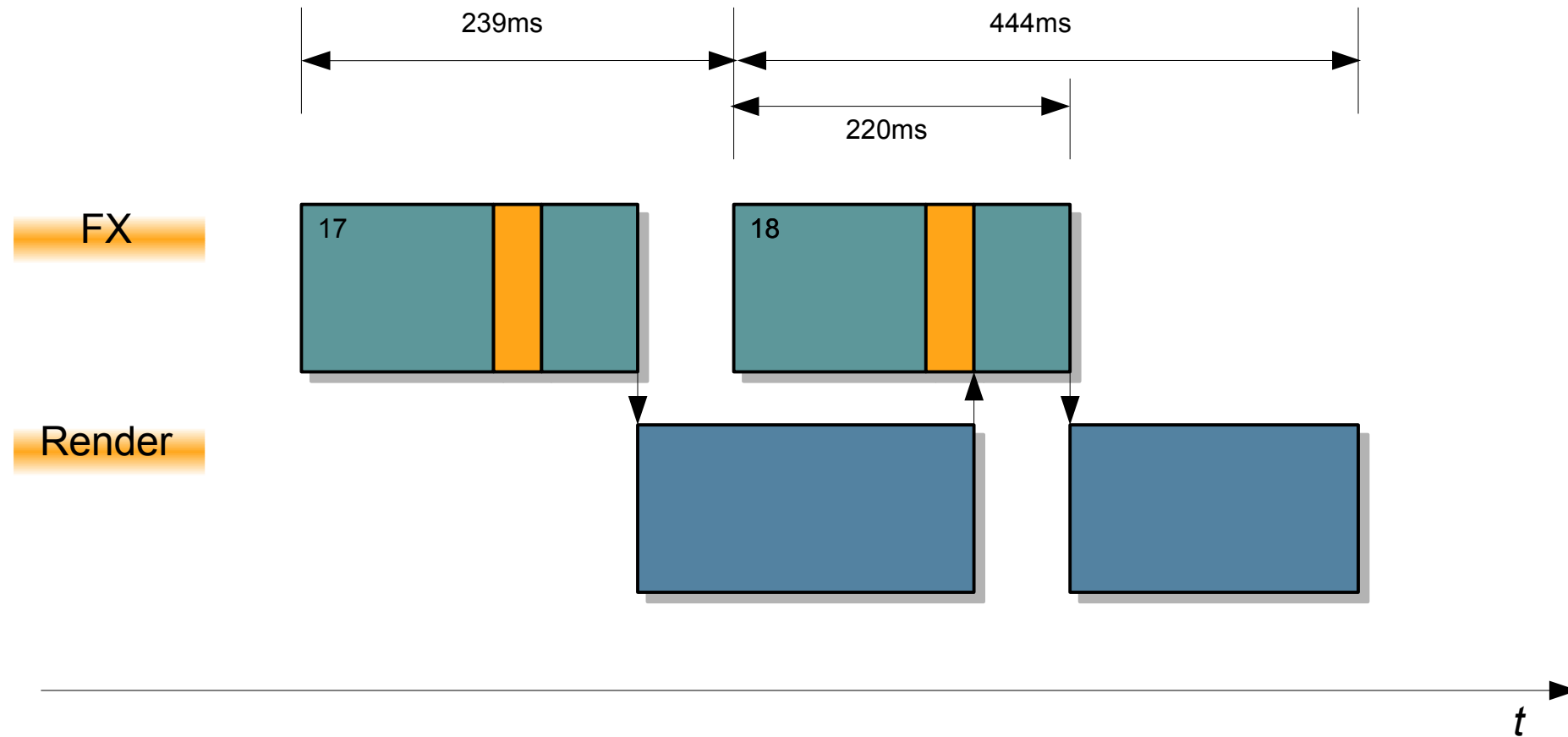
PulseLogger example

```
java -Djavafx.pulseLogger=true -jar GUIMark2.jar guimark2.BitmapTest
```

```
PULSE: 18 [239ms:444ms]
T11 (21 +0ms): CSS Pass
T11 (21 +3ms): Layout Pass
T11 (24 +20ms): Update bounds
T11 (44 +166ms): Waiting for previous rendering
T11 (211 +8ms): Copy state to render graph
T8 (220 +0ms): Dirty Opts Computed
T8 : 1 different dirty regions to render
T8 : Dirty Region 0: RectBounds { minX:0.0, minY:14.0, maxX:1200.0, maxY:614.0}
(w:1200.0, h:600.0)
T8 : Render Root Path 0: [com.sun.javafx.sg.prism.NGGroup@1fb7f2f]
T8 (220 +1ms): Render Roots Discovered
T8 (221 +212ms): Painting
T8 (433 +10ms): Presenting
Counters:
    Nodes rendered: 18087
    Nodes visited during render: 18094

PULSE: 19 [224ms:453ms]
...
```


