Still Eclipse 3 ? Really ?

Table des matières

Still Eclipse 3 ? Really ?

Who ?

Olivier Prouvost :

- OPCoach (www.opcoach.com)
- Eclipse expert since 2004
- Provide training and consulting on Eclipse technologies
- France / Europe ... or even further
- Committer on e4 tools and platform UI
- **olivier@opcoach.com**
- Twitter : **@OPCoach_Eclipse**

Agenda and Goals

This talk will give you :

- A short introduction on E4 advantages and the tooling you need
- Demos on how you can use the E4 concepts
  - **to create your new plugins (even if you have legacy E3)**
  - **to interact with E3 legacy plugins**

This talk is for all Eclipse RCP Developers (E3 but also E4)

Reasons to change

The disadvantages of Eclipse 3.x :

- Eclipse 3.0 code delivered 14 years ago (2004) and Juno (3.8 - 2012)
- Many UI extension points and extensions all over the framework
- Strong dependency to the framework with inheritance
- No uniform API
- Many singletons and no OSGi services
- No Injection
- No separation between content and appearance (no renderer)
- Very hard to style the application

Advantages of Eclipse 4 runtime

- Application model is UI agnostic (SWT, Java FX...) thanks to POJOs
- Injection reduces the amount of code and simplifies testing
- Eclipse 4 event notifier is very concise and easy to use with injection
- The UI customization is easier thanks to :
  - CSS
  - renderers that can be overridden

The application model

- It describes the UI hierarchy of the application and other components
- Its API is an EMF API

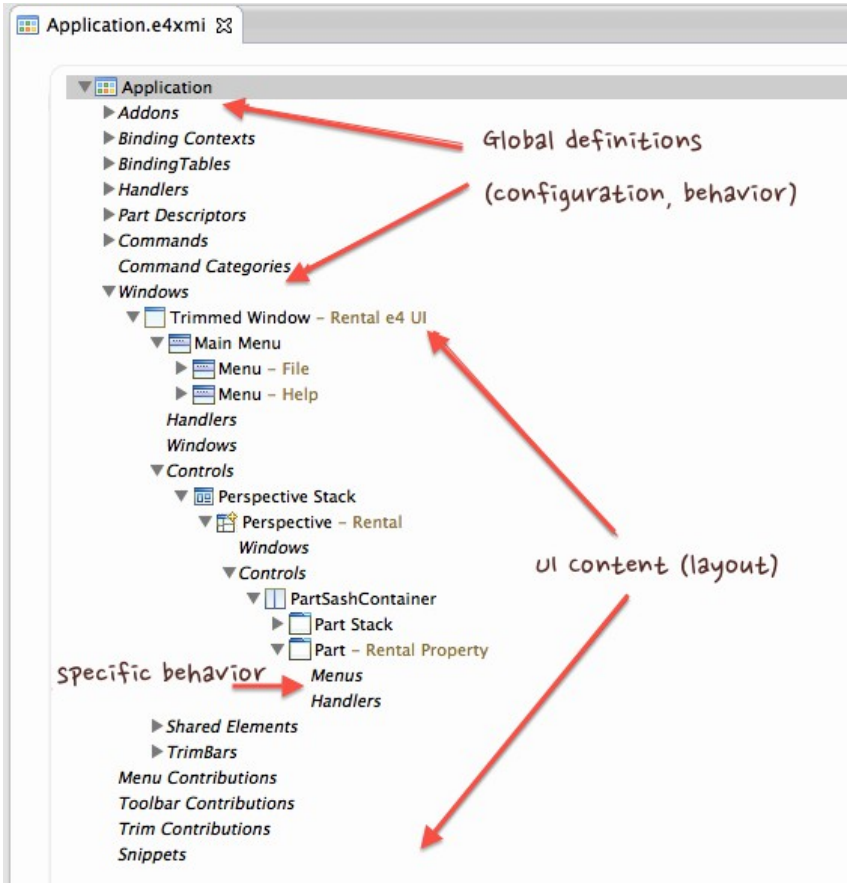
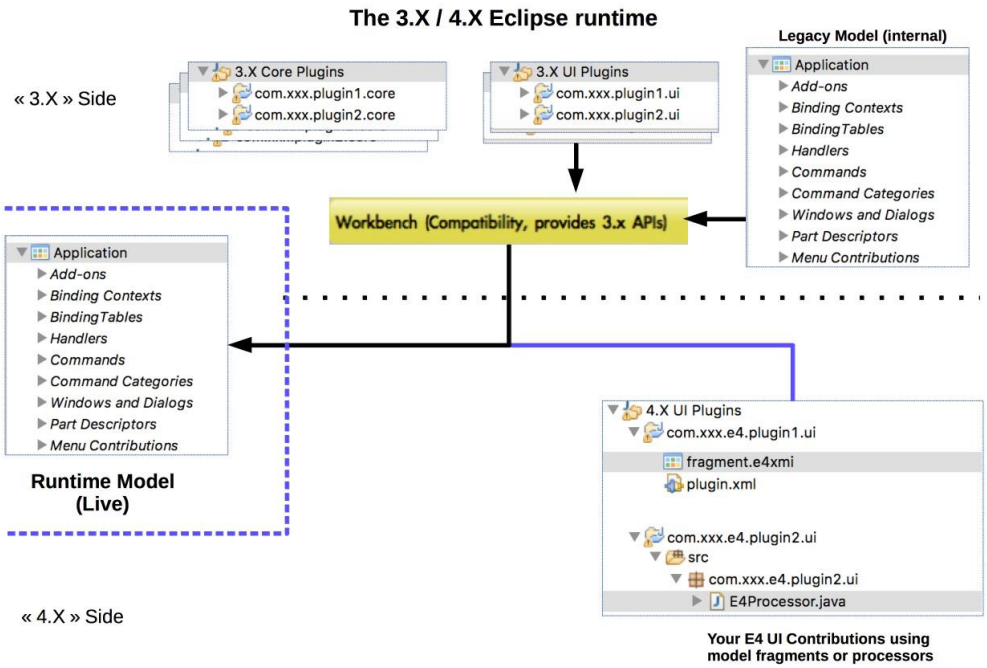


