e(fx)clipse APIs

BestSolution

Tom Schindl <tom.schindl@bestsolution.at>

Twitter: @tomsontom
Blog: http://tomsondev.bestsolution.at
Website: http://www.bestsolution.at

Matthew Elliot <matthew.elliot@360t.com>

Website: http://www.360t.com

About Tom

- CTO BestSolution.at Systemhaus GmbH
- Eclipse Committer
  - e4
  - Platform
  - EMF
- Project lead
  - e(fx)clipse
- Twitter: @tomsontom
- Blog: tomsondev.bestsolution.at
- Corporate: http://bestsolution.at

About Matt

BestSolution

- Software Team Lead @ 360T (Deutsche Börse Group)
- Focus on OTC Trading Applications
- Product Owner for IAF @ 360T
  - Eclipse RCP 4
  - JavaFX
  - e(fx)clipse
- E-Mail: matthew.elliot@360t.com
- Company: http://www.360t.com
- LinkedIn: https://de.linkedin.com/pub/matthew-elliot/24/43a/61

BestSolution

RCP / JavaFX / e(fx)clipse for Trading applications

Common / Difficult Requirement Set

Common / Difficult Requirement Set

BestSolution

- Requirements

Common / Difficult Requirement Set

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers
  - Multiple language support

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers
  - Multiple language support
  - Support for integration with 3rd parties at a visual level

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers
  - Multiple language support
  - Support for integration with 3rd parties at a visual level
  - Next 5, 10, 15 years, support, evolution, flexibility

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers
  - Multiple language support
  - Support for integration with 3rd parties at a visual level
  - Next 5, 10, 15 years, support, evolution, flexibility
- Two main options

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers
  - Multiple language support
  - Support for integration with 3rd parties at a visual level
  - Next 5, 10, 15 years, support, evolution, flexibility
- Two main options
  - JavaFX

Common / Difficult Requirement Set

BestSolution

- Requirements
  - Plugin architecture to support growing number of trading, reporting, and pricing tools
  - Flexibility in managing screen space, multiple monitor set ups, and ability for user to customise their layout and tool set
  - More flexibility in design and customisation for our customers
  - Multiple language support
  - Support for integration with 3rd parties at a visual level
  - Next 5, 10, 15 years, support, evolution, flexibility
- Two main options
  - JavaFX
  - HTML 5

UX Concept for Visual Integration Framework

UX Concept for Visual Integration Framework

- Core components

UX Concept for Visual Integration Framework

- Core components
  - Tabbed views with flexible layouts; built up like Tetris with different plugins acting as ‘building blocks’

UX Concept for Visual Integration Framework

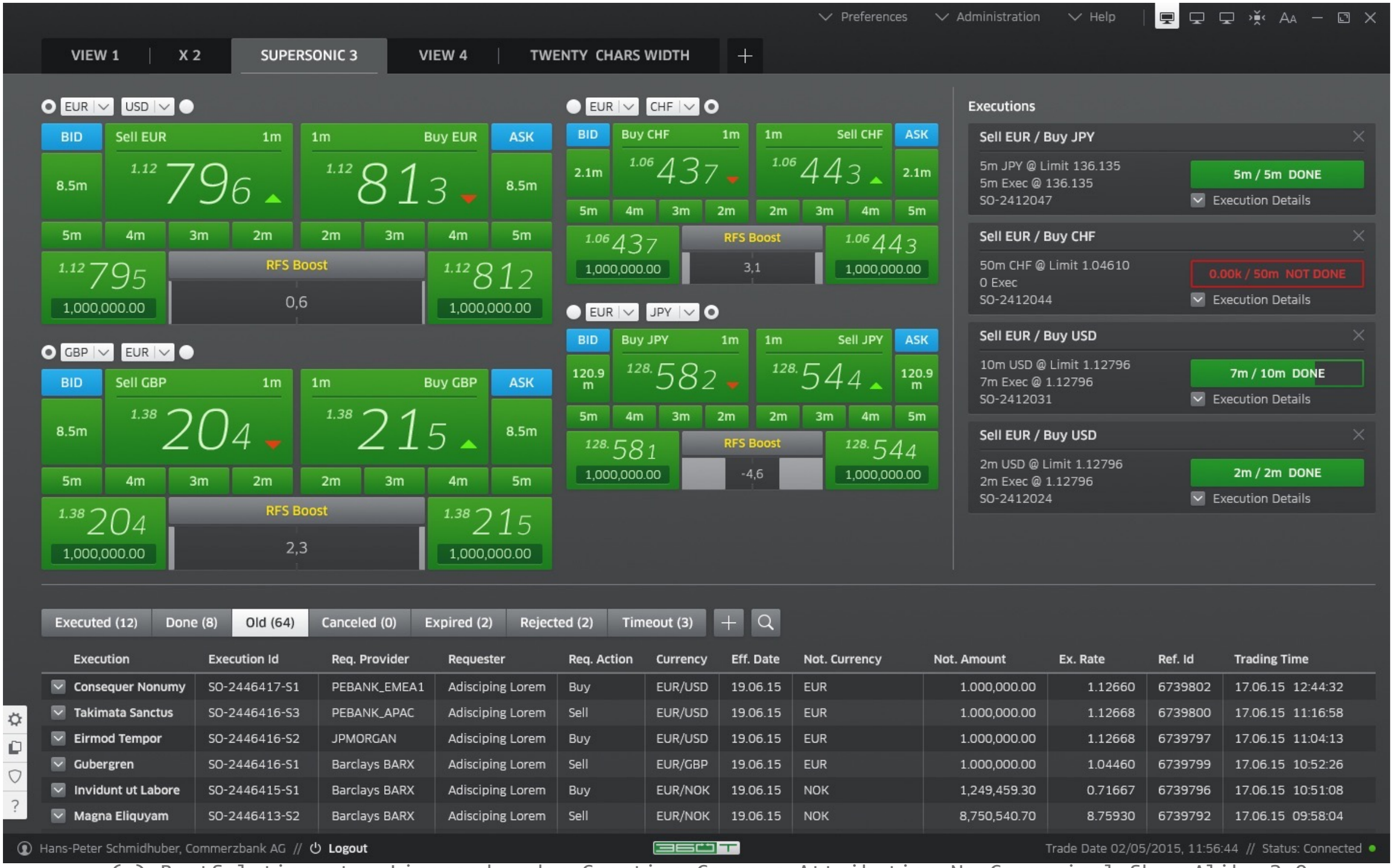
- Core components
  - Tabbed views with flexible layouts; built up like Tetris with different plugins acting as ‘building blocks’
  - Advanced controls for managing trading activity across multiple monitors

UX Concept for Visual Integration Framework

- Core components
  - Tabbed views with flexible layouts; built up like Tetris with different plugins acting as ‘building blocks’
  - Advanced controls for managing trading activity across multiple monitors
- Modern / flat design line / 360T of the future

UX Concept for Visual Integration Framework

BestSolution



‚The Leap‘ - A Technology Stack to Support an Enterprise

‚The Leap‘ - A Technology Stack to Support an Enterprise

- 2012-14 – Stalled, JavaFX?, HTML5?, Java8?, AppStore?

‚The Leap‘ - A Technology Stack to Support an Enterprise

- 2012-14 – Stalled, JavaFX?, HTML5?, Java8?, AppStore?
- 2014 – Preliminary Investigations into EclipseRCP with JavaFX

‚The Leap‘ - A Technology Stack to Support an Enterprise

- 2012-14 – Stalled, JavaFX?, HTML5?, Java8?, AppStore?
- 2014 – Preliminary Investigations into EclipseRCP with JavaFX
- 2015 (Q1) – Prototype Phase with JavaFX / Eclipse RCP (can it support our requirements?) / Best Solution engaged for experience

‚The Leap‘ - A Technology Stack to Support an Enterprise

- 2012-14 – Stalled, JavaFX?, HTML5?, Java8?, AppStore?
- 2014 – Preliminary Investigations into EclipseRCP with JavaFX
- 2015 (Q1) – Prototype Phase with JavaFX / Eclipse RCP (can it support our requirements?) / Best Solution engaged for experience
- 2015 (Q3) – Successful development of first concrete application

‚The Leap‘ - A Technology Stack to Support an Enterprise

- 2012-14 – Stalled, JavaFX?, HTML5?, Java8?, AppStore?
- 2014 – Preliminary Investigations into EclipseRCP with JavaFX
- 2015 (Q1) – Prototype Phase with JavaFX / Eclipse RCP (can it support our requirements?) / Best Solution engaged for experience
- 2015 (Q3) – Successful development of first concrete application
- 2016 – Well I can’t give away our strategic roadmap…

BestSolution as ‚Sparring Partner‘

BestSolution as ‚Sparring Partner‘

- Trading applications can’t look like my IDE… what else is possible?