PulseLogger intervals



Java Mission Control / Java Flight Recorder

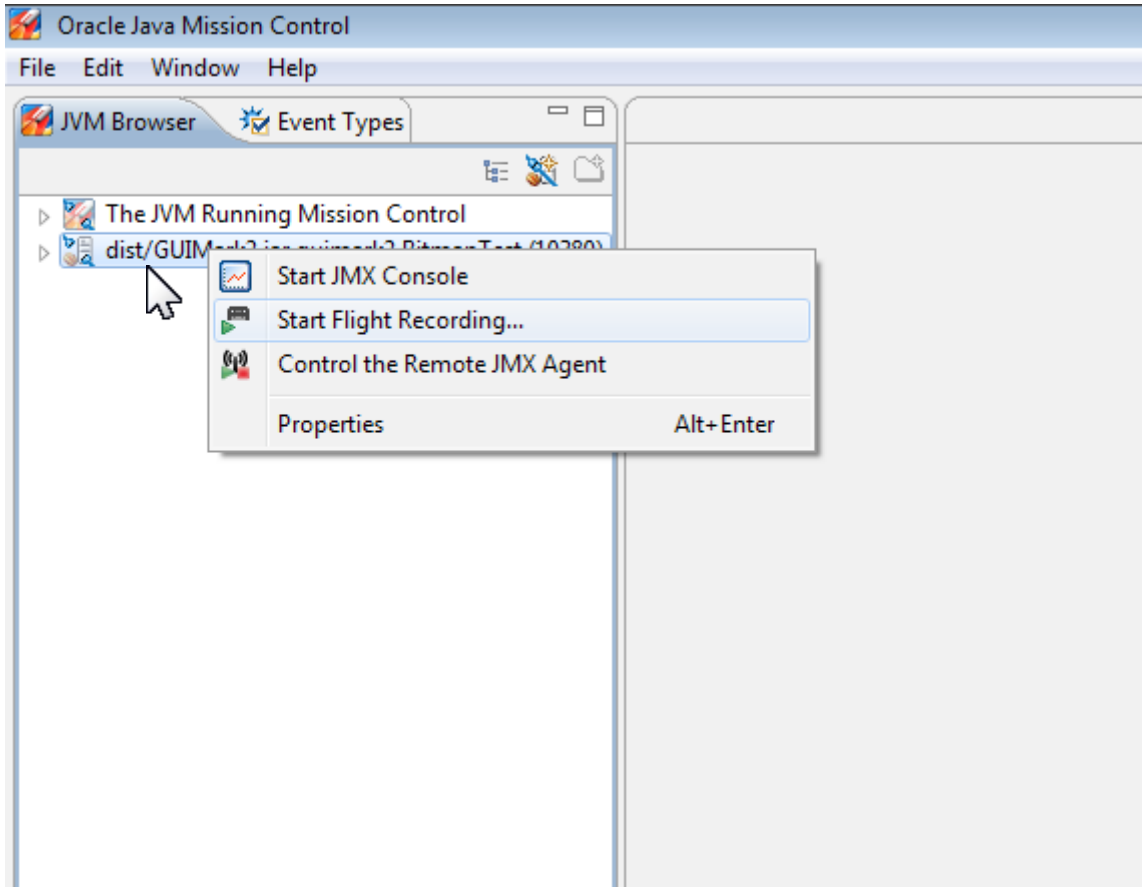
Java Mission Control / Java Flight Recorder

- JMC – a tools suite for monitoring, managing, diagnosing, and profiling
- JFR – profiling and event collection framework built into the Oracle JDK
- JavaFX support since Java SE 8u20
 - JavaFX events
 - JavaFX plugin

Starting JavaFX application

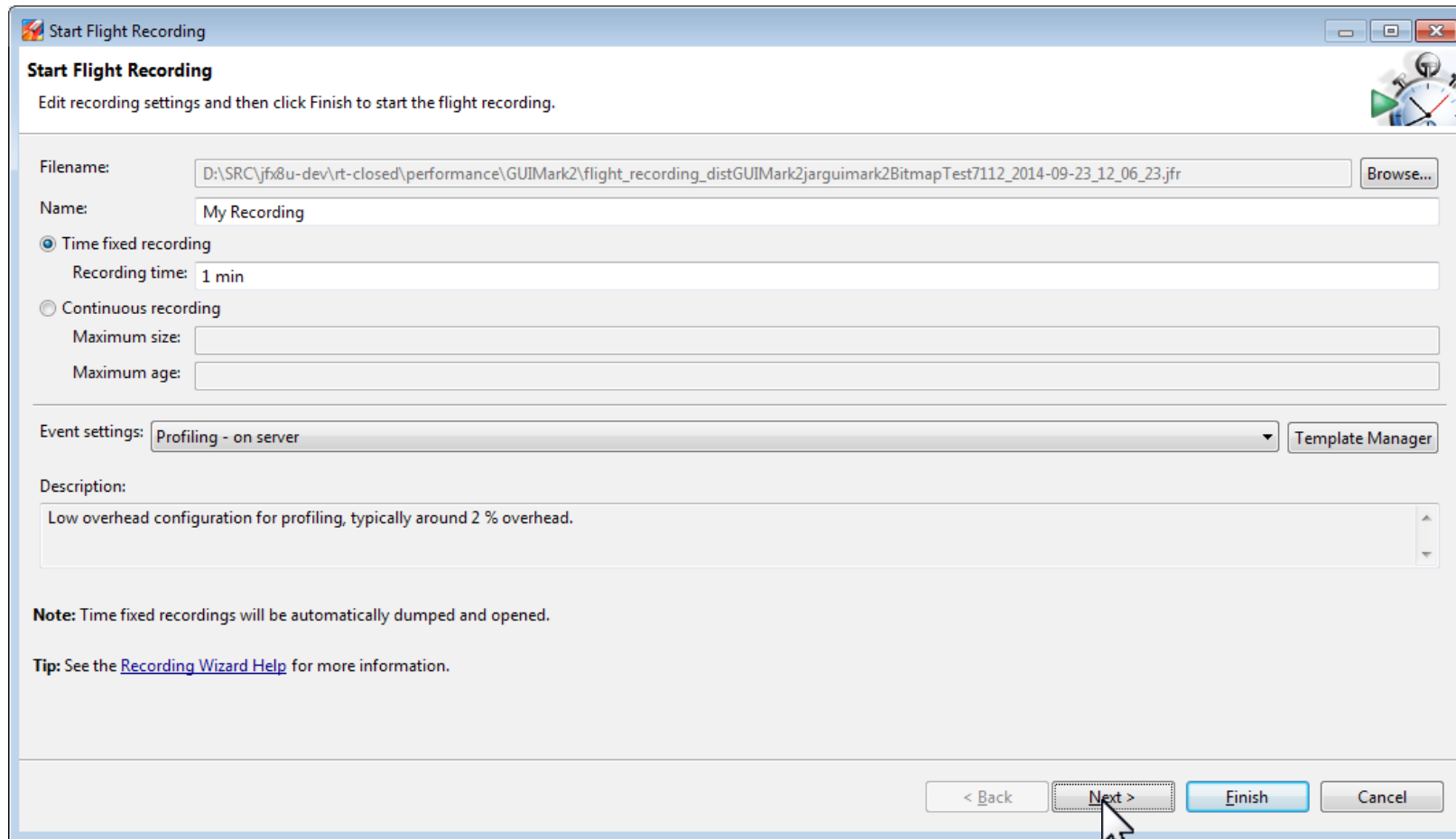
- Add to the JVM options:
 - XX:+UnlockCommercialFeatures -XX:+FlightRecorder**
- Either on the command line or
- Setting **_JAVA_OPTIONS** environment variable or
- Using deployment-specific mechanisms

Recording events with JMC



- Start `<jdk8u20>/bin/jmc`
- JVM Browser →
Running JVM instance →
Start Flight Recording...