Image 1 Application Model

Fragments, processors and migration

Fragments and processors are the key concept for your development:



Warning: when using the compatibility mode the legacy model is not editable

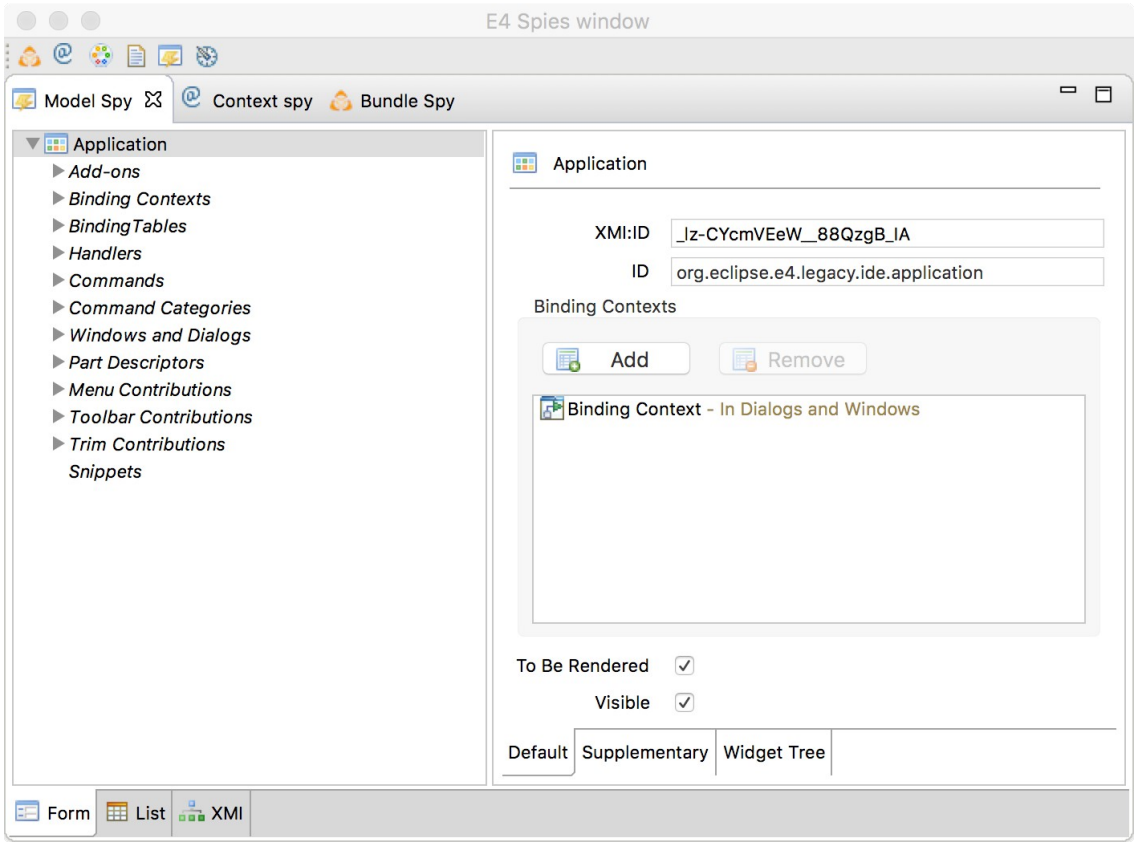
E4 Tooling

There are 2 major tools to use

- the E4 spies
- the fragment model editor

The E4 spies

- Once you are running using a 4.X runtime
- Whatever your development (3.X or 4.X way)
  - **install the E4 spies**
  - **pick them up from the Eclipse Market Place**
- You can look at the live model using the model Spy (Alt Shift F9)

