# Table of contents

I - Migration ......................................................... 5

A. Migration to E4.................................................. 5

B. Migration tooling.............................................. 8

C. Model Fragments and Processors...................... 12

D. Extension Migration......................................... 15

E. Resources....................................................... 20

Eclipse Con Reston 2016, Migration to E4
This part of the talk will explain:
- some general issues about migration
- tooling that could be used
- the model fragments and processors
- how to migrate some standard extensions
- Demo

**A. Migration to E4**

**The technical reasons for using E4 application platform**

- Application model is dynamic and platform agnostic (SWT, Java FX...) thanks to POJOs
- Injection is pretty cool, reduces the amount of code and simplifies testing
- Eclipse 4 event notification system (IEventBroker) is very concise and easy to use with injection
- You want to use the CSS styling capability and change element renderers of Eclipse 4
- You want to use the E4 spies to help to develop your application
- Your application will still live several years and it will provide an opportunity to refactor and decouple your components

**The global prerequisites**

Be sure of your team’s knowledge:
- do they know Eclipse 3 and Eclipse 4 ?
- do they know the application ! ! ?
- do they know how to migrate ?

**Create a migration strategy**

- Identify the features you want to migrate and the reasons why
- Be aware that you will may not be able to migrate the entire application !
Big picture of 3.X application with 4.X runtime

The 3.X / 4.X Eclipse runtime

Legacy Model (internal)
- Application
  - Add-ons
  - Binding Contexts
  - Binding Tables
  - Handlers
  - Commands
  - Command Categories
  - Windows and Dialogs
  - Part Descriptors
  - Menu Contributions

Workbench (Compatibility, provides 3.x APIs)

3.X Side
- Application
  - Add-ons
  - Binding Contexts
  - Binding Tables
  - Handlers
  - Commands
  - Command Categories
  - Windows and Dialogs
  - Part Descriptors
  - Menu Contributions

Runtime Model (Live)

4.X Side
- Your E4 UI Contributions using model fragments or processors

Eclipse Con Reston 2016, Migration to E4
Big picture of what we should do

The technical prerequisites
To prepare your E3 plugin/application migration you have to:
➢ remove the org.eclipse.ui internal package uses and imports
➢ ensure the application can be launched using the compatibility layer
  ➢ org.eclipse.equinox.ds
  ➢ org.eclipse.equinox.event
  ➢ org.eclipse.equinox.util
  ➢ org.eclipse.e4.ui.workbench.addons.swt
➢ clearly separate core and ui plug-ins
➢ have packages for each entities to migrate: views, handlers, etc...

Migration steps
To migrate a core plugin [without dependency to org.eclipse.ui], you must:
➢ do nothing!
To migrate a UI plug-in, you must:
➢ move the ui E3 extensions to a model fragment (or to the application model)
➢ migrate the relevant code
➢ remove all E3 extensions
➢ remove the org.eclipse.ui dependency when it is not used anymore
➢ add the jface dependency and others instead

Then, once all the plug-ins have been migrated, it is possible to remove the compatibility layer.
Practical advices

➢ Create a xxx.e4.xxx package to receive the migrated class, in the current migrated plug-in
➢ for instance : xxx.e4.handlers or xxx.e4.parts
➢ Copy the E3 class and its dependencies in this package and keep the names
➢ Set the E3 classes as ‘deprecated’
➢ Annotate with a //E34 comment the current migrated areas when they are not finished
➢ Remove the old E3 packages when the migration is finished

These tips help maintain existing plugins and the build process

Displaying the //E34 tasks

It is possible to display the //E34 comments in the task view :

➢ open the 'Tasks' view
➢ add a E34 tag in the preference page of Java->Compiler->Task

E34 tasks

B. Migration tooling

E4 Spies

➢ The E4 spies are useful to develop an E4 application
➢ They help in browsing the application model, injection contexts, events, css....
➢ It is possible to write its own spy for any specific data
➢ Eclipse Mars does not include the E4 spies
➢ They will be soon delivered by default
➢ To install them, upload the update site from :
   ➢ http://download.eclipse.org/e4/downloads

Download the zipped update site and install it :

➢ Menu Help -> Install New Software..
➢ 'Add..', 'Archive..'
Then select ‘All Spies’:

Image 1 E4 tooling

**Using the spies**

There are 3 different ways to open the spy window:

- use one of the shortcut (Alt Shift F4 to Alt Shift F10 for instance) depending on the installed spies
- look for “spy” in the quick access field
- use the Window->Spies menu (only in Neon):

![Image showing the Window menu with Spies option highlighted](image)

This menu is available only from the Neon version (4.6) (see bug #482250)