BestSolution as ‚Sparring Partner‘

- Trading applications can’t look like my IDE… what else is possible?
- How hard is it to customize Eclipse RCP…?

BestSolution as ‚Sparring Partner‘

- Trading applications can’t look like my IDE… what else is possible?
- How hard is it to customize Eclipse RCP…?
- What about ‘insert grand UX design’…?

BestSolution as ‚Sparring Partner‘

- Trading applications can’t look like my IDE… what else is possible?
- How hard is it to customize Eclipse RCP…?
- What about ‘insert grand UX design’…?
- What about performance…?

BestSolution as ‚Sparring Partner‘

- Trading applications can’t look like my IDE… what else is possible?
- How hard is it to customize Eclipse RCP…?
- What about ‘insert grand UX design’…?
- What about performance…?
- What about OSGi? / what about Dependency Injection?

BestSolution

Demo

Eclipse RCP <-> 360T Mappings

360T UX Concept	Eclipse RCP
Screen	TrimmedWindow
Tab Container / Tab View	PartStack
Feature	Part
Menu	Main Menu

BestSolution

Boring code

(e4)App-Architecture

- Services
  - Everything is designed as a service in e4
  - Everything in your app SHOULD be designed as service
- Dependencies
  - NEVER create a (OSGi/e4) framework API dependency in your business code you can NOT mock easily or reimplement in other ENVs - watch out for transitive dependencies!

Some Helpers

Useful Base-APIs

- Lookup Services (if you are outside the DI-Container)

Useful Base-APIs

BestSolution

- Lookup Services (if you are outside the DI-Container)

```
Bundle bundle = FrameworkUtil.getBundle(ServiceSample.class);
BundleContext btx = bundle.getBundleContext();

if( btx == null ) {
  return;
}

ServiceReference<FilesystemService> ref = btx.getServiceReference(FilesystemService.class);
if( ref == null ) {
  return;
}

FilesystemService service = btx.getService(ref);
service.observePath(Paths.get("/tmp"), (k,p) -> {
  // Handle Path change
});
```

Useful Base-APIs

- Lookup Services (if you are outside the DI-Container)

```
Util.getService(ServiceSample.class, FilesystemService.class).ifPresent( service -> {
  service.observePath(Paths.get("/tmp"), (k,p) -> {
    // Handle Path change
  });
});
```

Useful Base-APIs

- Work with APIs with exceptions

Useful Base-APIs

BestSolution

- Work with APIs with exceptions

```
class EJDTSourceFileInput {
  private ICompilationUnit compilationUnit;

  // ...
  protected void doDispose( ) {
    if( compilationUnit != null ) {
      try {
        compilationUnit.restore( );
      } catch (JavaModelException e) {
        // Log the exception
      }

      try {
        compilationUnit.discardWorkingCopy( );
      } catch (JavaModelException e) {
        // Log the exception
      }
    }
    // ...
  }
}
```

Useful Base-APIs

- Work with APIs with exceptions

Useful Base-APIs

- Work with APIs with exceptions

```
public interface ExRunnable {
  public void wrappedRun() throws Throwable;
}

public final class ExExecutor {
  // …
  // many other wrappers for Supplier, Function
  public static void executeRunnable(ExRunnable r) {
    // super secrete code
  }
}
```

Useful Base-APIs

BestSolution

- Work with APIs with exceptions

```
public interface ExRunnable {
  public void wrappedRun() throws Throwable;
}

public final class ExExecutor {
  // …
  // many other wrappers for Supplier, Function
  public static void executeRunnable(ExRunnable r) {
    // super secrete code
  }
}
```

```
class EJDTSourceFileInput {
  private ICompilationUnit compilationUnit;

  // ...
  protected void doDispose() {
    ExExecutor.executeRunnable(compilationUnit::restore);
    ExExecutor.executeRunnable(compilationUnit::discardWorkingCopy);
    // ...
  }
}
```

Update

Update

- p2 API is powerful but soooo low-level

Update

BestSolution

- p2 API is powerful but soooo low-level

```
class UpdateHandler {

 public void update(UpdateService u) {
  u.checkUpdate(ProgressReporter.NULLPROGRESS_REPORTER).onComplete(this::checkComplete);
 }

 private void checkComplete(Optional<UpdatePlan> op) {
   op.ifPresent( p -> p.runUpdate( ProgressReporter.NULLPROGRESS_REPORTER ) );
 }
}
```

Command Execution

Command Execution

- e4 defines `ECommandService` and `EHandlerService` in `org.eclipse.e4.core.service`
  - exposes us to `org.eclipse.core.commands`
  - isn't as simple to use as it could be

Command Execution

- e4 defines `ECommandService` and `EHandlerService` in `org.eclipse.e4.core.service`
  - exposes us to `org.eclipse.core.commands`
    - isn’t as simple to use as it could be

```
import org.eclipse.core.commands.ParameterizedCommand;
import org.eclipse.e4.core.commands.ECommandService;
import org.eclipse.e4.core.commands.EHandlerService;

@Inject
public CommandSample(ECommandService c, EHandlerService h) {
 ParameterizedCommand cmd = c.createCommand("my.command.id", Collections.emptyMap() );
 String rv = h.executeHandler(cmd);
 if( rv != null ) {
   rv = "Default";
 }
}
```

Command Execution

- e4 defines `ECommandService` and `EHandlerService` in `org.eclipse.e4.core.service`
  - exposes us to `org.eclipse.core.commands`
  - i

Disapproved by Tom

```
import (
import org.eclipse.e4.core.commands.ECommandService;
import org.eclipse.e4.core.commands.EHandlerService;

@Inject
public CommandSample(ECommandService c, EHandlerService h) {
 ParameterizedCommand cmd = c.createCommand("my.command.id", Collections.emptyMap() );
 String rv = h.executeHandler(cmd);
 if( rv != null ) {
   rv = "Default";
 }
}
```

Command Execution

Command Execution

- fx.core defines a `CommandService`
  - who is simple to use
  - does not expose you to any complex APIs

Command Execution

- fx.core defines a `CommandService`
  - who is simple to use
  - does not expose you to any complex APIs

```
import org.eclipse.fx.core.command.CommandService;

@Inject
public CommandSample(CommandService c) {
 Optional<String> rv = c.execute("my.command.id", Collections.emptyMap());
 String v = rv.orElse("Default");
}
```

Logging

Logging the efxclipse way

- e4 Log API is not in acceptable state
- wanted our framework to not depend on a concrete logging API (not even SLF4J but in most minimal env we run on Java-Util-Logging)
- we wanted a modern log API who eg accepts lambdas, printf-formats, …
- framework users need to have an easy way to reconfigure and log EVERYTHING with their preferred framework

Logging

Logging


```
private final Logger logger;

   @Inject
   public MyComponent(@Log Logger logger) {
      this.logger = logger;
   }

   void method() {
      logger.debugf("Duke's Birthday: %1$tm %1$te,%1$tY", Calendar.getInstance());
      logger.debug(() -> "This data is very complex to calculate");
   }
```

Logging

BestSolution

```
private final Logger logger;

   @Inject
   public MyComponent(@Log Logger logger) {
      this.logger = logger;
   }

   void met
      logge
      logge
   }
```

```
@Component
public class MyComponent {
   private LoggerFactory factory;
   private Logger logger;

   @Reference
   void setLoggerFactory(LoggerFactory factory) {
      this.factory = factory;
      this.logger = factory.createLogger(MyComponent.class.getName());
   }

   void unsetLoggerFactory(LoggerFactory factory) {
      // clean up
   }
}
```

Events

The IEventBroker

The IEventBroker

- VERSIONED package level imports are not possible

The IEventBroker

- VERSIONED package level imports are not possible
- Service definition has only 4 methods
  - 2 for publishing: expose **`String`** and **`Object`**
  - 2 for subscribing: expose **`String`** and **`EventHandler`**

The IEventBroker

- VERSIONED package level imports are not possible
- Service definition has only 4 methods
  - 2 for publishing: expose **`String`** and **`Object`**
  - 2 for subscribing: expose **`String`** and **`EventHandler`**
- org.osgi.service.event.EventHandler exposes o.o.s.e.Event

The IEventBroker

Hierarchical view of plug-ins required by 'org.eclipse.e4.core.services (2.0.0.v20150403-1912)':