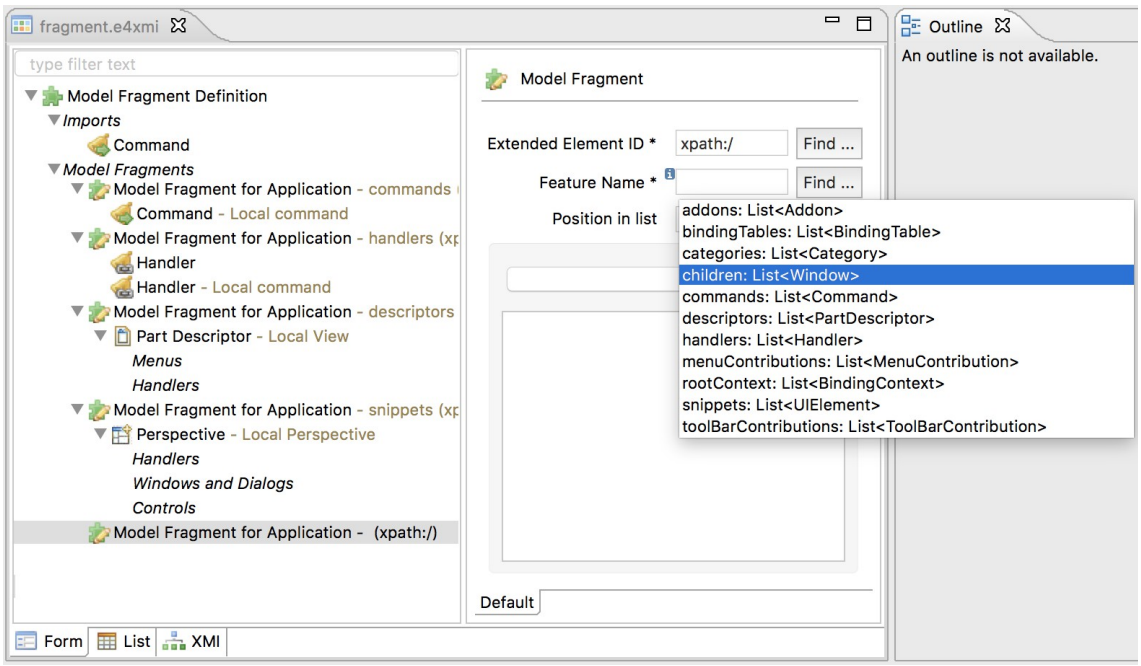


e4 Spies Window

The application fragment Editor

The fragment editor is used to edit some parts of the model

It is THE major tool for your E4 development



Demo

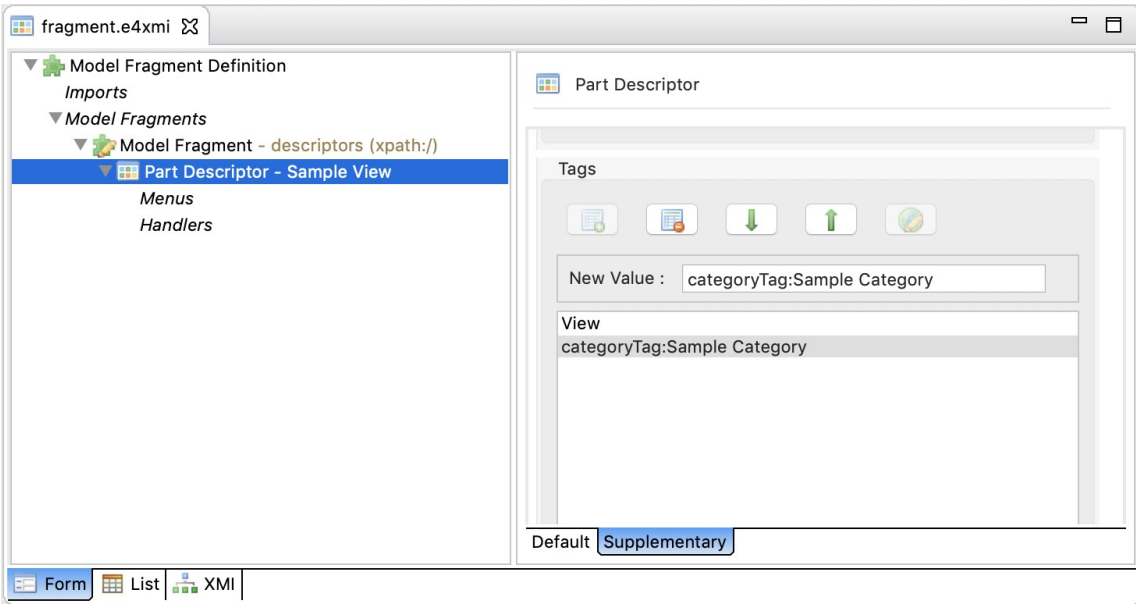
In this second part, we will:

- create a view using a model fragment
- add command to this view
- add command to a legacy E3 view
