

Using the E4 spies to debug your Eclipse Application



Presentation

	<p>Olivier Prouvost</p> <ul style="list-style-type: none">▫ Eclipse expert, trainer and committer (platform.ui, E4 tools, PDE)▫ olivier.prouvost@opcoach.com▫ @OPCoach_Eclipse
	<ul style="list-style-type: none">- OPCoach founded in June 2009 in Toulouse- Member of the Eclipse Foundation (as Contributing member)- Web site: https://www.opcoach.com- Provide Eclipse training and consulting in French and English <p>•</p>

Agenda

- Spy general overview
- What can we debug with those spies?
 - bundle spy
 - model spy
 - context spy
 - event spy
 - css spy
 - preference spy

What is a 'spy' in Eclipse 4?

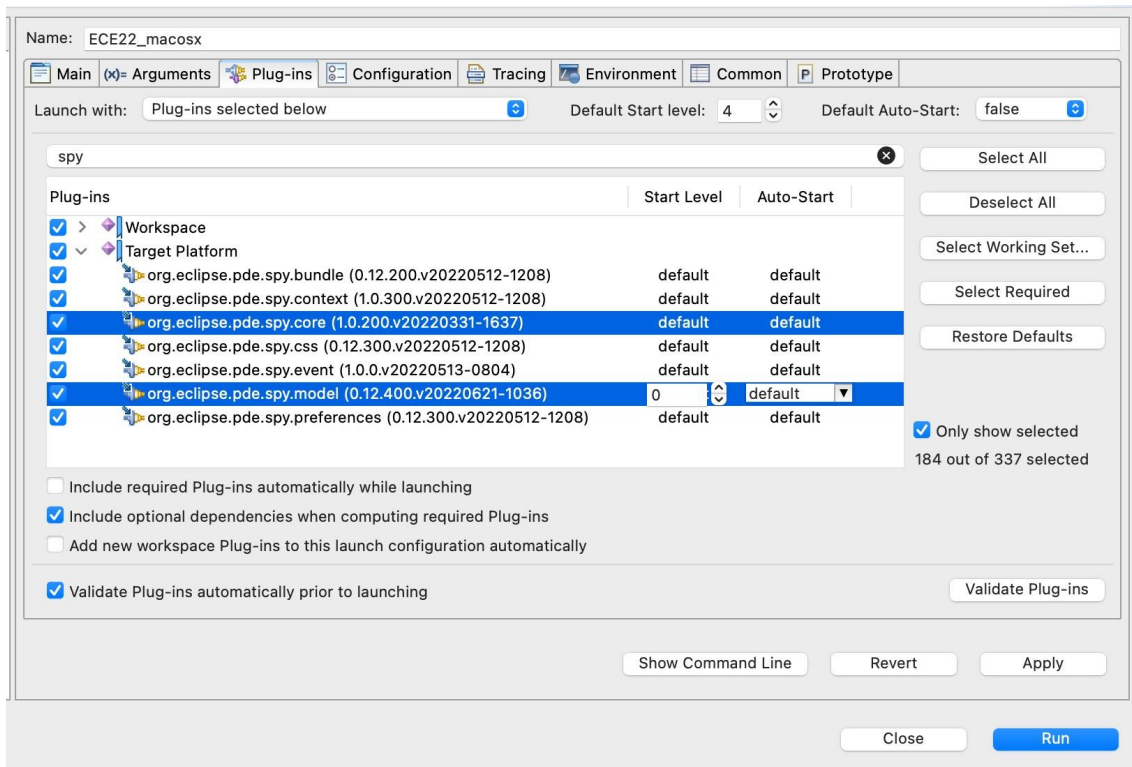
- A spy displays dynamic information from a developer point of view
- They are not supposed to be delivered to the end user
- E4 spies are used to display each E4 concept in one main dashboard:
 - application model, injection contexts, event, css, preferences ...
- The E4 spy platform can be easily extended to define its own spy (global dashboard for a concept).

Where are the spies defined ?