- org.eclipse.e4.core.services (2.0.0.v20150403-1912)
  - javax.annotation.jre (1.2.0.201510061341)
  - javax.inject (1.0.0.v20091030)
  - org.eclipse.core.jobs (3.7.0.v20150330-2103)
    - org.eclipse.equinox.common (3.7.0.v20150402-1709)
    - org.eclipse.osgi (3.10.101.v20150820-1432)
  - org.eclipse.e4.core.contexts (1.4.0.v20150828-0818)
    - javax.inject (1.0.0.v20091030)
    - org.eclipse.e4.core.di (1.5.0.v20150421-2214)
    - org.eclipse.osgi (3.10.101.v20150820-1432)
  - org.eclipse.e4.core.di (1.5.0.v20150421-2214)
    - javax.annotation.jre (1.2.0.201510061341)
    - javax.inject (1.0.0.v20091030)
    - org.eclipse.e4.core.di.annotations (1.4.0.v20150528-1451)
    - org.eclipse.osgi (3.10.101.v20150820-1432)
  - org.eclipse.equinox.common (3.7.0.v20150402-1709)
    - org.eclipse.osgi (3.10.101.v20150820-1432)
  - org.eclipse.equinox.preferences (3.5.300.v20150408-1437)
    - org.eclipse.equinox.common (3.7.0.v20150402-1709)
    - org.eclipse.equinox.registry (3.6.0.v20150318-1503)
    - org.eclipse.osgi (3.10.101.v20150820-1432)
    - org.eclipse.equinox.preferences (3.5.300.v20150408-1437)
  - org.eclipse.osgi (3.10.101.v20150820-1432)
    - org.eclipse.osgi.compatibility.state (1.0.100.v20150402-1551)
  - org.eclipse.osgi.services (3.5.0.v20150519-2006)
    - javax.servlet (3.1.0.v201410161800)
    - org.eclipse.osgi (3.10.101.v20150820-1432)
    - org.eclipse.osgi.services (3.5.0.v20150519-2006)

The IEventBroker

Disapproved by Tom

EventBus

EventBus

- VERSIONED package imports possible

EventBus

- VERSIONED package imports possible
- Simple interface who does not expose ANY OSGi-APIs
  - 2 publish methods
  - 1 subscribe methods

EventBus

- VERSIONED package imports possible
- Simple interface who does not expose ANY OSGi-APIs
  - 2 publish methods
  - 1 subscribe methods
- Enhanced type safety

EventBus

- VERSIONED package imports possible
- Simple interface who does not expose ANY OSGi-APIs
  - 2 publish methods
  - 1 subscribe methods
- Enhanced type safety
- Non OSGi-Version based upon Guava EventBus is available

EventBus: From e4 to efx

BestSolution

```
public class MyComponent {
   private static final String TOPIC = "my/topic";

   @Inject
   public MyComponent(IEventBroker eventBroker) {
      eventBroker.subscribe(TOPIC, this::handleEvent);
      eventBroker.send(TOPIC, "Hello World");
      eventBroker.unsubscribe(this::handleEvent);
   }

   private void handleEvent(Event event) {
      handle((Long) event.getProperty(IEventBroker.DATA));
   }

   private void handle(Long data) {
      System.err.println("The data is" + data);
   }
}
```

EventBus: From e4 to efx

BestSolution

```
public class MyComponent {
   private static final Topic<Long> TOPIC = new Topic<>("my/topic");

   @Inject
   public MyComponent(EventBus eventBroker) {
      eventBroker.subscribe(TOPIC, this::handleEvent);
      eventBroker.send(TOPIC, "Hello World");
      eventBroker.unsubscribe(this::handleEvent);
   }

   private void handleEvent(Event event) {
      handle(event.getData());
   }

   private void handle(Long data) {
      System.err.println("The data is" + data);
   }
}
```

EventBus: From e4 to efx

BestSolution

```
public class MyComponent {
   private static final Topic<Long> TOPIC = new Topic<>("my/topic");

   @Inject
   public MyComponent(EventBus eventBroker) {
      Subscription s = eventBroker.subscribe(TOPIC, this::handleEvent);
      eventBroker.publish(TOPIC, "Hello World",true);
      s.dispose();
   }

   private void handleEvent(Event<Long> event) {
      handle(event.getData());
   }

   private void handle(Long data) {
      System.err.println("The data is" + data);
   }
}
```

EventBus: From e4 to efx

BestSolution

```
public class MyComponent {
   private static final Topic<Long> TOPIC = new Topic<>("my/topic");

   @Inject
   public MyComponent(EventBus eventBroker) {
      Subscription s = eventBroker.subscribe(TOPIC, this::handleEvent);
      eventBroker.publish(TOPIC, "Hello World",true);
      s.dispose();
   }

   private void handleEvent(Event<Long> event) {
      handle(event.getData());
   }

   private void handle(Long data) {
      System.err.println("The data is" + data);
   }
}
```

Compiler Error

EventBus: From e4 to efx

BestSolution

```
public class MyComponent {
   private static final Topic<Long> TOPIC = new Topic<>("my/topic");

   @Inject
   public MyComponent(EventBus eventBroker) {
      Subscription s = eventBroker.subscribe(TOPIC, this::handleEvent);
      eventBroker.publish(TOPIC, 1l,true);
      s.dispose();
   }

   private void handleEvent(Event<Long> event) {
      handle(event.getData());
   }

   private void handle(Long data) {
      System.err.println("The data is" + data);
   }
}
```

EventBus: From e4 to efx

BestSolution

```
import static org.eclipse.fx.core.event.EventBus.*;
public class MyComponent {
   private static final Topic<Long> TOPIC = new Topic<>("my/topic");

   @Inject
   public MyComponent(EventBus eventBroker) {
      Subscription s = eventBroker.subscribe(TOPIC, data(this::handle));
      eventBroker.publish(TOPIC, 1l,true);
      s.dispose();
   }

   private void handleEvent(Event<Long> event) {
      handle(event.getData());
   }

   private void handle(Long data) {
      System.err.println("The data is" + data);
   }
}
```

IEclipseContext

The IEclipseContext

The IEclipseContext

- Retrieval is easy just use `@Inject`

The IEclipseContext

- Retrieval is easy just use `@Inject`
- Publishing is hard
  - The publisher needs to know where the value has to go. Is the perspective-, window- or the application-context the target?
- Your code gets a dependency on Eclipse 4 APIs

The IEclipseContext

- Retrieval is easy just use `@Inject`
- Publishing is hard
- T go.
  I text
  t
- Your code gets a dependency on Eclipse 4 APIs

Disapproved by Tom

The IEclipseContext

The IEclipseContext

- Stop thinking about the `IEclipseContext` as a `Map` of key-value-pairs

The IEclipseContext

- Stop thinking about the `IEclipseContext` as a `Map` of key-value-pairs
- Start treating each key-value-slot in `IEclipseContext` as an Observable-Value

The IEclipseContext

- Stop thinking about the `IEclipseContext` as a `Map` of key-value-pairs
- Start treating each key-value-slot in `IEclipseContext` as an Observable-Value

```
public interface ContextBoundValue<T> extends Adaptable {
  public T getValue();
  public void publish(T value);
}
```

The IEclipseContext

BestSolution

- Stop thinking about the `IEclipseContext` as a `Map` of key-value-pairs

- Start treating each key-value-slot in `IEclipseContext` as an Observable-Value

```
public interface ContextBoundValue<T> extends Adaptable {
  public T getValue();
  public void publish(T value);
}
```

```
public static class SimpleInject {
  @Inject
  @ContextValue("simpleValue")
  public ContextBoundValue<String> value;
}
```

The IEclipseContext

BestSolution

- Stop thinking about the `IEclipseContext` as a `Map` of key-value-pairs
- Start treating each key-value-slot in `IEclipseContext` as an Observable-Value

```
public interface ContextBoundValue<T> extends Adaptable {
  public T getValue();
  public void publish(T value);
}
```

```
public static class SimpleInject {
  @Inject
  @ContextValue("simpleValue")
  public ContextBoundValue<String> value;
}
```

```
public static class SimpleInject {
  @Inject
  @ContextValue("simpleValue")
  public IObservableValue<String> value;
}
```