Recording events with JMC (2)



Start Flight Recording

Edit recording settings and then click Finish to start the flight recording.

Filename:

Name:

Time fixed recording

Recording time:

Continuous recording

Maximum size:

Maximum age:

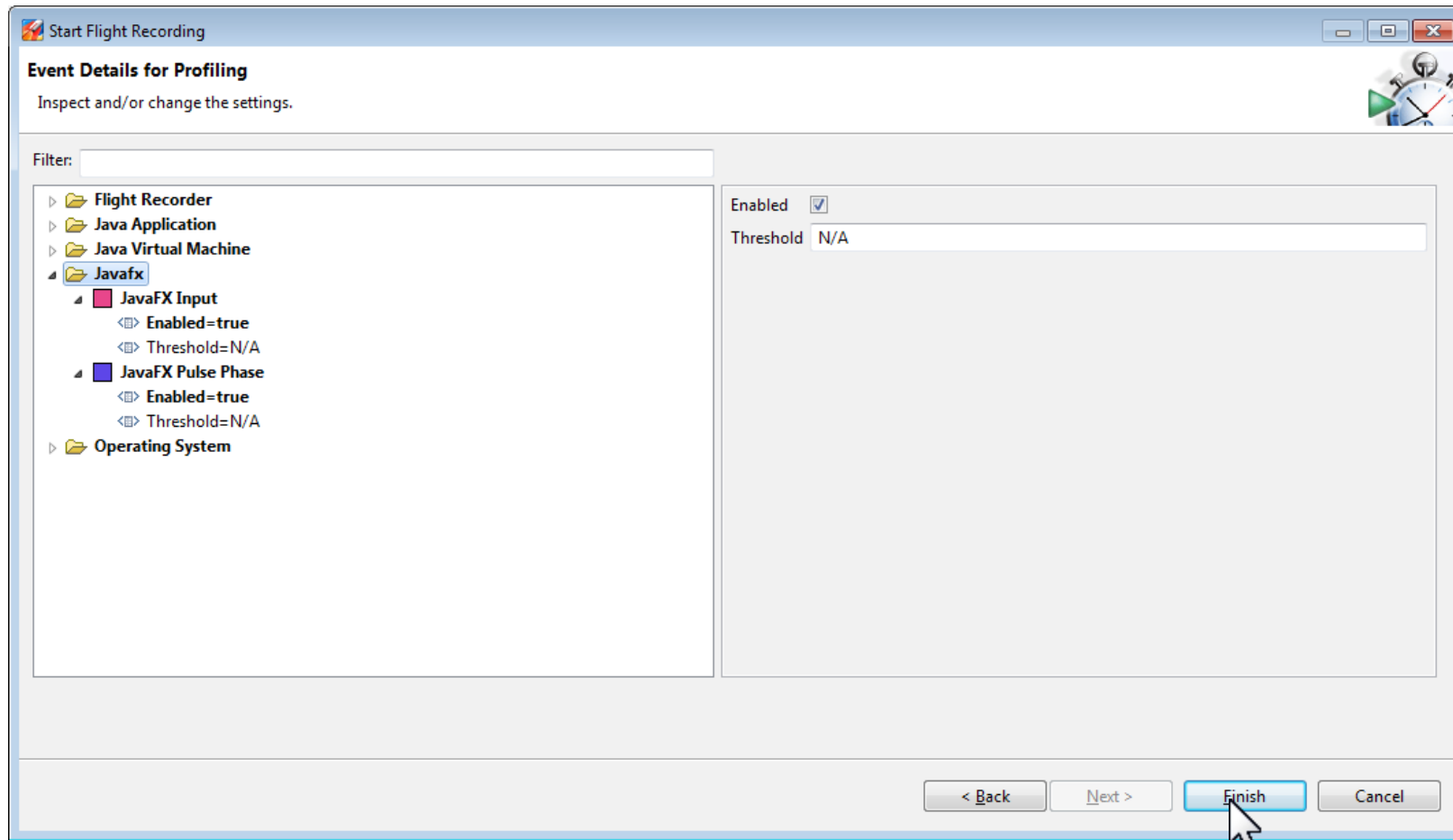
Event settings:

Description:

Note: Time fixed recordings will be automatically dumped and opened.

Tip: See the [Recording Wizard Help](#) for more information.

Recording events with JMC (3)