- Up to the 2022-03 release, the spies were managed in the E4 tools project
- Since 2022-06, all the spies are included in the PDE project.
 - Nothing to install anymore
- They are directly available in the IDE for RCP and RAP developer.
- To use the spies in your project, add the **org.eclipse.pde.feature** in your target platform

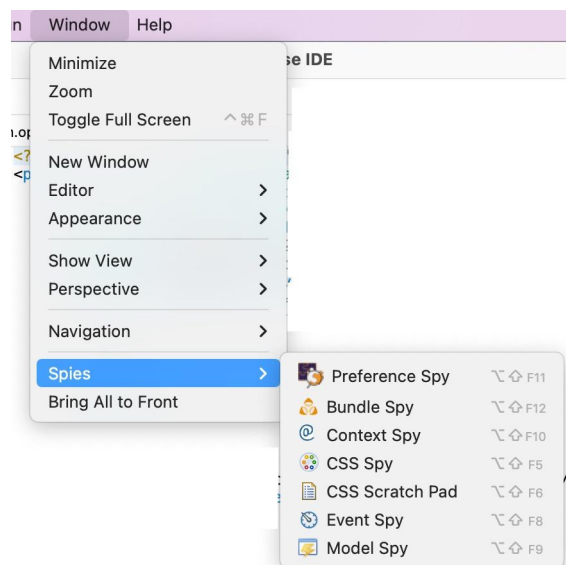
How to launch the spies?

- Your application must be launched with
 - at least one of the spies available
 - the **org.eclipse.pde.spy.core** plugin
- **NO DEPENDENCY IS NEEDED**



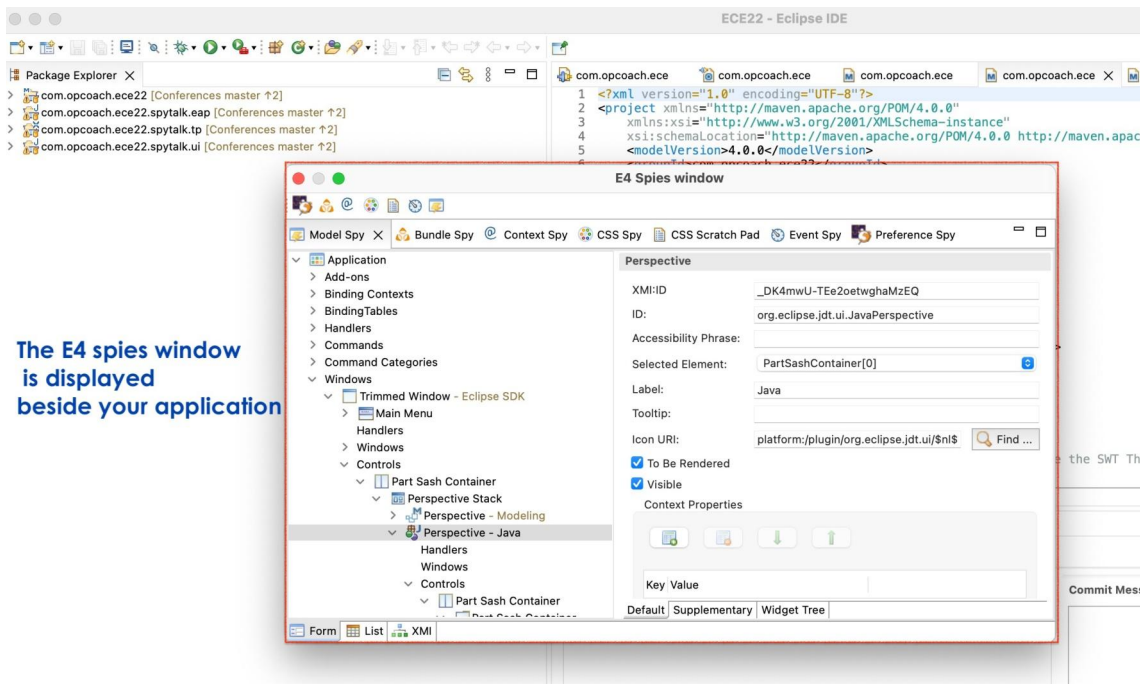
How to display the spies?