- create a perspective mixing E3 and E4 views
- manage both E3 and E4 selection
- show shortly how to test the E4 views

Defining your views

- Use a model fragment with : **xpath:/** and the **descriptors** feature
- Define your view using a **PartDescriptor**
- Add your menus and/or specific handlers

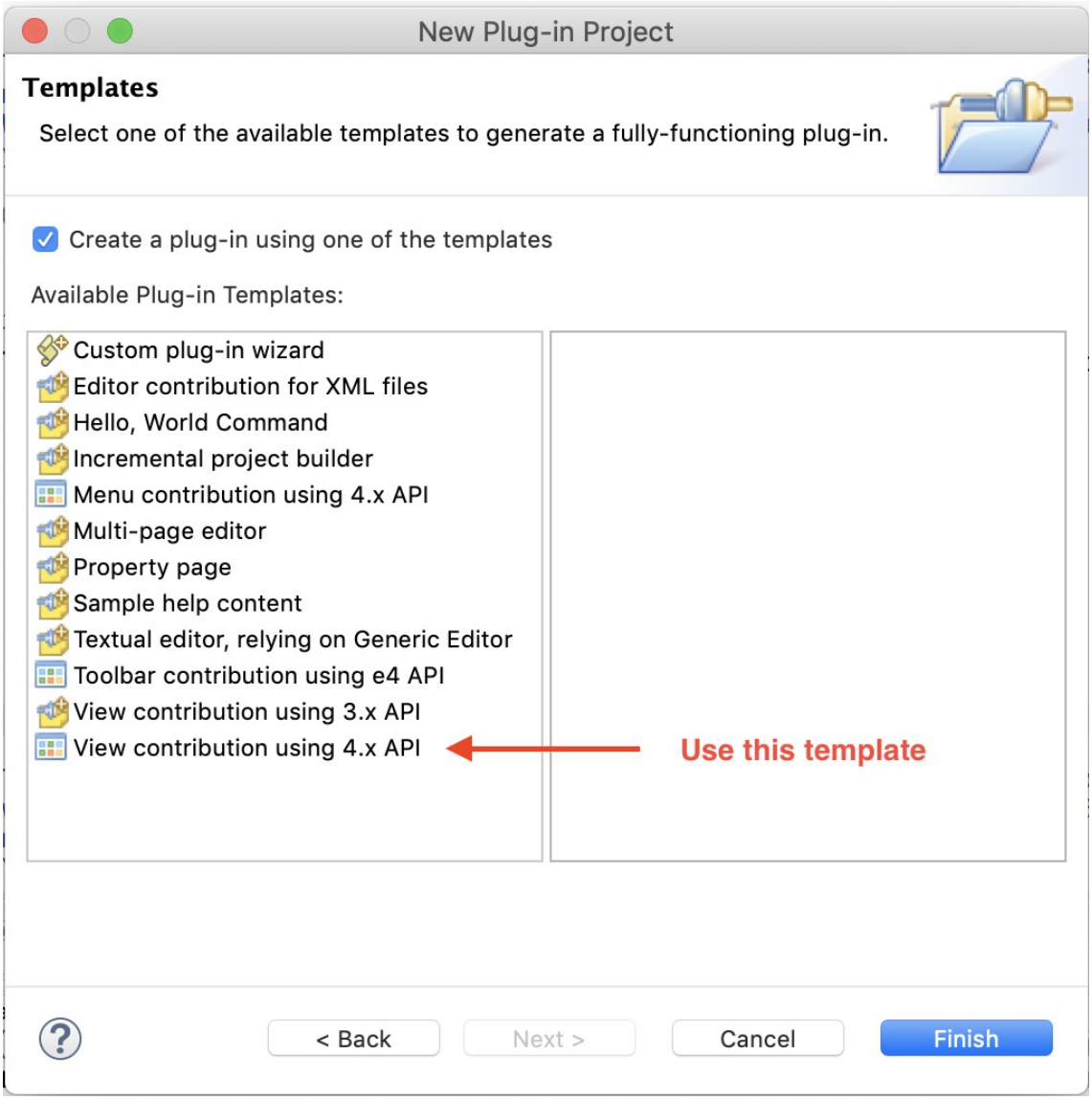
- To see your view in the menu Window->Show view, add the specific tags :