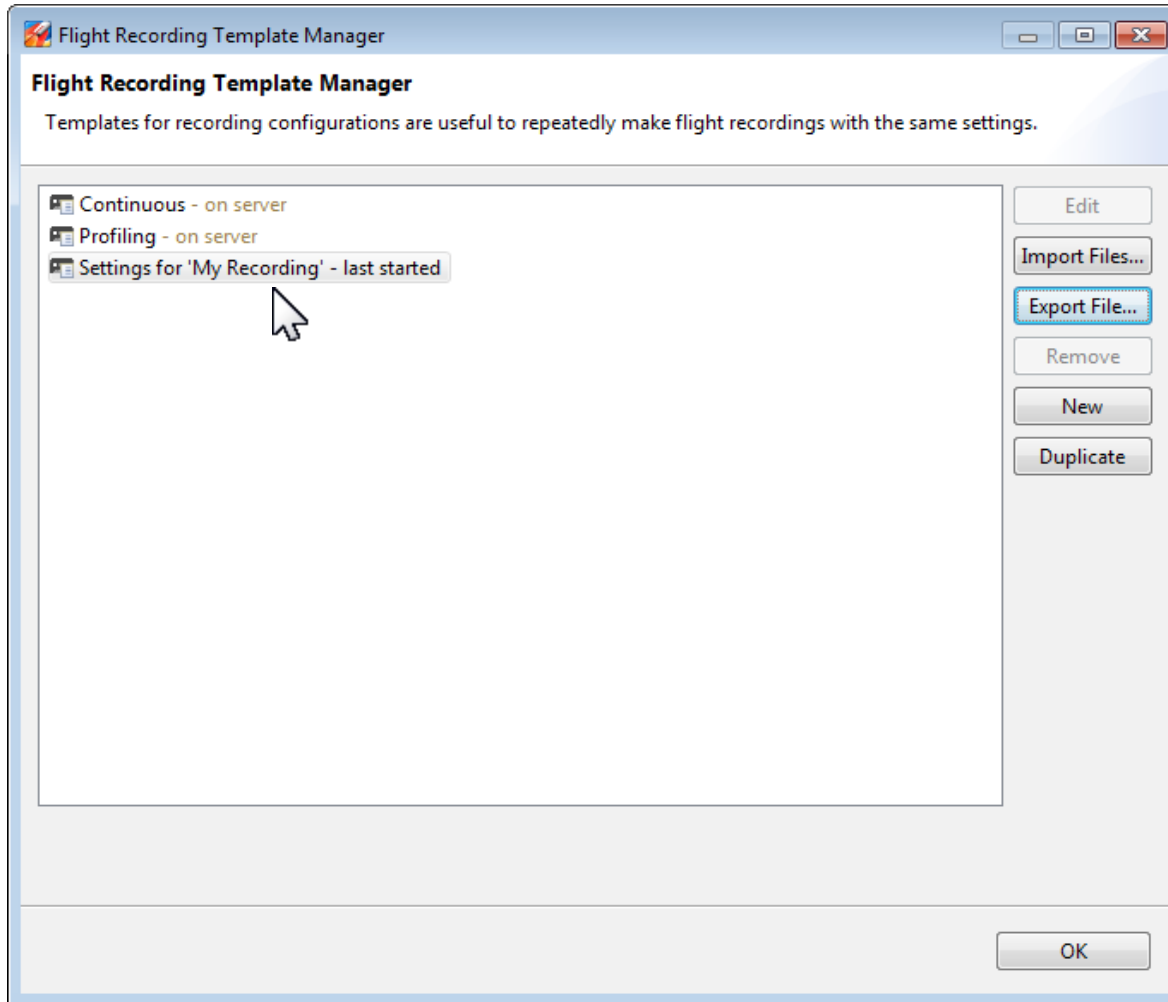
Alternative ways of getting recordings

- Add to the JVM options:

-XX:+UnlockCommercialFeatures -XX:+FlightRecorder

-XX:StartFlightRecording=duration=20s,settings=javafx.jfc,filename=javafx.jfr

Creating .jfc file



- Start Flight Recording... →
Template Manager →
Select 'Settings..' ->
Export File...

JMC – Event Overview

Overview

Interval: 19 s 957 ms (all) Synchronize Selection

9/23/14 2:52:34 PM 9/23/14 2:52:54 PM

Producers

Filter Column: Producer Show Only Operative Set

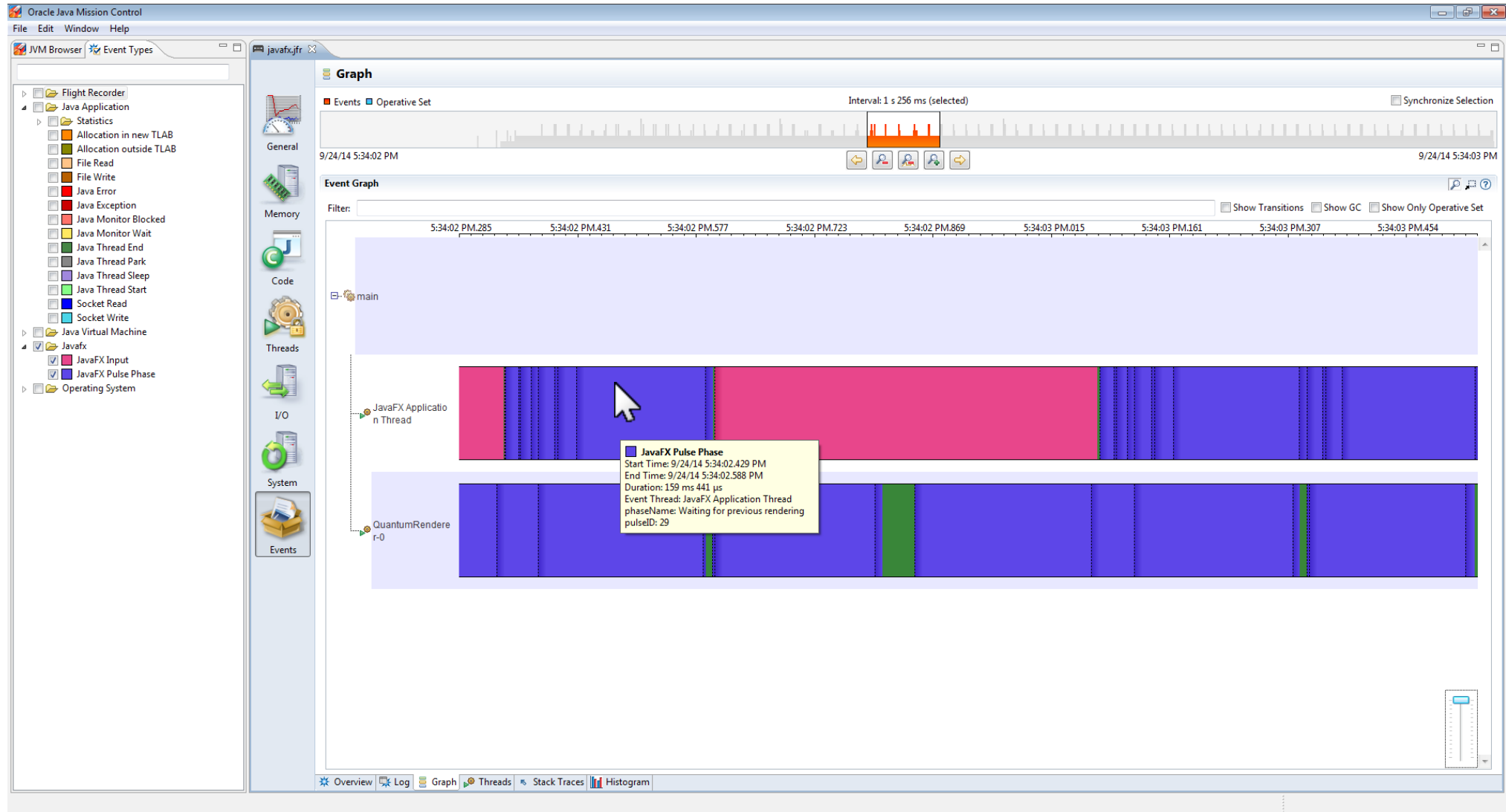
Producer	Total	Count
JavaFX producer	37 s 431 ms	930
HotSpot JVM	55 s 748 ms	290
Java Runtime	0 s	3