- The spy feature provides a new submenu in the 'Window' menu
- If your application does not contain this Window menu, it will be added
- All the defined spies are listed in the spies submenu :



The E4 Spies window

- All the spies will open in the E4 spy window
 - But you can move them in another window after



The E4 spies window is displayed beside your application

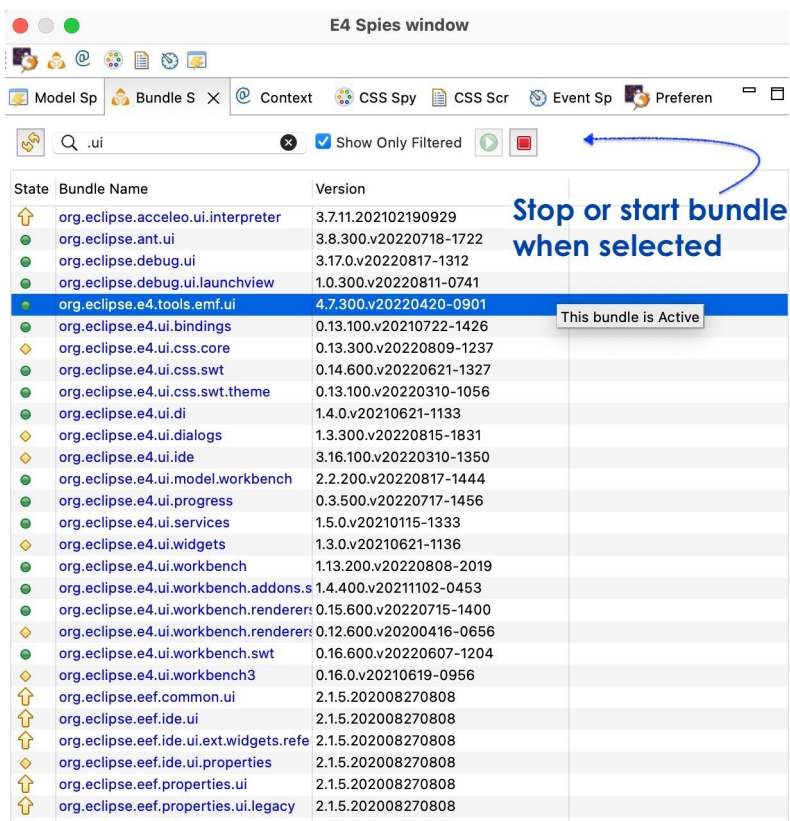
Spies overview

Let's have a look on the different spies and what are their benefits:

- bundle
- model
- context
- event
- css
- preference

Bundle Spy

- This is an OSGi level spy
- It displays all bundles launched in your application with their OSGi status
- It is possible to start or stop any bundle.



What can we debug with the bundle spy?

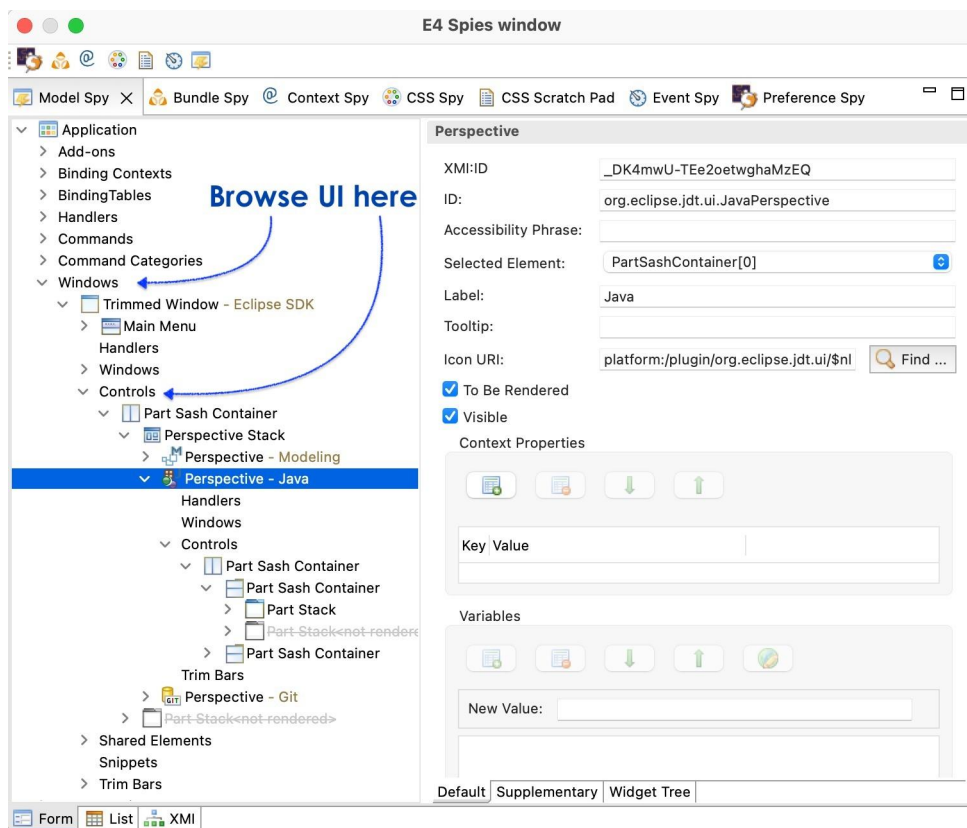
If you don't see a plugin contribution in your application:

- Check if your plugin is present at runtime
- Check if your plugin is started as expected (or at least starting)

This is usually the first check when the expected contribution is not present.

Model Spy

- The application model describes the content of the application
- The model spy displays the live model and allows modifications
 - positions in sash containers
 - perspectives layouts



What can we debug with the model spy?

- Check real IDs and control if they are used correctly
- Check why a view is not displayed
- Check disabled/enabled commands
- Fix global layout
- Control the UI components
- Check if model fragments are correctly included in live model
- Modify the live model
- and many other tips

Model spy demo

Demo

- checking IDs for objects (main menu, main toolbar...)
- show control
- changing layout
- checking parts/part descriptors

E4 context and injection.

- Injection is used everywhere in the E4 runtime.
- **@Inject** fields or methods will use values stored in the E4 context.
- The E4 context is a tree of maps containing keys (class or string) and instances.
- When an instance is replaced in the context, E4 will:
 - re-invoke the **@Inject** method having this instance as a parameter
 - re-initialize the **@Inject** field with the new instance
- But what can we inject in the code?

Context spy

- The context spy is used to browse this tree of maps.
- The context spy will display all available values at any level
 - root level (OSGi) : services instances (available everywhere in the code)
 - application level values (available everywhere in the code)
 - lower level values (available in a part or in a handler code)

The screenshot shows the 'E4 Spies window' with a search filter set to 'active'. The tree of maps is expanded to show the 'MultiPageEditorSite' context. A table below the tree lists the keys and values for the selected context.

Key	Value
Inherited values injected or updated using this context	
Local values managed by this context	
@ activeChildContext	TrimmedWindowImpl (IDEWindow) Context
(x) activeContexts	[org.eclipse.ui.contexts.window, org.eclipse.jdt.ui.JavaActionSet, org.eclipse.jdt.ui.JavaElement
@ activeEditor	java.lang.Object@83e0adf
● activeFocusControl	java.lang.Object@83e0adf
@ activeFocusControlId	java.lang.Object@83e0adf
● activeMenu	[]
● activeMenuEditorInput	java.lang.Object@83e0adf
@ activeMenuSelection	java.lang.Object@83e0adf
@ activeShell	Shell {E4 Spies window} [layout=org.eclipse.e4.ui.workbench.renderers.swt.TrimmedPartLayout
@ activeWorkbenchWindow	org.eclipse.ui.internal.WorkbenchWindow@31f96cf
@ activeWorkbenchWindow.activePerspective	org.eclipse.jdt.ui.JavaPerspective
● activeWorkbenchWindow.isCoolbarVisible	true
● activeWorkbenchWindow.isPerspectiveBarVisible	true
● activeWorkbenchWindow.isStatusLineVisible	true
● activeWorkbenchWindowShell	Shell {ECE22 - Eclipse IDE} [layout=org.eclipse.e4.ui.workbench.renderers.swt.TrimmedPartLay
(x) e4ActivePart	ora.eclipse.pde.sov.context.ContextSovPart=ora.eclipse.e4.ui.model.application.ui.basic.imol.P

What can we debug with the context spy?