BestSolution

Preference & UI-State

Preferences and UI-State

Preferences and UI-State

- **Preference-Store**:
  Used for none UI states

Preferences and UI-State

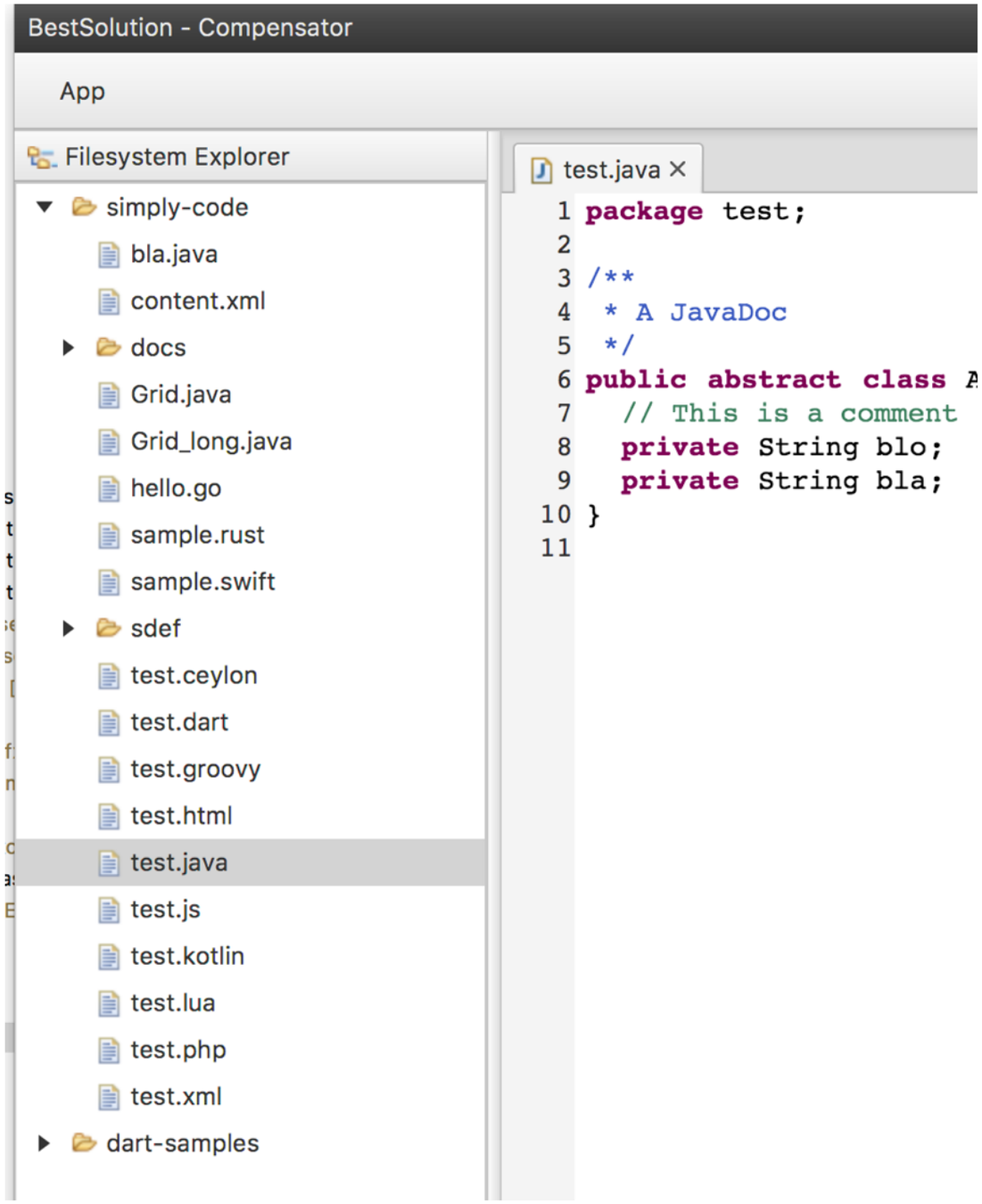
- **Preference-Store**:
  Used for none UI states

- **MApplicationElement #persistedState: Map<String,String>**:
  To be used for UI states

Preferences and UI-State

- **Preference-Store**: Used for none UI states

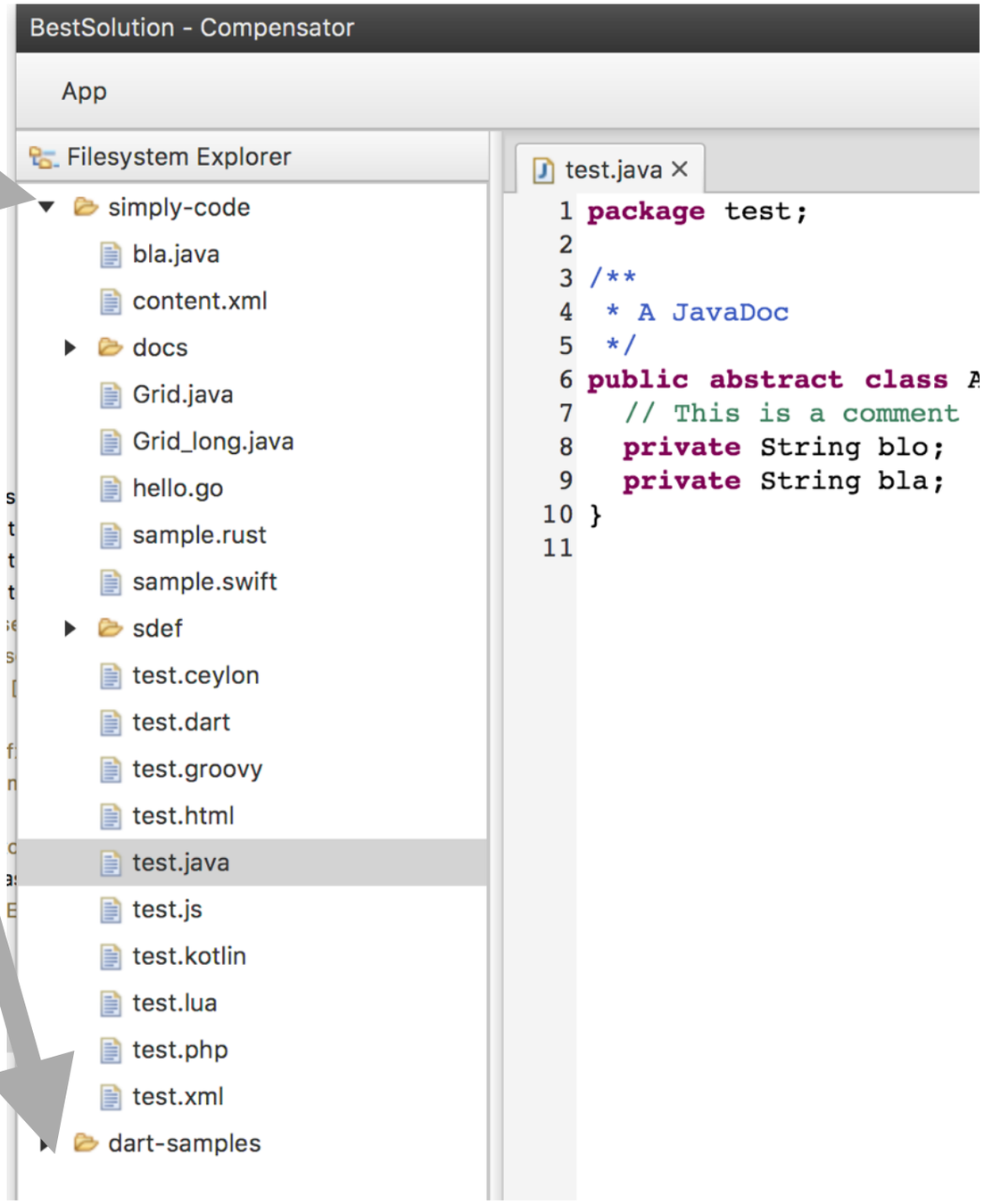
- **MApplicationElement #persistedState: Map<String,String>**: To be used for UI states