Part Descriptor in Fragment

Demo 1 : defining a view in E4

Demo : use the plugin template.



Adding existing commands in this view

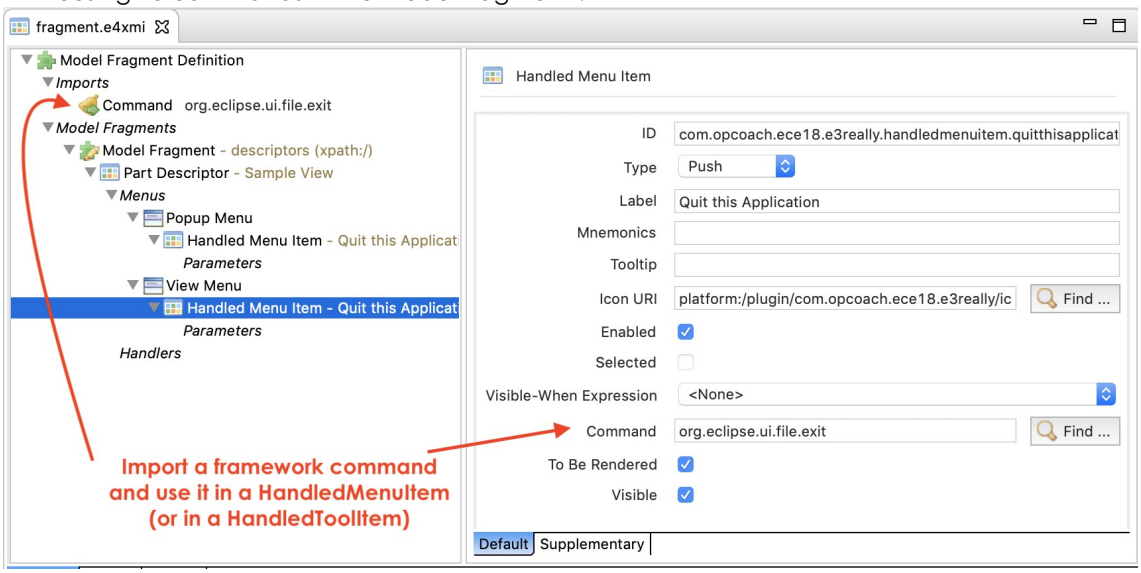
The E4 framework already defines some standard commands (copy, paste, exit...).