- Check the available values provided by the E4 runtime
- Find keys for objects (class name or specific String)
- Check that specific objects added manually are present
 - **context.set(MyClass.class, myInstance)**
- Control if an object is at the right level (when not injected)

Context Spy demo

Demo : display some instances in the contexts

Event spy

The E4 runtime provides an API to send easily events using the **IEventBroker**

```

9 public class EngineWatcher
10 {
11     // Define the sent topics          Define event keys
12     public static final String ALARM_TOPIC = "Alarm/*";
13     public static final String ALARM_RPM_TOO_HIGH = "Alarm/RpmTooHigh";
14     public static final String ALARM_SPEED_TOO_HIGH = "Alarm/SpeedTooHigh";
15
16     // Get the event broker by injection
17     @Inject
18     IEventBroker ebroker;    Receive the Event Broker using injection
19
20     @Optional
21     @Inject
22     public void checkRpmValue(final @Named(EngineSimulator.ENGINE_RPM_VALUE) int value)
23     {
24         // Inject the RPM value from context
25         if (value > 5000)
26         {
27             // Send an alarm
28             Alarm a = new Alarm("rpm is too high (" + value + ")", value);
29             ebroker.send(ALARM_RPM_TOO_HIGH, a);
30         }
31     }
32 }

```

Send the event

Sending events

Receiving events

And to receive the event using injection:

```

118
119     @Inject @Optional
120     public void listenToAlarms(@UIEventTopic(EngineWatcher.ALARM_TOPIC) Alarm a)
121     {
122         // Receive event by injection
123         alarms.insertElementAt(a, 0);
124         if (viewer != null)
125         {
126             viewer.refresh();
127             viewer.setSelection(new StructuredSelection(a));
128         }
129     }

```

Receiving events

Available events

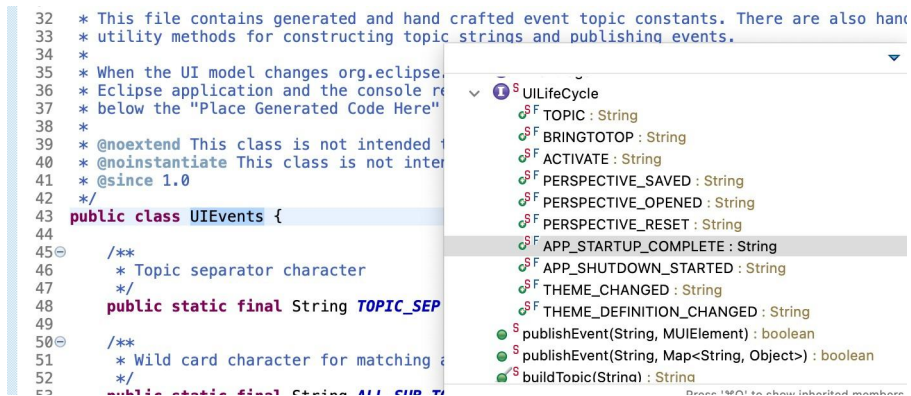
- An event is sent for each UIModel component modification (window move, ...)
- Business events sent using the IEventBroker in your code

