# Table des matières

## I - Eclipse 4 Workshop

A. OPCoach................................................................................................................5

B. Workshop Eclipse 4 : Building an E4 application.........................................................6

C. Setting the tooling.........................................................................................................9

D. Step 1 : Creating application project.......................................................................12

E. Application Model.......................................................................................................17

F. Step 2 : Adding a dashboard view.............................................................................21

G. Step 3 Reminder on the injection.............................................................................26

H. Step 4 : Context Spy.................................................................................................30

I. Step 5 : Adding an engine to update values...............................................................31

J. Step 6 : Adding the engine control panel.................................................................33

K. Step 7 : E4 Event Bus...............................................................................................35

L. Creating an alarm system using event spy...............................................................38

M. step 9 : Add an alarm viewer..................................................................................41

N. Step 10 : application styling......................................................................................43

O. Conclusion..................................................................................................................47
Eclipse 4 Workshop

A. OPCoach

OPCoach

➢ About us
  ➢ Olivier Prouvost
    ➢ Eclipse expert trainer and committer (E4 Tools)
    ➢ olivier.prouvost@opcoach.com
    ➢ @OPCoach_Eclipse
  ➢ Emmanuel Benjamin
    ➢ Eclipse expert and trainer in Eclitech
    ➢ Responsible for training and consulting in Spain
    ➢ ebenjamin@eclitech.com

➢ About OPCoach
  ➢ Company founded in June 2009
  ➢ Member of the Eclipse Foundation (as Solution Member)
  ➢ Web site: http://www.opcoach.com
  ➢ Provides Eclipse training and consulting
  ➢ Recruitment activity to link companies and applicants
B. Workshop Eclipse 4: Building an E4 application

Introduction
This workshop will explain how to create a basic Eclipse 4 Application
It will show you:
- how to use the application model
- how to create a sample dashboard part
- how to manage injection
- how to deal with events
- how to debug and to trace information using spies
**Application sample**

We will create an application containing:

- an engine simulator (to provide sample data)
- an engine UI (to start/stop the engine and to accelerate/brake)
- a dashboard to display the speed and rpm value
- an alarm manager to trigger alarms depending on conditions

Basically the application will be like this:

![Main screen](image)
Architecture

➢ To make this application properly we will create 3 plugins
  ➢ `ecf14.eap` : the main plugin containing application model and dashboard
  ➢ `ecf14.engine.ui` : the UI for engine, containing the engine panel and alarm viewer
  ➢ `ecf14 engine.core` : the core for engine, containing simulator and alarm checker
➢ These plugins depend on the E4 Luna target platform

Resources

➢ The goal of the workshop is to create the application from scratch and to help you
➢ Feel free to ask questions or any help if you have a problem
➢ All the code of this workshop is available on github:
  ➢ [https://github.com/opcoach/Conferences/](https://github.com/opcoach/Conferences/)
➢ Import in workspaces only the `ecf14_files` project

The history is like this (you can get the project from any step you want)

1 - [https://github.com/opcoach/Conferences/tree/master/ECF14](https://github.com/opcoach/Conferences/tree/master/ECF14)
2 - [https://github.com/opcoach/Conferences/](https://github.com/opcoach/Conferences/)
You will may be remark that master has not been yet pushed on repository.

C. Setting the tooling

**Eclipse 4 IDE installation**

- Install the latest release of Eclipse Luna for RCP developers
- Unzip and launch it with a new workspace folder.
- You need to have the JRE 1.7 to launch Luna

**Eclipse 4 tooling installation**

- Install the latest release of E4 tools. To find it
- go to this URL http://download.eclipse.org/e4/downloads/ and select latest release
- or use directly this update site:
  
  http://download.eclipse.org/e4/downloads/drops/I20140615-2200/repository/
- or get it from the USB stick and import it as an archive update site
Select the following tools:
- core tools
- css spy
- event spy
- context spy

**Nebula installation**
- For this workshop, we will also use Nebula.
- Nebula is a library of cool graphical widgets (gauge, ...)
- Install it from:
  - the USB Stick (there is a zipped update site)
  - or from the nebula update site: http://download.eclipse.org/technology/nebula/snapshot
- Select all features
- Restart your Eclipse.