To reuse an existing command we just have to :

- import its ID in the model fragment
- optionally redefine a behavior in the handler
- use this command in any Menu or Tool Item

Command import

Reusing E3 commands in the model fragment :



Demo2 : importing E3 commands

- add the existing Quit command in the view menus

Adding new commands in the view

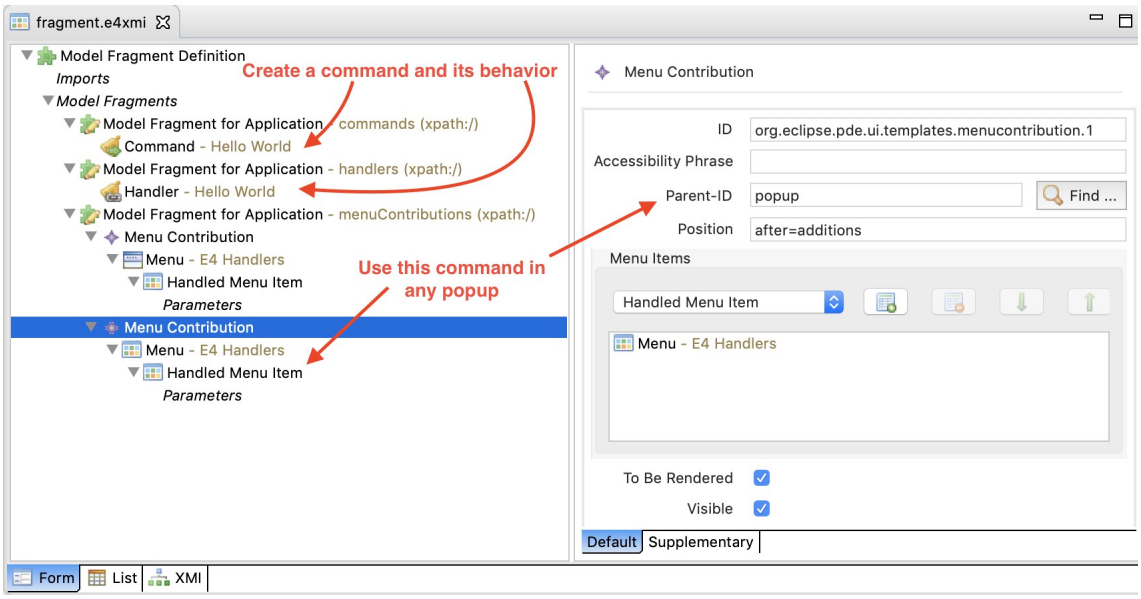
To add a new command in the application we need to :

- define the command in a fragment with **xpath:/** on **commands**
- define the behavior in a fragment with **xpath:/** on **handlers**
- use this command in any Menu or Tool Item

This can be done in the current fragment or in any other model fragment

New command definition

Creating new commands in the model fragment :



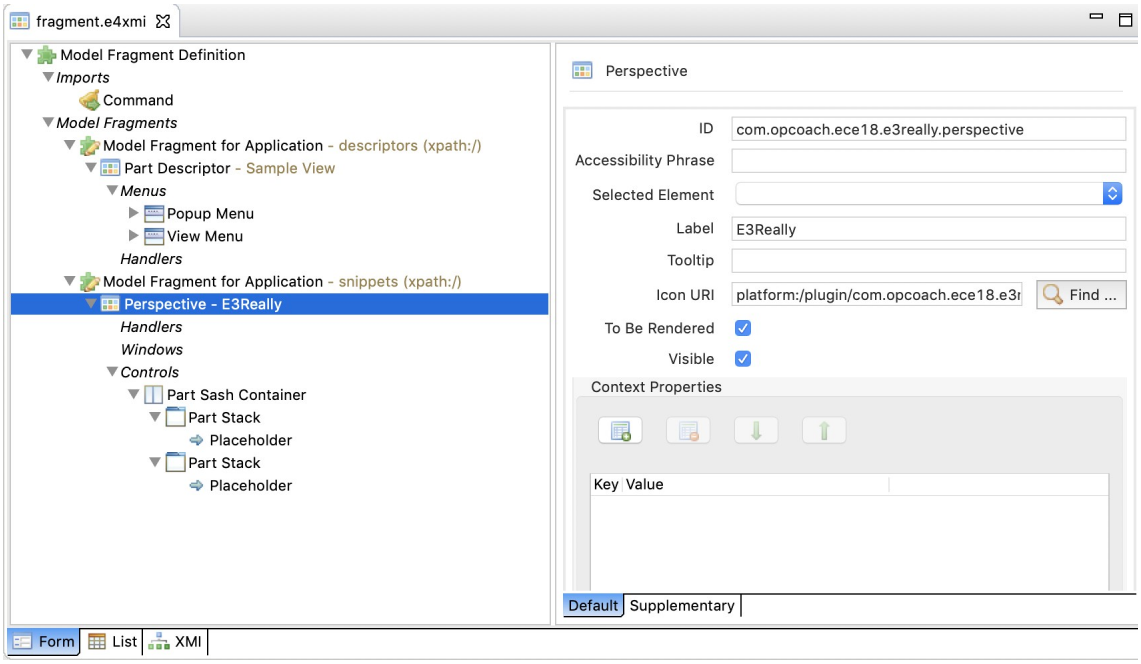
Demo 3 : using new E4 command

We will add **in another plugin**:

- a new Hello World command with its handler
- add this command in the main menu
- add this command in the popup of any view in the application
  - but it could be also added in a specific view using its ID

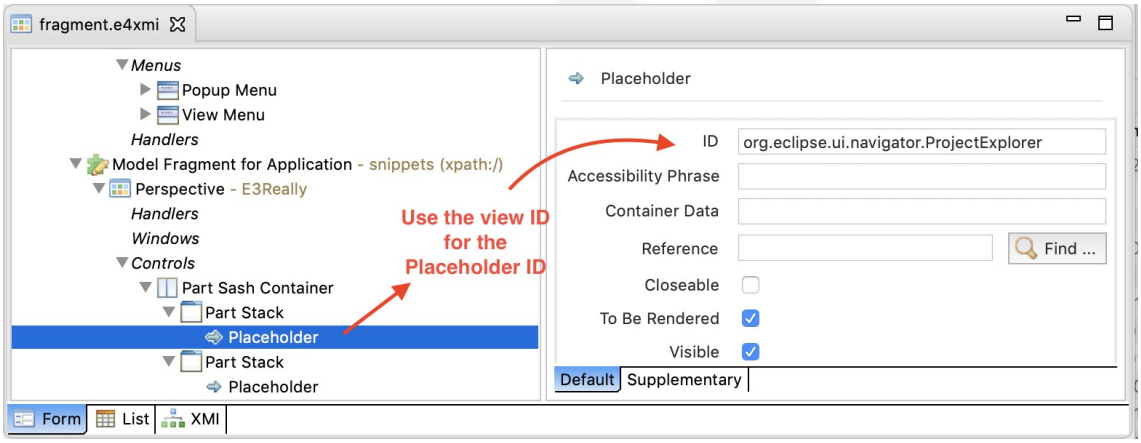
Defining your perspective with E3 and E4 views

- Define a Perspective **snippet** in the model fragment using **xpath:/** and **snippets** feature
- Describe the perspective using PartSashContainer, PartStack and PlaceHolder
- **For PlaceHolder set their IDs with the ID of the view (E3 or E4)**
- To display the perspective :
  - in the IDE : use the perspective switcher
  - in the code use the **EPartService.switchPerspective**



Perspective Snippet

Set the view ID in the PlaceHolder ID:



PlaceHolder init

Demo 4 : mix E3 and E4 views in an E4 perspective

Demo :

- create the perspective in a snippet and mix E3 and E4 views using PlaceHolders
- check with the model spy that placeHolder is bound, at runtime, to the shared element

Managing selection in both E3 and E4 modes

In a pure E4 application the **ESelectionService** manages directly the selected Object

- inject the IServiceConstants.ACTIVE_SELECTION as the expected type or as an Object[]
- there is no more received **ISelection**

In a E3 application, we have only **ISelection**

How to deal with both modes ?

Using the IServiceConstants.ACTIVE_SELECTION

If an E4 view must deal with both E3 and E4 selection we must have 3 methods :

- **@Inject void selE3(@Named(IServiceConstants.ACTIVE_SELECTION) ISelection s)**
  - to get the ISelection sent by E3 mechanism (compatibility mode)
- **@Inject void selE4(@Named(IServiceConstants.ACTIVE_SELECTION) TypedObject s)**
  - to get directly the TypedObject from E4 selection or called by selE3
- **@Inject void selE4(@Named(IServiceConstants.ACTIVE_SELECTION) Object[] s)**
  - to deal with multiple selection from E4 or called by selE3

Demo 5 : deal with E3 and E4 selections

Demo : looking in the code of the template.