- An event is also sent during each step of the E4 application's life cycle:

```

32 * This file contains generated and hand crafted event topic constants. There are also han
33 * utility methods for constructing topic strings and publishing events.
34 *
35 * When the UI model changes org.eclipse
36 * Eclipse application and the console re
37 * below the "Place Generated Code Here"
38 *
39 * @noextend This class is not intended
40 * @noinstantiate This class is not inter
41 * @since 1.0
42 */
43 public class UIEvents {
44
45     /**
46      * Topic separator character
47      */
48     public static final String TOPIC_SEP
49
50     /**
51      * Wild card character for matching
52      */
53     public static final String ALL_SUB_T

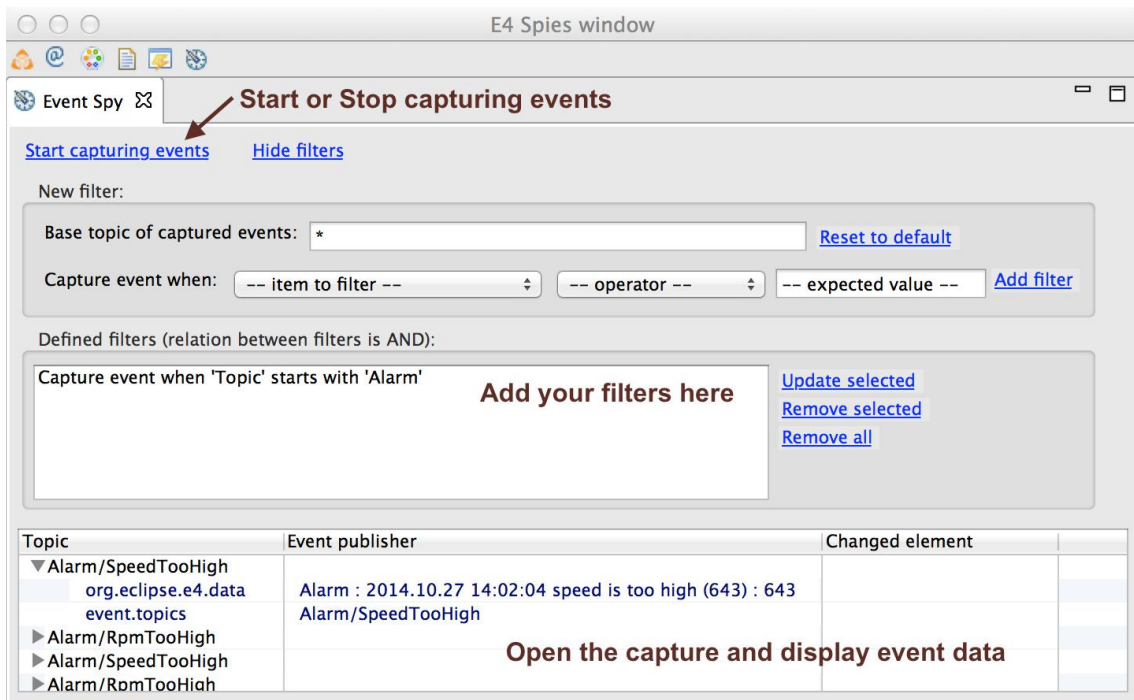
```



The screenshot shows the Eclipse IDE with the `UIEvents` class open in the editor. The class contains several constants for event topics, such as `TOPIC`, `BRINGTOTOP`, `ACTIVATE`, `PERSPECTIVE_SAVED`, `PERSPECTIVE_OPENED`, `PERSPECTIVE_RESET`, `APP_STARTUP_COMPLETE`, `APP_SHUTDOWN_STARTED`, `THEME_CHANGED`, and `THEME_DEFINITION_CHANGED`. There are also methods for publishing events and building topic strings.

How to trace these events?

- The Event spy will receive all events and display their values.



Event Spy

What can we debug with the Event spy?

The Event Spy shows the generated events

It can also filter them on their topic

Demo:

open the event spy, listen to events and display events when the window is moving or resizing.

CSS Spy

- The E4 renderer can manage CSS on the application.
- The CSS is applied with the **applicationCSS** parameter
 - A CSS may include other CSS using the **@import url** notation
- But... how can we find the values for the different widget's properties?

CSS Spy

- The CSS Spy will:
 - display the possible values
 - browse the widget hierarchy
 - provide a scratch pad to test the values.

Select object in UI

Change a value

Get the CSS fragment

Property	Value
background	
background-color	#e8e8e8
background-image	none
color	#000000
cursor	
eclipse-perspective-keyline-...	
font	
font-adjust	

```

Canvas {
  /* actual values */
  background-color: #e8e8e8;
  background-image: none;
  color: #000000;
  font-family: "Lucida Grande";
  font-size: 11;
  font-style: normal;
  font-weight: normal;
  swt-background-mode: none;
  text-transform: none;
  visibility: visible;
}
    
```