Event Types

Filter Column: Event Type Show Only Operative Set

Event Type	Total	Count
JavaFX Pulse Phase	35 s 452 ms	840
Java Thread Park	15 s 832 ms	164
Java Monitor Wait	39 s 916 ms	117
JavaFX Input	1 s 979 ms	90
Java Thread Start	0 s	9
Java Error	0 s	3

JMC – Event Graph



JMC – Event Graph (2)

Oracle Java Mission Control

JVM Browser Event Types

javafxjfr

Graph

Events Operative Set

Interval: 1 s 256 ms (selected)

9/24/14 5:34:02 PM

9/24/14 5:34:03 PM

Synchronize Selection

Event Graph

Filter:

Show Transitions Show GC Show Only Operative Set

5:34:02 PM.285 5:34:02 PM.431 5:34:02 PM.577 5:34:02 PM.723 5:34:02 PM.869 5:34:03 PM.015 5:34:03 PM.161 5:34:03 PM.307 5:34:03 PM.454

main

JavaFX Application Thread

QuantumRenderer-0

JavaFX Input (longest)
2 events in slot. Zoom in to see details.
Start Time: 9/24/14 5:34:02.600 PM
End Time: 9/24/14 5:34:03.071 PM
Duration: 471 ms 869 us
Event Thread: JavaFX Application Thread
inputType: VIEW_EVENT: RESIZE

Overview Log Graph Threads Stack Traces Histogram

JMC – Event Graph (3)

The screenshot shows the Oracle Java Mission Control (JMC) interface. The main window displays an Event Graph for a JavaFX application. The graph shows the execution of the main thread and two worker threads (JavaFX Application Thread and QuantumRenderer-0) over time. The x-axis represents time, with markers at 5:34:02 PM.279, 5:34:02 PM.380, 5:34:02 PM.481, 5:34:02 PM.582, 5:34:02 PM.683, 5:34:02 PM.784, 5:34:02 PM.885, 5:34:02 PM.986, and 5:34:03 PM.087. The y-axis lists the threads: main, JavaFX Application Thread, and QuantumRenderer-0. The graph shows various events and transitions for each thread. A tooltip is visible for a 'Java Thread Park' event, showing its duration and stack trace.

Java Thread Park
Start Time: 9/24/14 5:34:02.806 PM
End Time: 9/24/14 5:34:02.845 PM
Duration: 39 ms 301 μ s
Event Thread: QuantumRenderer-0
Class Parked On: java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject
Park Timeout: 0 s
Address of Object Parked: 0xA36F218

Stack Trace:
at sun.misc.Unsafe.park(boolean, long)
at java.util.concurrent.locks.LockSupport.park(Object)
at java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject.await()
at java.util.concurrent.LinkedBlockingQueue.take()
at java.util.concurrent.ThreadPoolExecutor.getTask()
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor\$Worker)
at java.util.concurrent.ThreadPoolExecutor\$Worker.run()
at com.sun.javafx.tk.quantum.QuantumRenderer\$PipelineRunnable.run()
at java.lang.Thread.run()

JMC – Event Histogram

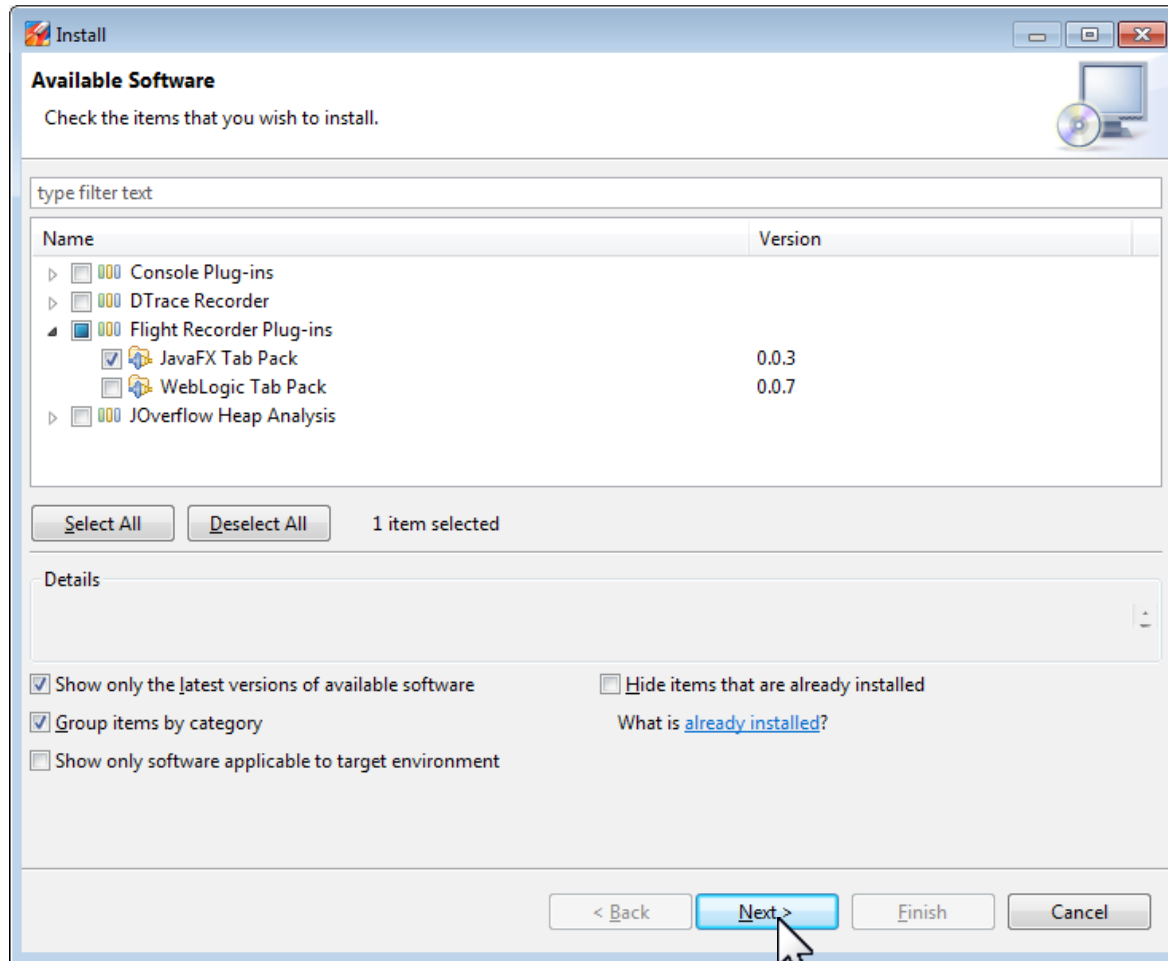
The screenshot shows the Oracle Java Mission Control (JMC) interface. The main window displays the 'Event Histogram' for a JavaFX application. The histogram shows a pulse phase event selected, with a table of event statistics below it.