Testing your E4 POJOs

E4 POJOs need injection and a proper initialized context

OPCoach has developped a specific E4 test case that initializes this context

The test case provides several convenient methods like:

- creating a Part using the context
- creating a Handler using the context
- setting the current selection
- checking if a text field in a POJO contains the right value
- checking if a tree is expanded with the correct nodes

Status of the project

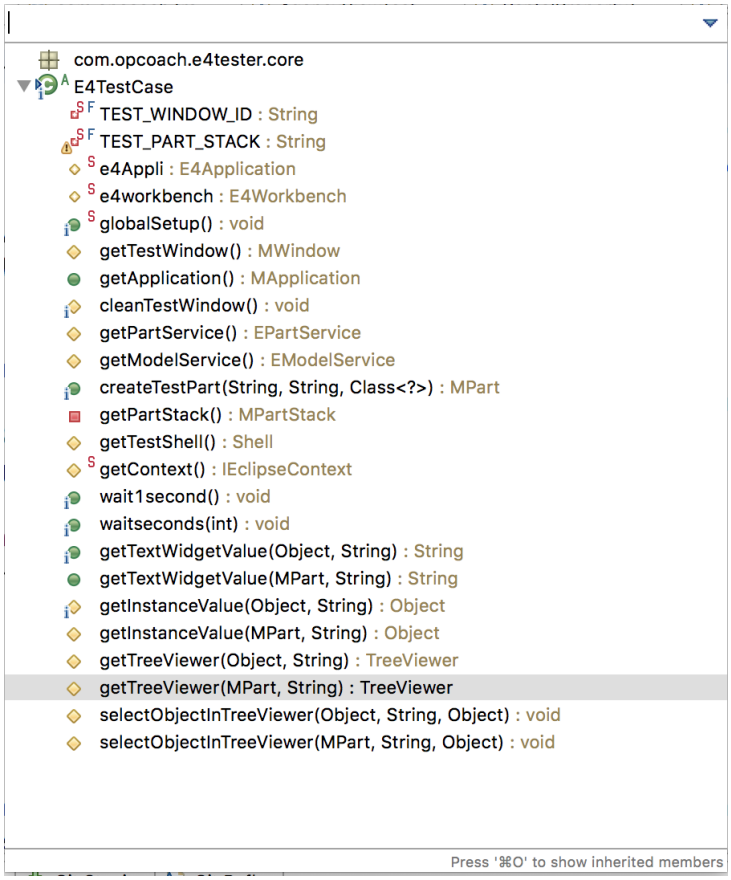
- This is still in progress but it works well
- It is available on:
  - **https://github.com/opcoach/E4Tester**
- It is delivered under EPL license
- There is an update site to set in your target platform:

```
19
20  // Add the E4 test platform from opcoach website
21  location "https://www.opcoach.com/repository/photon" {
22     com.opcoach.e4tester.feature.feature.group
23  }
24
```

The E4 TestCase

- The **E4TestCase** abstract class must be extended to define your test.

- It provides these basic methods:



Defining your E4 POJO Tests

Define your test in a plugin fragment

Then extend the E4TestCase, create your part(s) and check the POJO's fields

AgencyViewAndPropertyPartTest.java

```
21 public class AgencyViewAndPropertyPartTest extends E4TestCase {
22
23     MPart propertyPart, agencyPart;
24
25     @Before // See issue #3 (https://github.com/opcoach/E4Tester/issues/3), replace with
26             // BeforeEach later
27     public void setUp() throws Exception {
28
29         System.out.println("Testing AgencyViewAndPropertyPartTest... ");
30         // Create a part for the test...
31         try {
32             propertyPart = createTestPart("Rental Property", RentalPropertyPart.VIEW_ID, RentalPropertyPart.class);
33             agencyPart = createTestPart("Rental Agency", RentalAgencyPart.VIEW_ID, RentalAgencyPart.class);
34
35             // Must expand all nodes for selection !
36             TreeViewer tv = getTreeViewer(agencyPart.getObject(), "agencyViewer");
37             tv.expandAll();
38
39         } catch (Throwable ex) {
40             ex.printStackTrace();
41         }
42
```

```
65
66  @Test
67  // For junit5 when available in tycho : @DisplayName("Test if a rental is
68  // selected")
69  public void testRentalSelection() throws InterruptedException {
70
71      RentalAgency a = getContext().get(RentalAgency.class);
72
73      Rental rental = a.getRentals().get(1);
74      selectObjectInTreeViewer(agencyPart, "agencyViewer", rental);
75
76      wait1second();
77
78      assertEquals("Customer Name displayed is not correct", rental.getCustomer().getDisplayName(),
79              getTextWidgetValue(propertyPart, "customerNameLabel"));
80  }
81
```

E4 Tester talk

Thursday at 14:00

Conclusion

It is time to use E4 and to code your views,
handlers, contributions and perspectives in fragments !

Evaluate

Don't forget to evaluate this talk !

Any Questions ?

olivier@opcoach.com