Preferences and UI-State

- **Preference-Store**: Used for none UI states

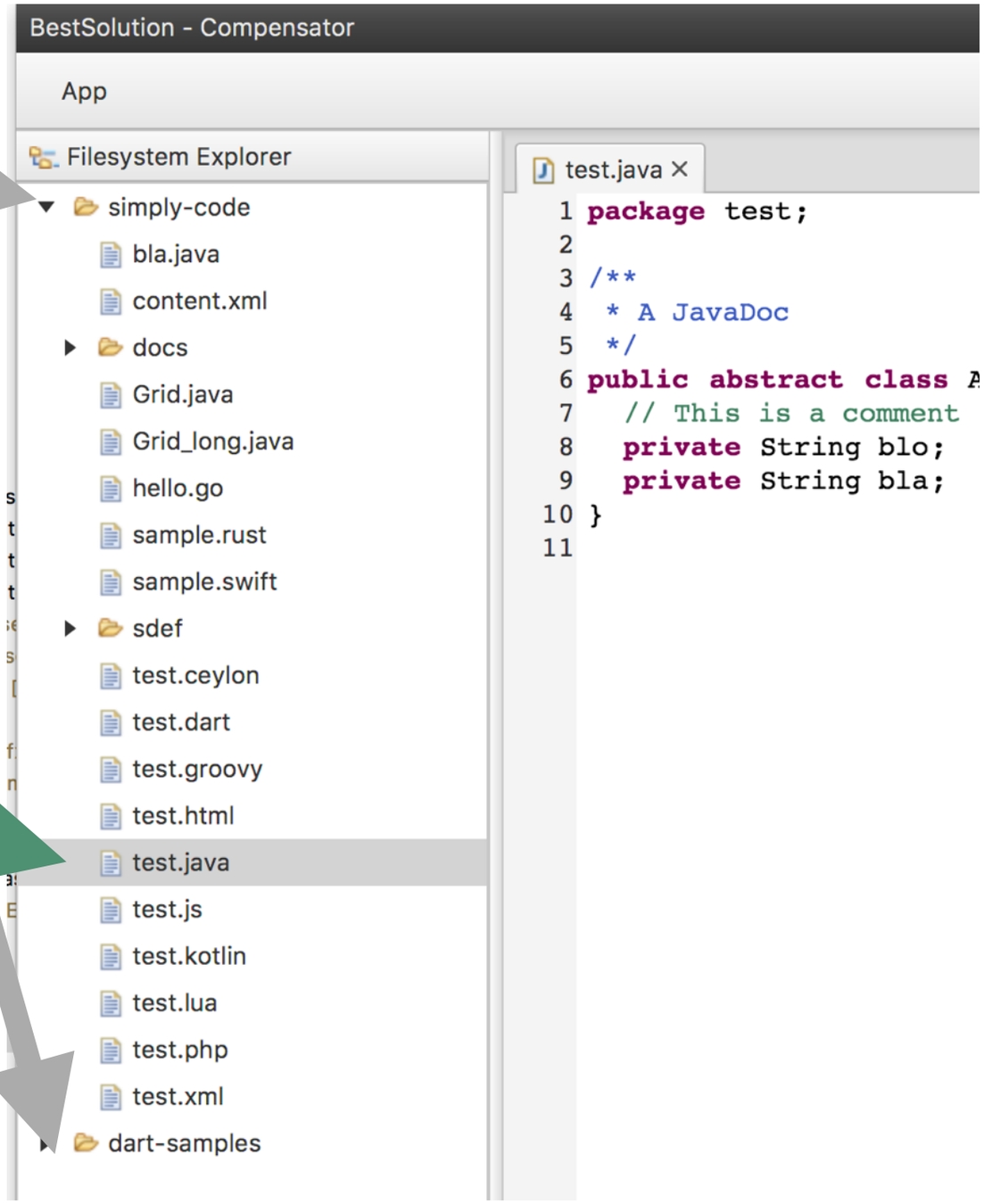
- **MApplicationElement #persistedState: Map<String,String>**: To be used for UI states



Preferences and UI-State

- **Preference-Store**: Used for none UI states

- **MApplicationElement #persistedState: Map<String,String>**: To be used for UI states



State persistence (e4)

State persistence (e4)

- @Preference

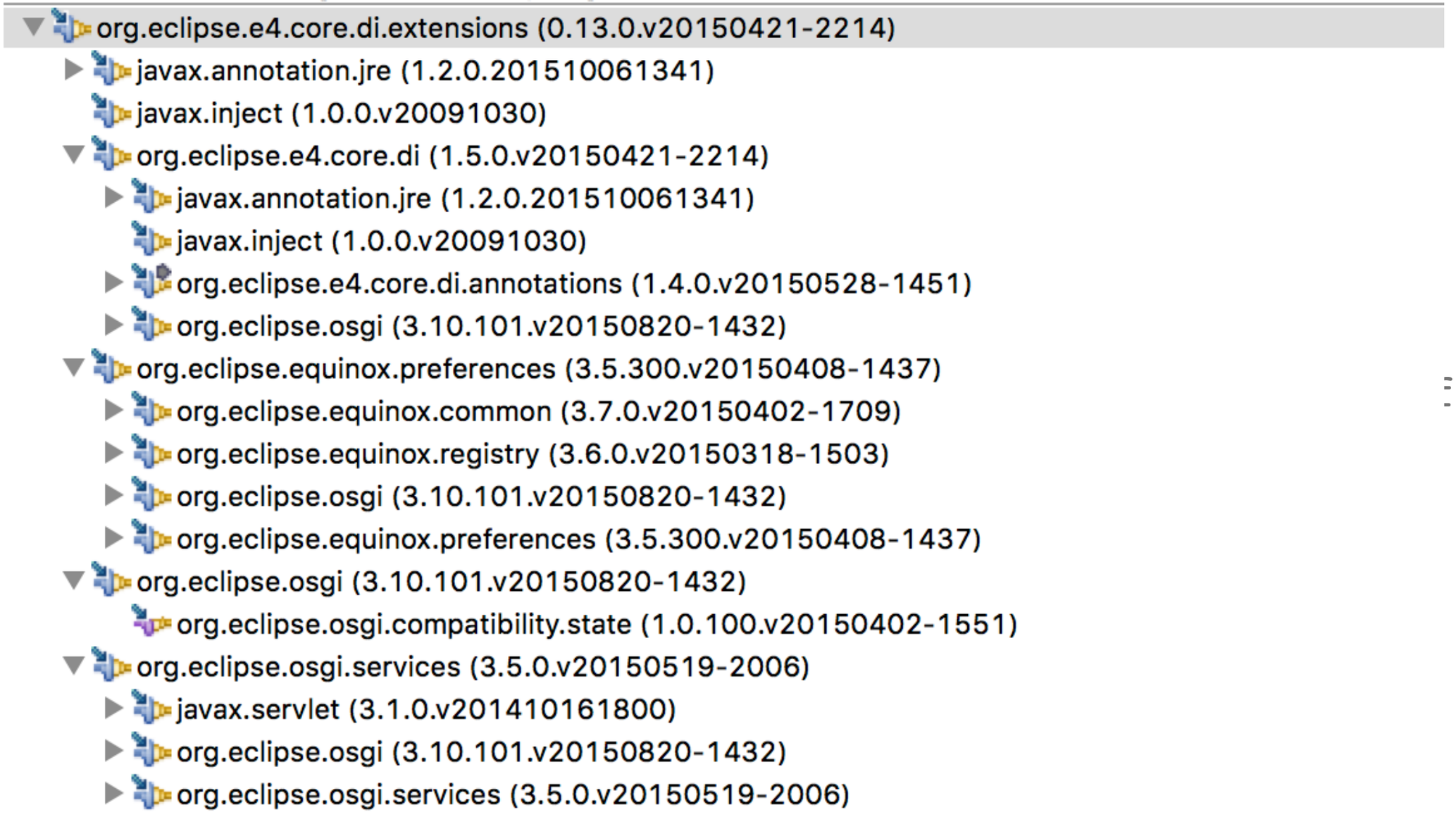
State persistence (e4)

- @Preference
  - Supports only simple values (int, String, …)

State persistence (e4)

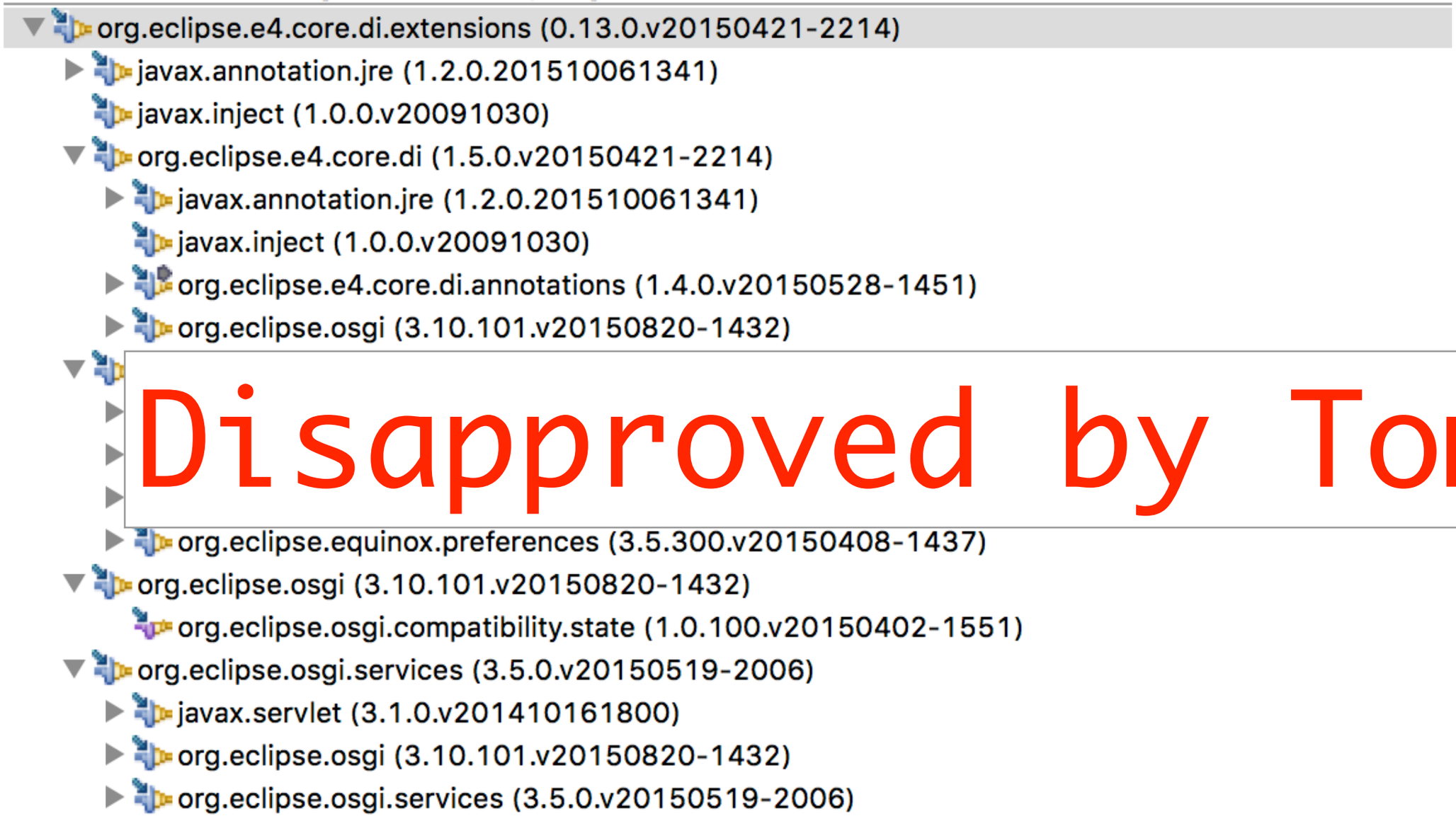
- @Preference
  - Supports only simple values (int, String, …)
  - Our application code gets a dependency on org.eclipse.core.runtime.preferences.IEclipsePreference