**Target platform setting**
The target platform is the set of plugins needed for the application
For this workshop we will use the running platform containing:
- Luna
- E4 tools
- Nebula.

**Getting the workshop files**
- To help you to make the workshop, some files are available
- You must import the git repository from:
  - the github web site: https://github.com/opcoach/Conferences.git
  - the USB stick
- To install it just copy the git url in the git repository view (access from git perspective)
Then import the ecf14_files project in your workspace using the import command.
D. Step 1 : Creating application project

New project

The easiest way to start a new E4 application is to use the wizard and to create a sample content.
Select a wizard

Wizards:

- Eclipse 4 Application Project
  - Classes
  - Model
  - Eclipse Modeling Framework
  - EMF Forms
  - Example EMF Model Creation Wizards
  - Git
  - Java
Name your project (eap is used for the top level plugin):

![New Plug-in Project dialog box](image_url)

- Project name: `com.opcoach.ecf14.eap`
- `Use default location` is checked
- Location: `/Users/olivier/Documents/OPCoach/workspaces`

**Project Settings**

- `Create a Java project` is checked
- Source folder: `src`
- Output folder: `bin`

**Target Platform**

This plug-in is targeted to run with:

- Eclipse version: 3.5 or greater
- an OSGi framework: `Equinox`

**Working sets**

- [Back] [Next] [Cancel] [Finish]
Skip the 2nd page (do not contribute to UI) and finally create a sample content:

Don’t forget to create a sample content
**You sample project is ready**

You can launch it directly by creating a new launch configuration:

**Launch config**

And you get the default application:
E. Application Model

*User Interface skeleton*

Any user interface contains always the same elements (views, commands ...
Eclipse 3 to Eclipse 4

Eclipse 3:
- All these components are defined using extensions defined in different plugins
- The main menus and the main perspective are defined by different classes (advisors)
Eclipse 4:

➢ defines an application model combining all these concepts:

The E4 application model

➢ It is a global model that brings together the usual extension points:
  ➢ View, perspective, menus (visual)
  ➢ Command handlers, key bindings (non-visual)
➢ It simply describes the structure of the UI without detailing its contents
➢ Its structure is defined by an Ecore meta model
➢ It can be edited with a dedicated editor
➢ It can be modified and UI is refreshed
➢ This model is independent from the display
  ➢ A ‘renderer’ allows specific display (swt and javafx)
➢ Classes referenced in the application model are simple annotated POJOs

The application model

The application model is stored in the e4 project
**URIs in the application model**

The application model references classes or resources (icons, properties ...)

**Class identification:**
- $\text{bundleclass://BundleID/package.classname}$
- $\text{bundleclass://com.opcoach.training.e4.rental.ui/com.opcoach.training.rental.e4.ui.views.AgencyView}$

**Resource identification:**
- $\text{platform:/plugin/BundleID/path/filename}$
- $\text{platform:/plugin/com.opcoach.training.e4.rental.ui/icons/Agency.png}$

**The visualization of the model at runtime**
- At any time you can see the actual application model
- Shortcut: **Alt Shift F9**
- You must add the plugin `org.eclipse.e4.tools.emf.liveeditor` in the launch configuration
- You can edit the model and see the changes
F. Step 2: Adding a dashboard view

Adding the dashboard view

For the moment, this dashboard will stand in eap plugin (but it could be moved later in engine.ui)

To do this:

- Copy the Dashboard from the step2 folder in the parts package of your plugin
- And manage the dependencies...
- Check the code to see how figures are created
- Replace the Sample Part in application model

Add the dependencies to nebula ...