It will open a specific E4 Spies Window with a toolbar to display each spy.
For instance the **Model Spy**:

![E4 Spies Window](image)

### e4 Spies Window

**A tooling to help to evaluate the migration cost**

- OPCoach developed a specific statistic view dedicated to migration
- This plugin is available on github: [http://opcoach.github.io/E34MigrationTooling/](http://opcoach.github.io/E34MigrationTooling/)
- It is delivered under EPL license and it is free
Select the projects in the workspace and get some statistics about used UI extension points:

- Migration Stat View
  - An evaluation form to check your migration
  - OPCoach provides a form to help you to evaluate the work

- The plugin templates with model fragments
  - From Neon M4, it is now possible to create pure E4 plugins using model fragments.
    - a plugin with a pure SWT E4 view
C. Model Fragments and Processors

Introduction
You can contribute to an application model by using two mechanisms:

➢ a **model fragment**: with the ID or xpath of model objects
➢ a **processor**: with a piece of code modifying the injected application

Model fragment

➢ The model fragment adds content to an existing application model
➢ To create a model fragment,
  ➢ use the model fragment wizard (Ctrl N + fragment)
  ➢ extract a piece of model into a fragment (contextual menu on application model editor)

Application fragment

It is possible to add any contribution to any object
➢ just select the ID of the object
➢ then select the feature to be populated
then add a content

If you contribute on the top level application, you can use:
➢ the ID of the application
➢ the ID of the legacy E4 application : org.eclipse.e4.legacy.ide.application
➢ the 'xpath:/' to get any application whatever its ID (see bug #437958)
➢ This is the best practice for the top level contributions

Image 2 Addon in fragment

**Model fragment**

Don't forget to declare the fragment in an extension (org.eclipse.e4.workbench.model)

Image 3 Model Fragment

**Processor declaration**

➢ The processor is used when the object's ID is not known (application for instance)
➢ The application is received using injection so as to be modified
It must be declared in the `org.eclipse.e4.workbench.model` extension using a processor parameter:

Extension for a processor

**Processor code**

- The processor code is a POJO with a `@Execute` annotation
- The method receives the application and needed services as fields or parameters
- Use the `modelService.createElement` method to create instances

```java
/** A sample E4 processor adding a command in application */
public class SampleE4Processor {
    public void process(MApplication application, MModelService modelService) {
        // Just create a command and add it in the application
        MCommand command = modelService.createElement(MCommand.class);
        command.setElementID("id.of.my.command");
        command.setCommandName("Launch My Command");
        String contributorURI = "platform:plugin/" + FrameworkUtil.getBundle(getClass()).getSymbolicName();
        command.setContributorURI(command, contributorURI);
        command.setDescription("A sample command added in application");
        application.getCommands().add(command);
    }
}
```

Code for a processor

**Demo**

Will show:
- spies (model, context, css)
- migration tool
D. Extension Migration

Content

➢ This part will give some advices to migrate the main `org.eclipse.ui` extensions
➢ To find how to migrate an element, you can launch your application using the model spy and check what the compatibility layer has generated in the model.

View migration

An `org.eclipse.ui.views` extension is actually a `PartDescriptor` in the application model

To migrate a view:

➢ Copy your ViewPart code in the xxx.e4.parts package
➢ Transform the code into a POJO:
  ➢ remove inheritance to ViewPart
  ➢ add `@PostConstruct` before the `createPartControl` method
  ➢ add `@Focus` before the `setFocus` method
  ➢ update the code to manage the selection using injection
  ➢ remove the extension and the E3 code
➢ Bind this pojo in a model fragment:

To make the view appear in the 'Window -> Show view' menu:

➢ add this tags in the supplementary tab

```
Bind the contribution to:
ID = xpath:/
feature = descriptors
```

```
Find ...
```

```
Find ...
```

```
Add Remove
```

```
Default Supplementary
```

```
Bind to the Pojo
```
**Command Migration**

An `org.eclipse.ui.command` extension can be defined in the 'commands' feature of the application model:

- keep the same ID
- add the command in the fragment:

  ```xml
  <fragment.e4xml>
  <Model Fragment Definition>
  <Imports>
  <Model Fragments>
  <Model Fragment - commands (xpath:)/>
  <Command - Hello World/>
  </Model Fragment - commands (xpath:)/>
  </Model Fragment Definition>
  </fragment.e4xml>
  ```

**Handler Migration**

To migrate an `org.eclipse.ui.handlers` extension:

- Copy the E3 handler code in the xxx.e4.handlers package
- Transform the code into a POJO:
  - remove inheritance to AbstractHandler
  - add `@Execute` before the `execute` method
  - add `@CanExecute` annotated method if needed
  - receive needed values as parameters (will be injected)
- Bind this pojo in a model fragment (`xpath:` and `handlers`)

**MenuContribution Migration**

An `org.eclipse.ui.menus` extension must be redefined in the model fragment:

- use `xpath:` and 'menuContributions' feature
- The link is done using the parent ID
Menu Contribution

MenuContribution / Parameters

The following parent ID can be used:

- ID of an existing view (it must have been registered using the EMenuService)
- ID of an existing menu
- org.eclipse.ui.main.menu : used for the main menu
- popup : used to be located in any part
- org.eclipse.ui.main.toolbar : used to be located in the main toolbar.