Event Histogram Data:

Group By:	Total	Average	Count
Painting	1 s 10 ms	202 ms 44 µs	5
Waiting for previous rendering	159 ms 481 µs	39 ms 870 µs	4
Presenting	115 ms 633 µs	28 ms 908 µs	4
Pulse start	89 ms 335 µs	17 ms 867 µs	5
Update bounds	86 ms 520 µs	17 ms 304 µs	5
Copy state to render graph	34 ms 849 µs	8 ms 712 µs	4
Layout Pass	13 ms 383 µs	2 ms 676 µs	5
Render Roots Discovered	1 ms 933 µs	1 ms 933 µs	1
Dirty Opts Computed	118 µs 271 ns	118 µs 271 ns	1
CSS Pass	19 µs 396 ns	3 µs 879 ns	5

The interface also includes a left sidebar with a tree view of event types, a top menu bar, and a bottom navigation bar with tabs for Overview, Log, Graph, Threads, Stack Traces, and Histogram.

JMC – JavaFX Plugin Installation



- Help →
Install New Software...

JavaFX Plugin – Pulse Events

Oracle Java Mission Control

JVM Browser Event Types

The JVM Running Mission Control

General

Memory

Code

Threads

I/O

System

Java FX

Events

+ Pulse

Events Operative Set Interval: 20 s 64 ms (all) Synchronize Selection

9/25/14 3:45:42 PM 9/25/14 3:46:02 PM

Pulses

A pulse is a lap in the Java FX render loop

450 ms

400 ms

350 ms

300 ms

250 ms

200 ms

150 ms

100 ms

50 ms

0 s

3:45:46 PM 3:45:48 PM 3:45:51 PM 3:45:54 PM 3:45:57 PM 3:46:00 PM

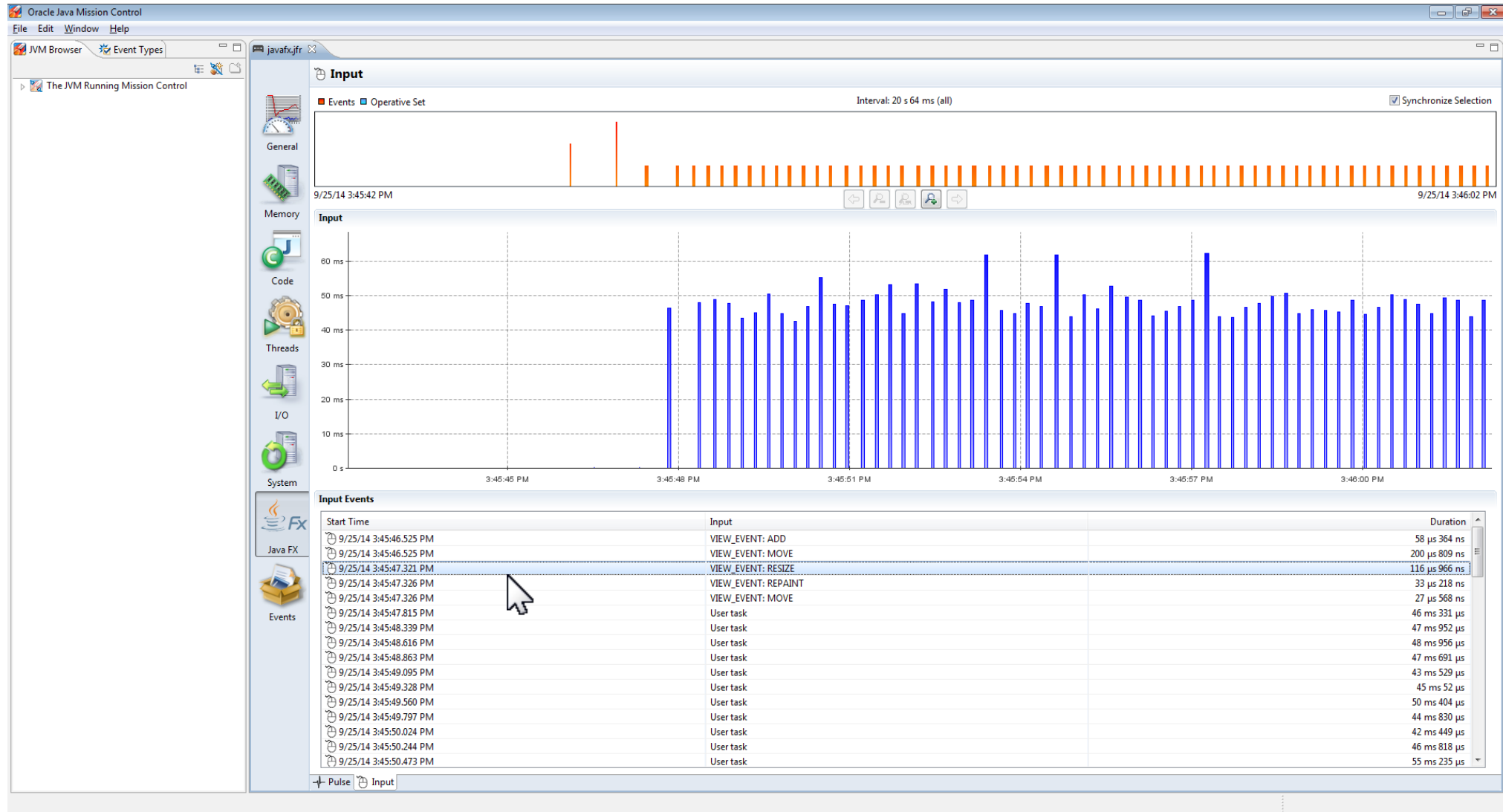
Pulses by ID

Pulse ID	Longest Phase
+1	360 ms 284 μ s
+2	415 ms 407 μ s
+3	223 ms 34 μ s
+4	209 ms 823 μ s
+5	213 ms 973 μ s
+6	217 ms 973 μ s
+7	224 ms 562 μ s
+8	214 ms 573 μ s
+9	207 ms 700 μ s
+10	216 ms 974 μ s
+11	223 ms 37 μ s
+12	206 ms 893 μ s
+13	255 ms 984 μ s
+14	226 ms 302 μ s
+15	214 ms 871 μ s
+16	227 ms 305 μ s

Pulse Phases for Selection

Start Time	Phase	Duration