- The dashboard part uses nebula, draw2d...
- With Ctrl Shift O, be careful to not import the classes for swt.GridLayout and swt.Label (it must come from draw2D)
➢ Add the required bundles for each compilation error:

Image 5 Add required bundles
## Dependencies

### Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

<table>
<thead>
<tr>
<th>Plugin Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.inject</td>
<td>1.0.0</td>
</tr>
<tr>
<td>org.eclipse.core.runtime</td>
<td>3.10.0</td>
</tr>
<tr>
<td>org.eclipse.swt</td>
<td>3.103.0</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.model.workbench</td>
<td>1.1.0</td>
</tr>
<tr>
<td>org.eclipse.jface</td>
<td>3.10.0</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.services</td>
<td>1.1.0</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.workbench</td>
<td>1.1.0</td>
</tr>
<tr>
<td>org.eclipse.e4.core.di</td>
<td>1.4.0</td>
</tr>
<tr>
<td>org.eclipse.e4.ui.di</td>
<td>1.0.0</td>
</tr>
<tr>
<td>org.eclipse.e4.core.contexts</td>
<td>1.3.100</td>
</tr>
<tr>
<td>org.eclipse.nebula.visualization.xygraph</td>
<td>1.0.0</td>
</tr>
<tr>
<td>org.eclipse.nebula.visualization.widgets</td>
<td>1.0.0</td>
</tr>
<tr>
<td>org.eclipse.draw2d</td>
<td>3.9.100</td>
</tr>
</tbody>
</table>

*dependencies*

**Update the application model**

Just switch the sample part to the new dashboard part.
Launch the application and check the live model

The application is launched using the same launcher:

If you have the default application instead of the dashboard, check the clear option in the launch configuration.
Launch

➢ You can check the dynamic model using the shortcut: Alt Shift F9
➢ The 'model spy' appears in a dedicated window: E4 spies
➢ You can change the model (name of part, visible, not visible...)

Access to other spies

Live update

E4 Spies
G. Step 3 Reminder on the injection

Introduction / Principle

➢ The goal of injection is to delegate the field or parameters initializations to a framework
➢ Injection uses a context containing the values
➢ We use the annotation @Inject (Javax.inject) to inject the values
➢ It can be applied to a constructor, a method or a field.
➢ The framework introspects classes to manage the artifacts that own an @Inject annotation

```java
package com.opcoach.training.e4.codesamples;

import javax.inject.Inject;
import org.eclipse.e4.core.di.annotations.Optional;

public class SampleInjectedClass {
    // An injected field
    private MyService service;

    @Inject
    public SampleInjectedClass(Object1 object, @Optional Object2 a2) {
        // An injected constructor with 2nd parameter optional
    }

    @Inject
    public void onInjectedMethod(Object2 a2) {
        // An injected method
    }
}
```

Image 6 Sample injected class

ContextInjectionFactory Class

A class with injection annotations:

➢ must be instantiated using the ContextInjectionFactory
➢ can not be instantiated with a call to new

ContextInjectionFactory instantiates objects and injects the context.
This is a static class
**The injection context**

- The context stores the values associated with keys or classes
- The context is hierarchical (application -> perspective -> parts...)
- Branches in context are activated by the framework (UI focus, part activation, ...)
- Branch activation gives an access to the available values in this branch
- If a value is not found, the context will ask its parent (getParent())
- The context is an implementation of IEclipseContext

**Additional annotations for @Inject**

3 additional annotations are used to manage object creation:

- **@PostConstruct**: is used to annotate a method that must be called at the end of initialization
- **@Optional**: is used to indicate that a field or a parameter could be null
- **@Named**: is used to get an object by its name

**Call order**

The injection is processed in the following order:

- **@Inject** Constructor call
- **@Inject** Fields initialization
- **@Inject** Methods call
- **@PostConstruct** Method call
Constructor call (1)

- `ContextInjectionFactory` will choose a constructor with an `@Inject` annotation
- If several constructors are annotated, it chooses the constructor having the maximum parameters that can be injected
- Fields marked with `@Inject` have a null value in the constructor.
- Other fields can be used normally
- Parent constructors are called by the `super()` (implicit) or by `super(...)` (explicit)

Fields Injection (2)

- Each field preceded by a `@Inject` annotation is initialized after the constructor call
- The order of initialization is unspecified
- If a field can not be injected the injector generates an exception
- If the value changes after the injection, it will be re-injected automatically
- If a field can have no value, it must be annotated with `@Optional`
- The inherited `@Inject` fields are initialized before the others
@ Inject methods invocation (3)

- Methods annotated with @Inject are called:
  - after the constructor call
  - after the injected fields initialization
  - if an injected parameter value has changed
- If several methods are annotated with @Inject, the call order is undefined
- The inherited @Inject methods are called before the others
- The inherited and overridden @Inject methods are not called

@ PostConstruct methods invocation (4)

- The method annotated with @PostConstruct is called:
  - after the calls to each @Inject methods
- The inherited @PostConstruct methods are called before the others
- The inherited and overridden @PostConstruct methods are not called
- The methods annotated with @PostConstruct are called once

Fundamental : Basic rule of injection

If a value previously injected into the injector changes after injection:

- it will automatically be re-injected into the relevant @Inject fields
- @Inject methods that have received this value as a parameter are automatically re-called.