SWT Style Bits:
SWT.LEFT_TO_RIGHT
SWT.DOUBLE_BUFFERED

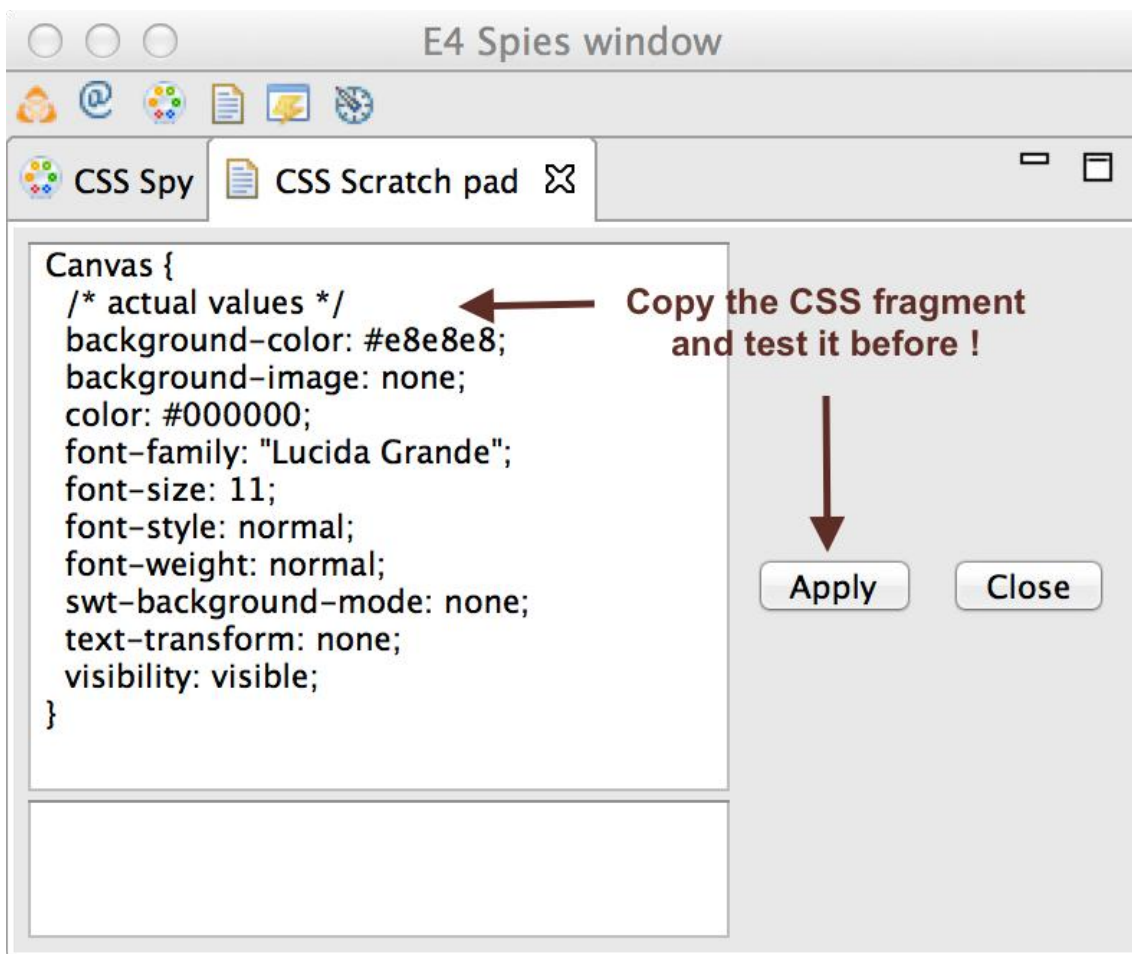
CSS Class Element:
org.eclipse.e4.ui.css.swt.dom.CompositeElement

SWT Layout: null
Bounds: x=-320 y=98 h=234 w=594

CSS Spy

The CSS Scratch pad

- Use it to test your CSS fragment on your application
- When it is ok, put it in your final CSS file
- CSS Scratchpad needs to run with **org.eclipse.ui.themes** (or an extension of **org.eclipse.e4.ui.css.swt.theme**)



CSS scratch pad

What can we debug with the CSS Spy?

- What are the possible values for UI properties in CSS?
- What is the CSS class ID of my UI object?
- Are my widgets in the UI well organized?

CSS Spy

Demo : open the css spy, select a widget and change one attribute. Then copy a css fragment and test it in the scratch pad.