9/25/14 3:45:47.863 PM	Pulse start	18 ms 899 μ s
9/25/14 3:45:47.882 PM	CSS Pass	3 μ s 401 ms
9/25/14 3:45:47.882 PM	Layout Pass	1 ms 893 μ s
9/25/14 3:45:47.884 PM	Update bounds	19 ms 971 μ s
9/25/14 3:45:47.904 PM	Waiting for previous rendering	415 ms 407 μ s
9/25/14 3:45:48.320 PM	Copy state to render graph	17 ms 354 μ s
9/25/14 3:45:48.337 PM	Dirty Opts Computed	13 ms 833 μ s
9/25/14 3:45:48.351 PM	Render Roots Discovered	29 ms 81 μ s
9/25/14 3:45:48.380 PM	Painting	216 ms 466 μ s
9/25/14 3:45:48.597 PM	Presenting	8 ms 137 μ s

JavaFX Plugin – Input Events



Implementation

PulseLogger

- Source roots:

```
jfx/rt/modules/base/src/main/java/
```

```
jfx/rt/modules/base/src/main/java-jfr
```

- Package

```
com.sun.javafx.logging
```

- Classes:

```
public PulseLogger
```

```
Logger
```

```
PrintLogger extends Logger
```

```
JFRLogger extends Logger imports com.oracle.jrookit.jfr
```

PulseLogger probes

```
import com.sun.javafx.logging.PulseLogger;
import static com.sun.javafx.logging.PulseLogger.PULSE_LOGGING_ENABLED;
...
    if (PULSE_LOGGING_ENABLED) { PulseLogger.newPhase("CSS Pass"); }
    if (PULSE_LOGGING_ENABLED) { PulseLogger.newInput("KEY_PRESSED"); }
    if (PULSE_LOGGING_ENABLED) { PulseLogger.addMessage("Dirty Region "+...); }
    if (PULSE_LOGGING_ENABLED) { PulseLogger.incrementCounter("Nodes rendered"); }
    if (PULSE_LOGGING_ENABLED) { PulseLogger.pulseStart(); }
    if (PULSE_LOGGING_ENABLED) { PulseLogger.renderStart(); }
...
```

PulseLogger probes in JavaFX runtime

- Source root:

`jfx/rt/modules/graphics/src/main/java`

- Phase events:

`javafx/scene/Scene.java`

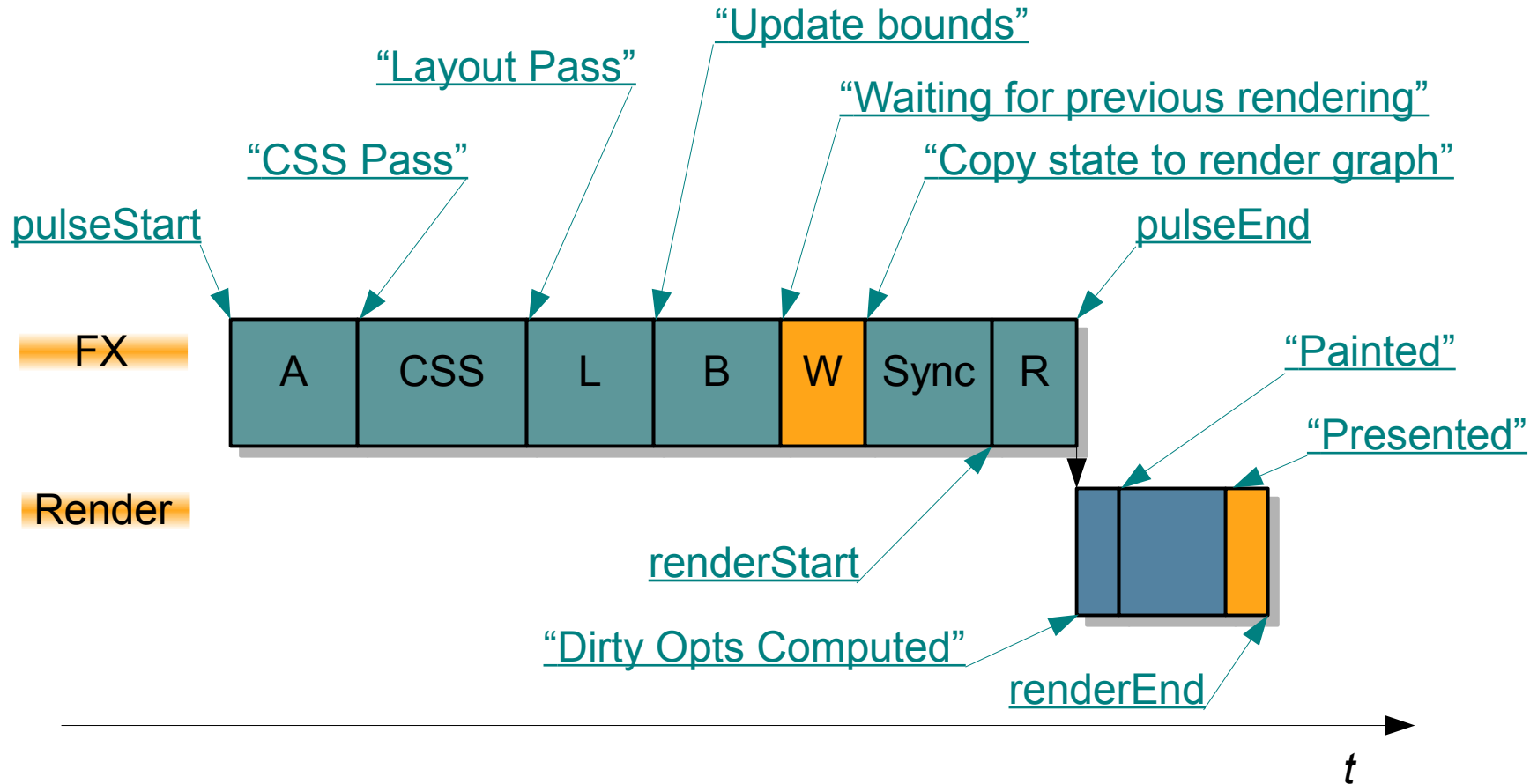
`com/sun/javafx/tk/quantum/PresentingPainter.java`