Sample with @Named

```java
@Inject
public void setSelection(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Object o, Adapter adapter) {
    Rental r = adapter.adapt(o, Rental.class);
    setRental(r);
}
```

Sample with @Preference

The @Preference retrieves the value of a preference.
It can be used for a class field or for a parameter method.
The preference value is computed using ObjectContextSupplier.

```java
private final static String PLUGIN_ID = RentalUIActivator.PLUGIN_ID;
@Inject
public void refreshTree(@Preference(nodePath=PLUGIN_ID, value=CUSTOMER_KEY) String custCol,
    @Preference(nodePath=PLUGIN_ID, value=RENTAL_KEY) String rk,
    @Preference(nodePath=PLUGIN_ID, value=RENTALOBJECT_KEY) String rok)
{
    if (agencyViewer != null)
    {
        labelProvider.initPalette();
        agencyViewer.refresh();
    }
}
```

Image 9 @Inject @Named @Optional

Image 10 @Preference
H. Step 4 : Context Spy

Open the context spy

Use the shortcut Alt Shift F10 or the button in E4 spies window

What are the local values managed by this context?

- This part of the tree contains for the selected context, all the values directly set in this context.
- Parent context can not see these values
- Only the current context and child context can access them

What are the Inherited values injected or updated using this context?

- This part of the tree displays only values defined in parent context(s)
- These values are injected using the current selected context
- It is possible to open it and to check where injection is used (method or field)
- The values injected with @PostConstruct are never displayed (because called once)

Use the search to find object in context

- For instance search for "Composite"
- Check there is a parent Composite for the dashboard part.
- You can see that a same instance is stored in context using different keys corresponding to ancestor classes
I. Step 5: Adding an engine to update values

Create an engine

- We will create an engine updating values of
  - speed
  - rpm

- This engine will contain:
  - a timer task to change the values
  - a specific UI to start/stop or accelerate/brake

The core engine

- This is the simulator.
- For modularity reason, it will stand in a specific core plugin
- In this step we will not yet create the UI.

Making the plugin

- Create the `com.opcoach.ecf2014.engine.core` plugin
- This plugin will use injection so set these dependencies:

```
com.opcoach.ecf2014.engine.core
```

**Engine core dependencies**

- Create the `com.opcoach.ecf2014.engine.core` package
- Copy the `EngineSimulator.java` from `files/step5` folder
- Export the package
Check the source code

➢ The EngineSimulator defines the constants to get/set the values
➢ It receives the Context using Injection
➢ It contains a timer updating the values.

Integrating engine and dashboard

➢ Make the eap plugin depend on the engine.core
➢ Write the code to create the Engine :
  ➢ It must be created using ContextInjectionFactory and with the application context
  ➢ And Start engine

```java
@Inject
public Dashboard(MApplication appli)
{
    // Step 5 : create and start Engine.
    EngineSimulator simu = ContextInjectionFactory.make(EngineSimulator.class, appli.getContext());
    appli.getContext().set(EngineSimulator.class, simu);
    simu.start();
}
```

Create Engine

Then add the code to inject the values :

```java
@Inject @Optional
public void listenToValue(final ECF2014.RPM.VALUE int value, UISynchronize sync)
{
    if (rpmCounter != null)
    {
        sync.asyncExec(new Runnable()
        {
            @Override public void run()
            {
                rpmCounter.setValue(value);
            }
        });
    }
}
```

Inject the rpm value

Must check if widget exists

Use an UISynchronize

Listen to value

Do the same for the speed.