State persistence (e4)



State persistence (e4)

BestSolution



State persistence (e4)

State persistence (e4)

- MApplicationElement#persistedState : Map<String,String>

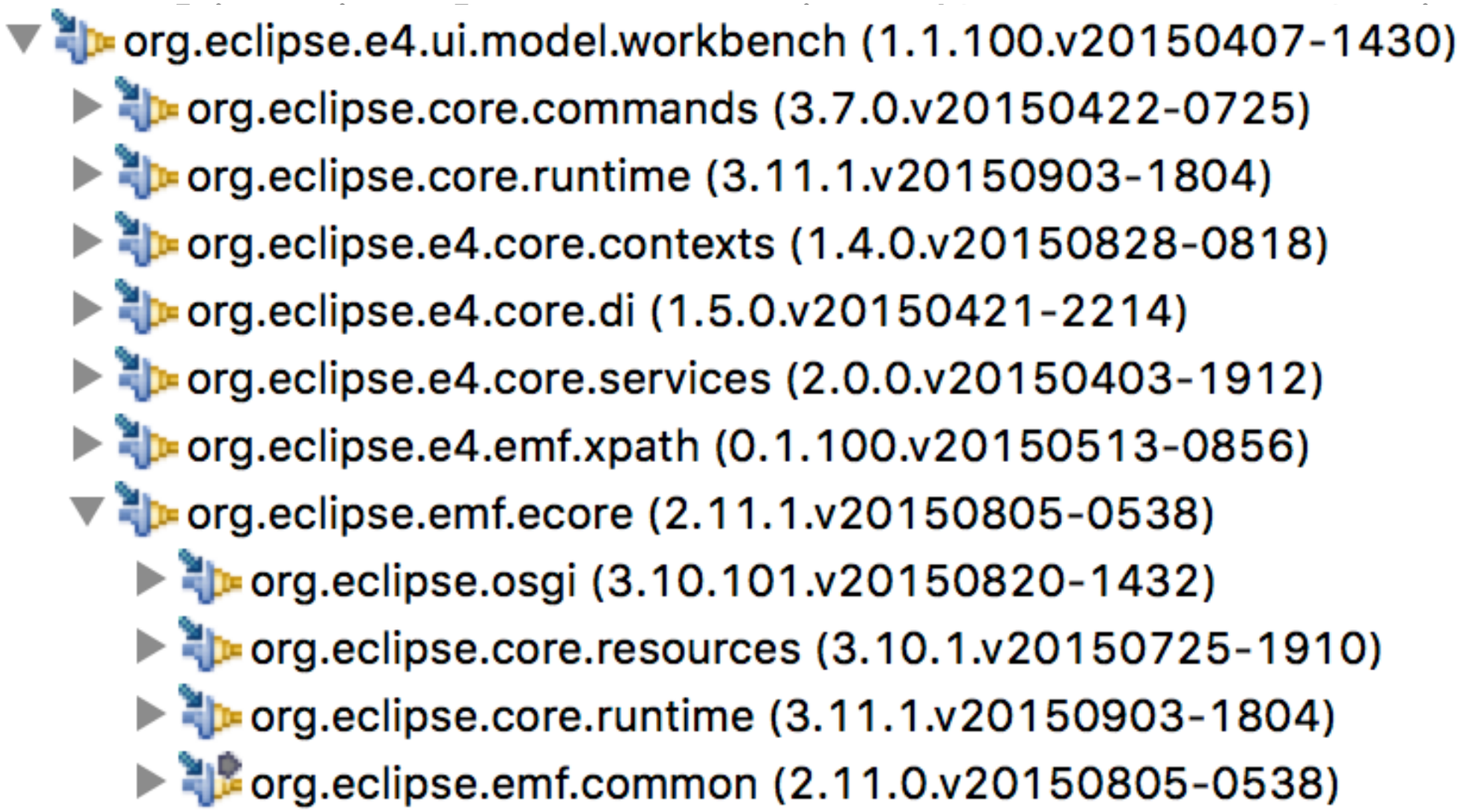
State persistence (e4)

- MApplicationElement#persistedState : Map<String,String>
  - Supports only storage of String

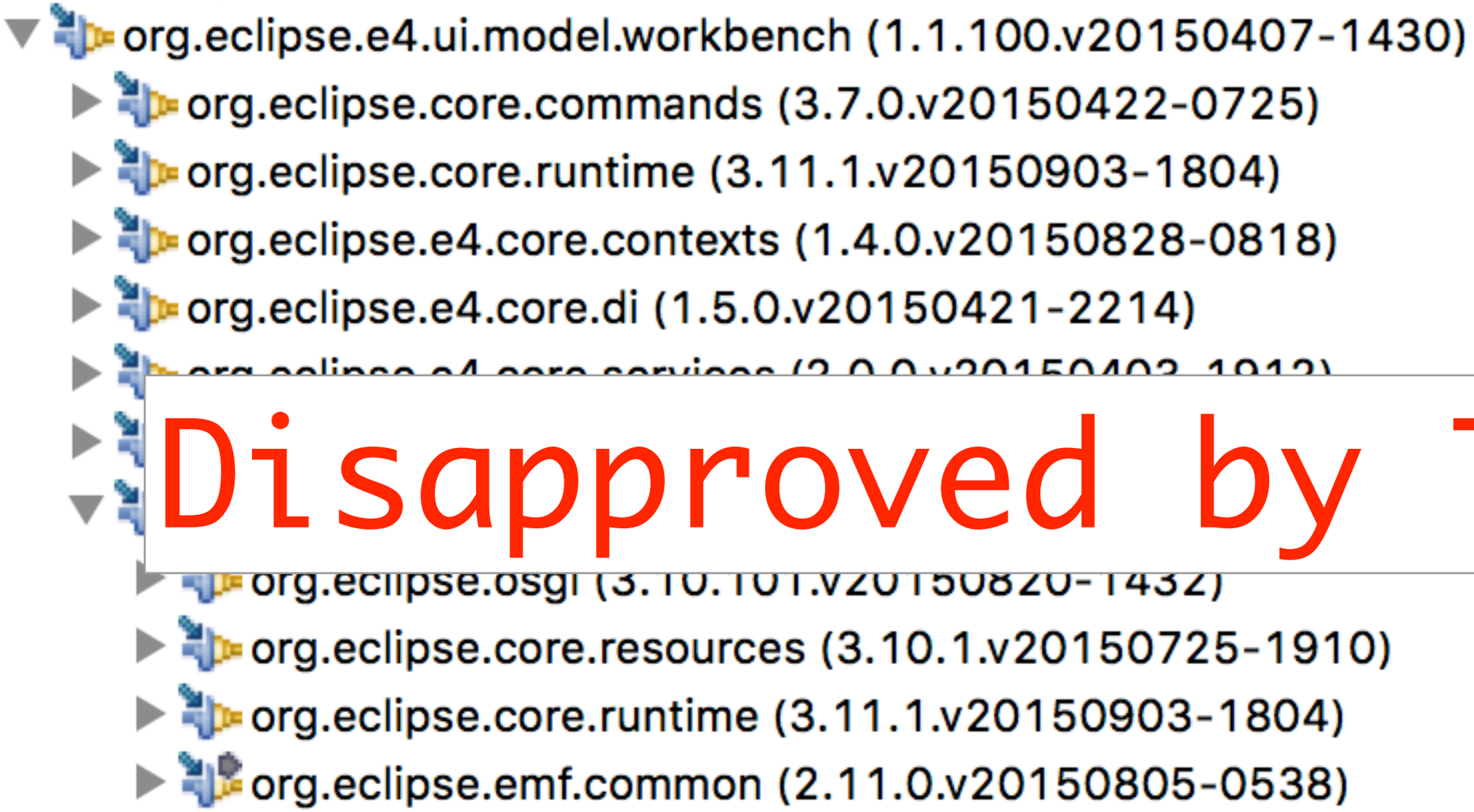
State persistence (e4)