`com/sun/javafx/tk/quantum/ViewPainter.java`

- Input events:

`com/sun/javafx/tk/quantum/GlassViewEventHandler.java`

PulseLogger diagram



PulseLogger probes in user code

- In animation handlers

```
if (PULSE_LOGGING_ENABLED) { PulseLogger.newPhase("My phase"); }
```

- In any user code executed on the JavaFX thread:

```
if (PULSE_LOGGING_ENABLED) { PulseLogger.newInput("My event"); }
```

```
...
```

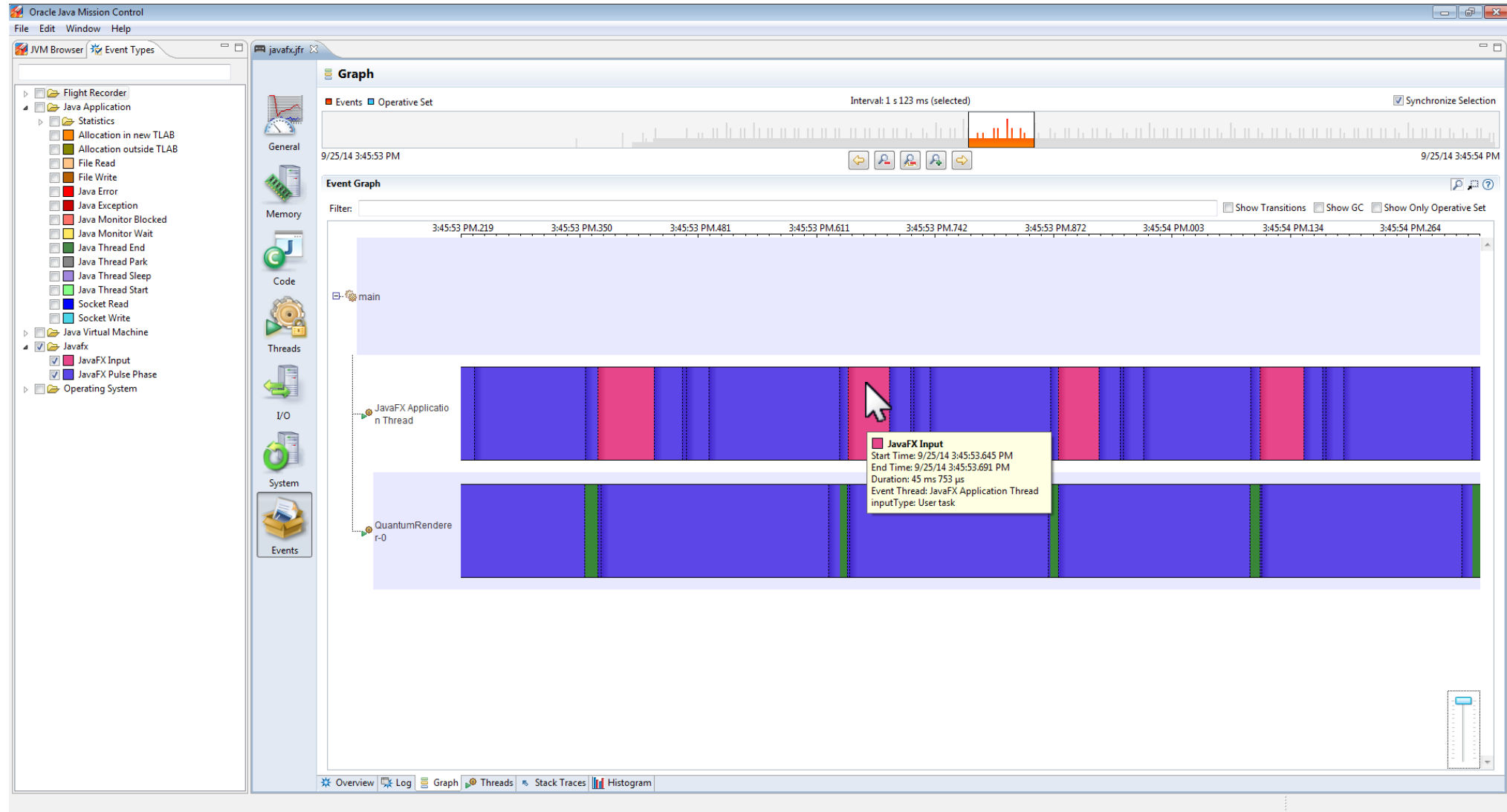
```
if (PULSE_LOGGING_ENABLED) { PulseLogger.newInput(null); }
```

PulseLogger probes in user code (2)

- Example

```
Platform.runLater(() -> {  
    if (PULSE_LOGGING_ENABLED) { PulseLogger.newInput("User task"); }  
    <user task code>  
    if (PULSE_LOGGING_ENABLED) { PulseLogger.newInput(null); }  
});
```


PulseLogger probes in user code (3)



Resources

- Oracle documentation

<http://oracle.com/missioncontrol>

- Marcus Hirt's blog

<http://hirt.se/blog/>

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



JavaOne™

ORACLE®