Launch the dashboard application

➢ The counters must change
➢ Check the context and find in the application context the speed and rpm values
J. Step 6 : Adding the engine control panel

Architecture

To separate UI and Core we will create a new UI pure E4 plugin.
To do it create an OSGi Bundle and then add the dependencies on:

- jface
- core.runtime
- draw2d
- nebula visualization widgets...

But don’t worry the required bundles will be proposed.
Dependencies

Add the EngineControlPanel

➢ Create a package and copy the EngineControlPart from step 6 folder
➢ Export this package
➢ Check the code:
  ➢ the engine simulator is optionally injected
  ➢ it will be updated when it will be added in context
  ➢ we must manage the case of ‘null’
  ➢ the Engine Simulator is updated in listeners and will update the context

Add the panel in the main window

In the eap project:
➢ Add the dependency to the new plugin
➢ Update the application model and add the part in a sash container
➢ Set the size ratios in the container data field (65% for top, 35% for engine control)
➢ Update the size of the window (600x575)

UI Organization
**Update the dashboard creation**

Now it's not necessary to start the engine when it is created:

- remove the start in Dashboard.

**Launch the application**

You should obtain this UI:

![Dashboard Image](image-link)

---

**K. Step 7: E4 Event Bus**

**Introduction**

- A good framework must provide an event management mechanism
- Usually, to be notified of an event, a listener must be defined
- And for each case a specific method must be defined
- Example: if you want to listen to what is going on with xxx, we would have:
  - xxxListener with xxxCreated (xxxEvent), xxxModified (xxxEvent) ...
  - or you can use the EMF adapters.
➢ In E4 there is a more simple mechanism: the IEventBroker.
➢ It manages the E4 event bus

**IEventBroker**

With the EventBroker you can:
- send an event identified with the relevant object
- receive event automatically when it appears (by injection)
- subscribe and be notified with an EventHandler (java.beans)

![Event Broker API](image11.png)

**Sending an event**

- Sending an event is simply a call to send or post
- The IEventBroker is received by injection

```java
@Inject IEventBroker broker;

@Execute
void execute() {
    // Create a new rental using factory
    Rental r = RentalFactory.getInstance().createRental();
    // broadcast the rental creation in broker
    broker.send("rental/new", r);
}
```

![Event Broker send](image12.png)

**Receipt and processing an event**

- We can do it in an injected method that receives a `@UIEventTopic` parameter
- The method being injected is called for each sent event
- String patterns can be used to receive events
The UIEvents class declares the framework events:

- **UILifeCyle** (BRINGTOTOP, ACTIVATE...)
- Application model updates (UIElement.VISIBLE...)

There is a dedicated spy to display events:

- You can open it using the shortcut: **Alt Shift F8**
- Or use the event spy button in E4 spy window
- The **org.eclipse.e4.tools.event.spy** plugin must be launched
L. Creating an alarm system using event spy

**Engine watcher**

The engine watcher will:

- get speed and rpm values
- compute an alarm event depending on conditions
- send an event with an alarm information depending on conditions

**Engine watcher implementation**

- We will create this class in the engine core simulator
- Copy it from the step 8 folder and add also the Alarm class
- It is a simple class that will be instantiated in the main dashboard
- It will define the event topics
- It will receive the values and the EventBroker by injection
- It sends instances of Alarm class in Event Broker
Engine Watcher

Alarm implementation

It is a simple bean with when, what and value:

```java
package com.opcoach.ecf2014.engine.core;

import java.text.SimpleDateFormat;
import java.util.Date;

public class Alarm
{
    private Date when; private String what; private int value;

    public Alarm(String what, int value)
    {
        this.what = what;
        this.value = value;
        when = new Date();
    }

    public Date getWhen() { return when; }
    public String getWhat() { return what; }
    public int getValue() { return value; }

    @Override public String toString()
    {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy.MM.dd HH:mm:ss");
        return "Alarm : " + sdf.format(when) + " + what : " + what + " + value : " + value;
    }
}
```
Engine watcher initialization.