Preference Spy

Preference spy will be used to:

- display default and instance values for a plugin
- search who defines a specific value

Enter the plugin name and get the values, or enter a value and find who defines it:

Nodepath	Key	Old Value	New Value
/default/org.eclipse.jdt.ui	semanticHighlighting.method.color	0,0,0	0,0,0
/default/org.eclipse.jdt.ui	closeBrackets	true	true
/default/org.eclipse.jdt.ui	java_doc_tag_italic	false	false
/default/org.eclipse.jdt.ui	semanticHighlighting.typeArgument.underline	false	false
/default/org.eclipse.jdt.ui	semanticHighlighting.interface.strikethrough	false	false
/default/org.eclipse.jdt.ui	semanticHighlighting.autoboxing.enabled	false	false
/default/org.eclipse.jdt.ui	semanticHighlighting.number.enabled	false	false
/default/org.eclipse.jdt.ui	pf_coloring_comment_bold	false	false
/default/org.eclipse.jdt.ui	editor_folding_default_imports	true	true
/default/org.eclipse.jdt.ui	Refactoring.savealleditors	false	false
/default/org.eclipse.jdt.ui	cleanup.make_private_fields_final	true	true
/default/org.eclipse.jdt.ui	semanticHighlighting.annotationElementReference.strike	false	false
/default/org.eclipse.jdt.ui	java_operator	0,0,0	0,0,0
/default/org.eclipse.jdt.ui	org.eclipse.jdt.ui.template.format	true	true
/default/org.eclipse.jdt.ui	semanticHighlighting.abstractMethodInvocation.italic	false	false
/default/org.eclipse.jdt.ui	cleanup.insert_inferred_type_arguments	false	false
/default/org.eclipse.jdt.ui	semanticHighlighting.annotation.enabled	true	true
/default/org.eclipse.jdt.ui	linkedPositionColor	121,121,121	121,121,121
/default/org.eclipse.jdt.ui	semanticHighlighting.method.strikethrough	false	false
/default/org.eclipse.jdt.ui	java_doc_link_italic	false	false

What can we debug with the Preference Spy?

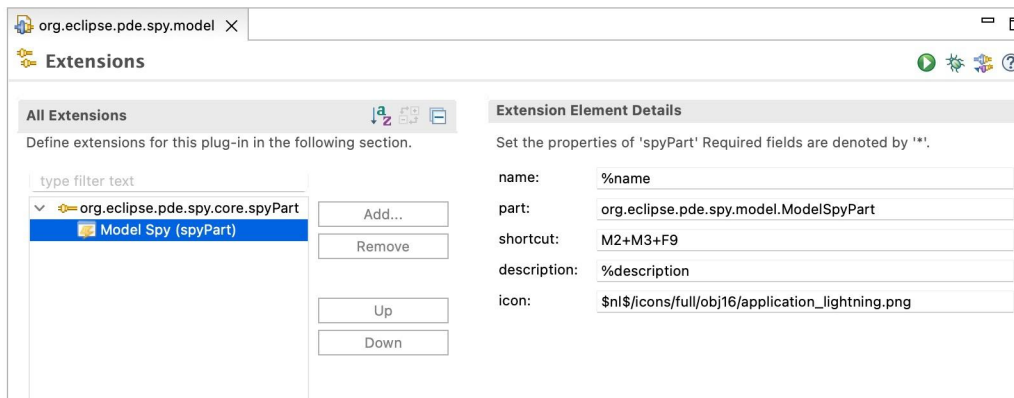
- conflicts with IDs between preferences (when defined in the same plugin)
- Quickly check the values (default and instance)
- Browse all preferences provided by a specific plugin
- Find unknown hidden preferences that are not visible

Preference Spy

Demo : filter preferences on a plugin. Open a preference page, find the plugin and display preference values (usually in core). Find who defines the color 100,70,50

Creating a new spy

- It is very easy to add your own spy
- Add a dependency to **org.eclipse.pde.spy.core**
- Extend **org.eclipse.pde.spy.core.spyPart**
- Your part is a pure E4 part (POJO, Injection...)
- The key binding is handled automatically
- It takes the time to create the UI part



What can we debug with your own spy?

- Basically everything you need to monitor.
 - Network transfers (mqtt, http..)
 - Memory management
 - Adapters between objects
 - Any stats on your application
 - Dashboard of your main business objects
 - ...

Q/A

- Thank you for attending!
- Don't forget to evaluate this talk



- You can download the pdf of this presentation on the eclipse con web site
- Keep in touch :
 - olivier.prouvost@opcoach.com
 - <https://www.opcoach.com>