For the position:

- an ID of any existing object (command, menu, etc...)
- after= additions : the default location

It is possible to open the model Spy so as to check the values used by the IDE

Wizard migration

- org.eclipse.ui.[??]Wizards
- Wizards are not defined in the application model
- There is also no extension point outside of org.eclipse.ui
- Therefore, the main dialog to choose a wizard is not available in a pure E4 application
- Nevertheless it is possible to open a specific wizard in a pure E4 code
- Wizards are only JFace code and can be adapted to deal with injected selection
They must not implement `INewWizard`, `IImportWizard` or `IExportWizard` anymore.

A command must be created to open the wizard, using the `WizardDialog` of JFace.

**Sample wizard**

```java
package com.opcoach.training.e4.codesamples;
import javax.inject.Inject;
import org.eclipse.e4.core.contexts.ContextInjectionFactory;
import org.eclipse.e4.core.contexts.IEclipseContext;
import org.eclipse.jface.wizard.Wizard;

public class SampleWizard extends Wizard {
    private SampleWizardPage firstPage = null;
    private IEclipseContext context;

    @Inject
    public SampleWizard(IEclipseContext ctx) {
        setWindowTitle("New Wizard");
        context = ctx;
    }

    @Override
    public void addPages() {
        firstPage = ContextInjectionFactory.make(SampleWizardPage.class, context);
        addPage(firstPage);
    }

    @Override
    public boolean performFinish() {
        // Do your stuff here by asking the pages...
        return true;
    }
}
```
Sample wizard page

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

import java.io.File;

public class SampleWizardPage extends WizardPage
{
    private Object selection;
    private Label filename;

    @Inject
    public SampleWizardPage(@Named(IServiceConstants.ACTIVE_SELECTION) Object currentSelection)
    {
        super("wizardPage");
        setTitle("Wizard Page title");
        setDescription("Wizard Page description");
        selection = currentSelection;
    }

    @Override
    public void createControl(Composite parent)
    {
        Composite container = new Composite(parent, SWT.MULI);
        filename = new Label(container, SWT.BORDER);
        if (selection instanceof File)
        {
            filename.setText(((File)selection).getName());
        }
        setControl(container);
    }

    @Override
    public boolean isPageComplete()
    {
        return filename.getText().length() > 0;
    }
}
```

Inject current selection for content init

Open wizard

```java
public class OpenSampleWizard {
    @Execute
    public void execute(IWizardContext ctx, Shell s)
    {
        Wizard w = ContextInjectionFactory.make(MyWizard.class, ctx);
        WizardDialog wd = new WizardDialog(s, w);
        wd.open();
    }
}
```

Preference pages Migration

- Like wizards, preference pages are not defined in the application model
- It is possible to use the plugin: https://github.com/opcoach/e4Preferences

2 - https://github.com/opcoach/e4Preferences

Eclipse Con Reston 2016, Migration to E4
➢ You need to:
➢ ensure that your preference pages are extending `FieldEditorPreferencePage`
➢ change the extension `org.eclipse.ui.preferencePages` to `com.opcoach.e4.preferences.e4PreferencePages`
➢ add the handler and the command in your model

For the default values, you can keep the `org.eclipse.core.runtime.preferences` extensions.

**Other migrations**

➢ There are still plenty tips for your migration
➢ Try to put it in your model fragment
➢ If you can not describe your contribution in a model fragment, use a model processor

### E. Resources

**Articles about migration**

➢ Eclipse magazine about migration (german):
  ➢ [https://jaxenter.de/ausgaben/eclipse-magazin-6-15](https://jaxenter.de/ausgaben/eclipse-magazin-6-15)
➢ Recipes for your Eclipse 4 migration (english):
➢ OPCoach’s article in eclipse magazine (german):
➢ Comment migrer vers eclipse 4 (french):

**Ask your questions**

Feel free to ask your questions
➢ in E4 forum
➢ by email:
  ➢ olivier@opcoach.com
  ➢ bsd@ml.ca
➢ Now after this talk or during the conference!