- MApplicationElement#persistedState : Map<String,String>
  - Supports only storage of String
  - Your code gets a dependency on e4 application model and transitiv on EMF

State persistence (e4)



State persistence (e4)



Disapproved by Tom

State persistence (efx)

State persistence (efx)

- `@Preference` from e(fx)clipse

State persistence (efx)

- `@Preference` from e(fx)clipse
  - Supports (pluggable) storage of complex values (by default JAXB is used/available)

State persistence (efx)

- `@Preference` from e(fx)clipse
  - Supports (pluggable) storage of complex values (by default JAXB is used/available)
  - Supports (pluggable) storage of most „value“ types in java (eg `BigDecimal`, `Date`, `Instant`, …)

State persistence (efx)

- `@Preference` from e(fx)clipse
  - Supports (pluggable) storage of complex values (by default JAXB is used/available)
  - Supports (pluggable) storage of most „value“ types in java (eg `BigDecimal`, `Date`, `Instant`, …)
  - Injects a simple placeholder (…`preference.Value`) for the preference slot in question

State persistence (efx)

State persistence (efx)

- org.eclipse.fx.Memento-Interface

State persistence (efx)

- `org.eclipse.fx.Memento`-Interface
  - Small interface with no external dependencies

State persistence (efx)

- `org.eclipse.fx.Memento`-Interface
  - Small interface with no external dependencies
  - Allows (pluggable) storage of complex values

Preference

Preference: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final IEclipsePreferences pref;

   @Inject
   public MyStateSample(@Preference("rootDirs") IEclipsePreferences pref) {
      this.pref = pref;
   }

   private void rememberRootDirs(List<String> rootDirs)
     throws BackingStoreException {
      String string = rootDirs.stream().collect(Collectors.joining("##-##"));
      this.pref.put("rootDirs", string);
      this.pref.flush();
   }

   @Inject
   void setRootDirs(@Preference("rootDirs") String rootDirs) {
      String[] dirs = rootDirs.split("##-##");
   }

}
```

Preference: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final IEclipsePreferences pref;

   @Inject
   public MyStateSample(@Preference("rootDirs") IEclipsePreferences pref) {
      this.pref = pref;
   }

   private void rememberRootDirs(List<String> rootDirs)
     throws BackingStoreException {
      String string = rootDirs.stream().collect(Collectors.joining("##-##"));
      this.pref.put("rootDirs", string);
      this.pref.flush();
   }

   @Inject
   void setRootDirs(@Preference("rootDirs") String rootDirs) {
      String[] dirs = rootDirs.split("##-##");
   }
}
```

NPE Ahead!!!

Preference: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final Value<String> pref;

   @Inject
   public MyStateSample(@Preference(key="rootDirs") Value<String> pref) {
      this.pref = pref;
   }

   private void rememberRootDirs(List<String> rootDirs)
     throws BackingStoreException {
      String string = rootDirs.stream().collect(Collectors.joining("##-##"));
      this.pref.put("rootDirs", string);
      this.pref.flush();
   }

   @Inject
   void setRootDirs(@Preference("rootDirs") String rootDirs) {
      String[] dirs = rootDirs.split("##-##");
   }

}
```

Preference: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final Value<String> pref;

   @Inject
   public MyStateSample(@Preference(key="rootDirs") Value<String> pref) {
      this.pref = pref;
   }

   private void rememberRootDirs(List<String> rootDirs) {

      String string = rootDirs.stream().collect(Collectors.joining("##-##"));
      this.pref.publish(string);

   }

   @Inject
   void setRootDirs(@Preference("rootDirs") String rootDirs) {
      String[] dirs = rootDirs.split("##-##");
   }

}
```

Preference: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final Value<String> pref;

   @Inject
   public MyStateSample(@Preference(key="rootDirs") Value<String> pref) {
      this.pref = pref;
   }

   private void rememberRootDirs(List<String> rootDirs) {

      String string = rootDirs.stream().collect(Collectors.joining("##-##"));
      this.pref.publish(string);

   }

   @Inject
   void setRootDirs(@Preference(key="rootDirs",defaultValue="") String rootDirs) {
      String[] dirs = rootDirs.split("##-##");
   }

}
```

Preference: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final Value<List<String>> pref;

   @Inject
   public MyStateSample(@Preference(key="rootDirs") Value<String> pref) {
      this.pref = pref;
   }

   private void rememberRootDirs(List<String> rootDirs) {


      this.pref.publish(rootDirs);

   }

   @Inject
   void setRootDirs(@Preference(key="rootDirs",factory=EmptyListFactory.class)
     List<String> rootDirs) {
   }
}
```

UI State

UI-State: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final MPart part;

   @Inject
   public MyStateSample(MPart part) {
      this.part = part;
      String string = this.part.getPersistedState().get("selectionList");
      String[] selection = string.split("##-##");
   }

   private void rememberSelection(List<String> selectionList) {
      String string = selectionList.stream().collect(Collectors.joining("##-##"));
      this.part.getPersistedState().put("selectionList", string);
   }
}
```

UI-State: From e4 to efx


```
public class MyStateSample {
   private final MPart part;

   @Inject
   public MyStateSample(MPart part) {
      this.part = part;
      String string = this.part.getPersistedState().get("selectionList");
      String[] selection = string.split("##-##"`
   }

   private void rememberSelection(List<String> selectionList) {
      String string = selectionList.stream().collect(Collectors.joining("##-##"));
      this.part.getPersistedState().put("selectionList", string);
   }
}
```

NPE Ahead!!!

UI-State: From e4 to efx

BestSolution

```
public class MyStateSample {
    private final Memento memento;

    @Inject
    public MyStateSample(Memento memento) {
        this.memento = memento;
        String string = this.memento.get("selectionList","");
        String[] selection = string.split("##-##");
    }

    private void rememberSelection(List<String> selectionList) {
        String string = selectionList.stream().collect(Collectors.joining("##-##"));
        this.part.getPersistedState().put("selectionList", string);
    }
}
```

UI-State: From e4 to efx

BestSolution

```
public class MyStateSample {
    private final Memento part;

    @Inject
    public MyStateSample(Memento part) {
        this.part = part;
        String string = this.part.get("selectionList","");
        String[] selection = string.split("##-##");
    }

    private void rememberSelection(List<String> selectionList) {
        String string = selectionList.stream().collect(Collectors.joining("##-##"));
        this.part.put("selectionList", string);
    }
}
```

UI-State: From e4 to efx

BestSolution

```
public class MyStateSample {
   private final Memento part;

   @Inject
   public MyStateSample(Memento part) {
      this.part = part;
      List<String> selection = this.part.get("selectionList",List.class,
        new ArrayList<>());
   }

   private void rememberSelection(List<String> selectionList) {
      this.part.put("selectionList", selectionList,
        ObjectSerializer.JAXB_SERIALIZER);
   }
}
```

Dialogs in FX

Lightweight Dialogs

- New Service to open lightweight dialogs

Lightweight Dialogs

- New Service to open lightweight dialogs

```
import org.eclipse.fx.ui.controls.stage.Frame;

public interface LightWeightDialogService {
  public enum ModalityScope {
    WINDOW,
    PERSPECTIVE,
    PART
  }

  public <T extends Node & Frame> T openDialog(Class<T> dialogClass, ModalityScope scope);
  public <T extends Node & Frame> void openDialog(T dialog, ModalityScope scope);
}
```

Lightweight Dialogs

- New Service to open lightweight dialogs