The watcher is created in the constructor of the dashboard:

```java
@Inject
public Dashboard(MApplication appli) {
    // We will use the application context to store and inject values.
    EclipseContext appliContext = appli.getContext();

    // Step 5: create and start Engine.
    EngineSimulator simu = ContextInjectionFactory.make(EngineSimulator.class, appliContext);
    appliContext.set(EngineSimulator.class, simu);

    // Step 6: create the engine alarm watcher and keep a reference on it!
    EngineWatcher watcher = ContextInjectionFactory.make(EngineWatcher.class, appliContext);
    appliContext.set(EngineWatcher.class, watcher);
}
```

Create the engine watcher

Launch the application and check the event spy

- Relaunch the application and open the event spy.
- Set the filter to Alarm/* in event spy.
- Start the engine and accelerate.
- After a moment alarms should appear in event spy.
M. step 9 : Add an alarm viewer

The alarm part will receive event using injection

It is a table viewer displaying alarms:

<table>
<thead>
<tr>
<th>Alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
</tr>
<tr>
<td>15/06/2014</td>
</tr>
<tr>
<td>15/06/2014</td>
</tr>
<tr>
<td>15/06/2014</td>
</tr>
<tr>
<td>15/06/2014</td>
</tr>
</tbody>
</table>

Alarm Part

Implementation

➢ The Alarm Part is a standard E4 part
➢ It stands in the engine.ui plugin
➢ It will receive the event by injection:

```java
@Inject @Optional
public void listenToAlarms(UIEventTopic(EngineWatcher.ALARM_TOPIC) Alarm a) {
    alarms.addElementAt(a, 0);
    if (viewer != null)
        { 
            viewer.refresh();
            viewer.setSelection(new StructuredSelection(a));
        }
}
```

Receiving events

Application model update

➢ Of course the part must be added in application model.
➢ Set the proportions between sash containers
Windows and Dialogs
  ▼ Trimmed Window – com.opcoach.ecf14.esp
    ▼ Main Menu

Handlers
Windows and Dialogs
▼ Controls
  ▼ Perspective Stack
  ▼ Perspective
  Windows and Dialogs
  ▼ Controls
    ▼ PartSashContainer
      ▼ PartSashContainer
      ▼ Part Stack
        ▼ Part – Dashboard
    ▼ PartSashContainer
      ▼ Part Stack
        ▼ Part – Engine Control
    ▼ Part Stack
      ▼ Part – Alarms

Shared Elements

Application model update
Final result

Finally you can launch it and you get your application:

![Main screen](image)

**N. Step 10 : application styling**

*Introduction*

E4 has its own CSS rendering engine
A css file can be set in the properties of the product extension (see eap plugin.xml)
Simply create a css referencing:
➢ the name of a SWT widget
➢ the name of the specific objects
➢ the name of class in application model
➢ or your own css class defined using the IStylingEngine

**Css Sample**

```css
Button {
    color: #FF00FF;
    font-family: "Lucida Grande";
    font-size: 15;
    font-style: bold;
}

CTabFolder#com-opcoach-ecf14-eap-partstack-1 {
    /* background-color: #FF00 #202020 100%; */
    font-family: "Lucida Grande";
    font-size: 15;
}

.MTrimmedWindow.topLevel {
    margin-top:15px;
    margin-bottom:15px;
    background-color: #08F #000 100%
}
```

**CSS Spy**

➢ We can explore the effects of the CSS with CSS Spy
➢ To open it use the quick search at the top of the screen or Ctrl Shift F5 (win32) or Alt Shift F5 (Mac)
➢ The item in question is visualized on the screen
➢ You can get the CSS fragment and copy it in your css file
Image 16: CSS spy

Get the CSS fragment and put it in your css file.
CSS Result

With the previousCss you get this result:
Workshop

➢ Just copy the default.css file from the step10 folder into the css folder of the eap plugin
➢ And relaunch your application.
➢ You can play with the CSS values.

O. Conclusion

E4 status

➢ E4 framework is a great framework to make your applications
➢ The spies are very helpful to understand what is happening
➢ The E4 tools will be improved in the coming months (we are working on it)
➢ Feel free to ask questions to:
  ➢ olivier@opcoach.com
  ➢ emmanuel.benjamin@eclitech.com
➢ Training on these technologies are available in french, english and spanish.
  ➢ Just ask at the OPCoach booth.

Any Questions?