```
import org.eclipse.fx.ui.controls.stage.Frame;

public interface LightWeightDialogService {
  public enum ModalityScope {
    WINDOW,
    PERSPECTIVE,
    PART
  }

  public <T extends Node & Frame> T openDialog(Class<T> dialogClass, ModalityScope scope);
  public <T extends Node & Frame> void openDialog(T dialog, ModalityScope scope);
}
```

- Usage

Lightweight Dialogs

New Service to open lightweight dialogs

```
import org.eclipse.fx.ui.controls.stage.Frame;

public interface LightWeightDialogService {
  public enum ModalityScope {
    WINDOW,
    PERSPECTIVE,
    PART
  }

  public <T extends Node & Frame> T openDialog(Class<T> dialogClass, ModalityScope scope);
  public <T extends Node & Frame> void openDialog(T dialog, ModalityScope scope);
}
```

Usage

```
public class NewProjectDialogHandler {
    @Execute
    public void createProject(LightWeightDialogService dialogService) {
        dialogService.openDialog(NewProjectDialog.class, ModalityScope.WINDOW);
    }
}
```

BestSolution

ScreenFlow Ablage Bearbeiten Markieren Einfügen Schrift Aktionen Darstellung Fenster Hilfe 97 % Di. 10:53

BestSolution - Compensator

App Projects

Test

- Sources
- Task Tracker
- CI System

MathHelper.java

```
1  package test;
2
3  public class MathHelper {
4      private String test;
5      private String bla;
6      private String blo;
7      private String bli;
8      public static double div(int a, int b) {
9          return a / b;
10     }
11 }
12
```

Outline

- MathHelper
  - bla : String
  - bli : String
  - blo : String
  - test : String
  - div(int, int) :

BestSolution

ScreenFlow Ablage Bearbeiten Markieren Einfügen Schrift Aktionen Darstellung Fenster Hilfe 97 % Di. 10:53

BestSolution - Compensator

App Projects

Test

- Sources
- Task Tracker
- CI System

MathHelper.java

```
1  package test;
2
3  public class MathHelper {
4      private String test;
5      private String bla;
6      private String blo;
7      private String bli;
8      public static double div(int a, int b) {
9          return a / b;
10     }
11 }
12
```

Outline

- MathHelper
  - bla : String
  - bli : String
  - blo : String
  - test : String
  - div(int, int) :

BestSolution

DI & OSGi

DI and OSGi-Services

DI and OSGi-Services

- Eclipse DI supports OSGi-Service injection BUT

DI and OSGi-Services

- Eclipse DI supports OSGi-Service injection BUT
  - it misses the dynamics of OSGi

DI and OSGi-Services

BestSolution

- Eclipse DI supports OSGi-Service injection BUT
  - it misses the dynamics of OSGi
  - You can only inject the highest ranked service

@Service

- `@Service` annotation

@Service

- `@Service` annotation

```
public class OSGiService {
  @Inject
  @Service
  FilesystemService fs;

  @Inject
  @Service
  List<FilesystemService> filesystemServiceList;
}
```

@Service

- `@Service` annotation

```
public class OSGiService {
  @Inject
  @Service
  FilesystemService fs;

  @Inject
  @Service
  List<FilesystemService> filesystemServiceList;
}
```

```
public Class<?> getCollectionType(java.lang.reflect.Type t) {
  if( t instanceof java.lang.reflect.ParameterizedType ) {
    java.lang.reflect.ParameterizedType pt = (java.lang.reflect.ParameterizedType) t;
    return (Class<?>) pt.getRawType();
  }
  return (Class<?>) t;
}
```

@LocalInstance

@LocalInstance

BestSolution

```
if( container != null ) {
  part = modelService.createModelElement(MPart.class);
  part.setCloseable(true);
  part.setLabel(URI.create(uri).lastSegment());
  String editorBundleURI = editorUrlProvider
    .stream()
    .filter( e -> e.test(uri)).findFirst()
    .map( e -> e.getBundleClassURI(uri))
    .orElse("bundleclass://org.eclipse.fx.code.editor.fx/org.eclipse.fx.code.editor.fx.TextEditor");
  part.setContributionURI(editorBundleURI);


  String iconUri = fileIconProvider
    .stream()
    .filter( f -> f.test(uri))
    .findFirst()
    .map( f -> f.getFileIconUri(uri))
    .orElse("platform:/plugin/org.eclipse.fx.code.editor.fx.e4/icons/file_16.png");
  part.setIconURI(iconUri);
  part.getPersistedState().put(Constants.DOCUMENT_URL, uri);
  part.getTags().add(EPartService.REMOVE_ON_HIDE_TAG);
  container.getChildren().add(part);
}
```

@LocalInstance

BestSolution

```
if( container != null ) {
  part = modelService.createModelElement(MPart.class);
  part.setCloseable(true);
  part.se
  String
    .stre
    .filt
    .map(
    .orElse("bundleclass://org.eclipse.fx.code.editor.fx/org.eclipse.fx.code.editor.fx.TextEditor");
  part.setContributionURI(editorBundleURI);


  String iconUri = fileIconProvider
    .stream()
    .filter( f -> f.test(uri))
    .findFirst()
    .map( f -> f.getFileIconUri(uri))
    .orElse("platform:/plugin/org.eclipse.fx.code.editor.fx.e4/icons/file_16.png");
  part.setIconURI(iconUri);
  part.getPersistedState().put(Constants.DOCUMENT_URL, uri);
  part.getTags().add(EPartService.REMOVE_ON_HIDE_TAG);
  container.getChildren().add(part);
}
```

What's wrong here?

@LocalInstance

BestSolution

```
if( container != null ) {
  part = modelService.createModelElement(MPart.class);
  part.setCloseable(true);
  part.setLabel(URI.create(uri).lastSegment());
  String editorBundleURI = editorUrlProvider
    .stream()
    .filter( e -> e.test(uri)).findFirst()
    .map( e -> e.getBundleClassURI(uri))
    .orElse("bundleclass://org.eclipse.fx.code.editor.fx/org.eclipse.fx.code.editor.fx.TextEditor");
  part.setContributionURI(editorBundleURI);
  part.setContributorURI("platform:/plugin/org.eclipse.fx.code.editor.fx.e4");

  String iconUri = fileIconProvider
    .stream()
    .filter( f -> f.test(uri))
    .findFirst()
    .map( f -> f.getFileIconUri(uri))
    .orElse("platform:/plugin/org.eclipse.fx.code.editor.fx.e4/icons/file_16.png");
  part.setIconURI(iconUri);
  part.getPersistedState().put(Constants.DOCUMENT_URL, uri);
  part.getTags().add(EPartService.REMOVE_ON_HIDE_TAG);
  container.getChildren().add(part);
}
```

@LocalInstance

- A first fix: enhance `EModelService` with

```
public <T extends MApplicationElement> T createModelElement(
  Class<?> caller,
  Class<T> elementType);
```

@LocalInstance

BestSolution

- A first fix: enhance `EModelService` with

```
public <T extends MApplicationElement> T createModelElement(
  Class<?> caller,
  Class<T> elementType);
```

- Another fix: Provide a factory service

```
public interface ModelElementFactoryService {
  ModelElementFactory createFactory(Class<?> caller);
}

public interface ModelElementFactory {
  <T extends MApplicationElement> T createElement(Class<T> elementType);
}
```

@LocalInstance

BestSolution

- A first fix: enhance `EModelService` with

```
public <T extends MApplicationElement> T createModelElement(
  Class<?> caller,
  Class<T> elementType);
```

- Another fix: Provide a factory service

```
private ModelElementFactory f;

@Inject
public LocalInstanceSample(ModelElementFactoryService s) {
  this.f = s.createFactory(getClass());
}

public void init() {
  // ...

  part = f.createModelElement(MPart.class);

  // ...
}
```

@LocalInstance

- Is there a more generic solution to create DI-Instance values with knowledge about the target (eg classloader, bundle, …)?

- Since 2.1 there a basic TypeProvider API in fx.core

@LocalInstance

BestSolution

- Is there a more generic solution to create DI-Instance values with knowledge about the target (eg classloader, bundle, …)?

- Since 2.1 there a basic TypeProvider API in fx.core

```
/**
 * A service who provides a type based upon selector value
 *
 * @param <S>
 *            the selector value type
 * @param <T>
 *            the type
 * @since 2.1
 */
public interface TypeProviderService<S, T> extends Predicate<S> {
    @Override
    boolean test(S t);

    public Class<? extends T> getType(S s);
}
```

@LocalInstance

- Upon the basic interface there are a specialized ones eg

@LocalInstance

BestSolution

- Upon the basic interface there are a specialized ones eg

```
/**
 * Service who provides a real type for an interface/abstract type
 *
 * @param <T>
 *            the type
 */
public interface TypeTypeProviderService<T>
   extends TypeProviderService<java.lang.reflect.Type, T> {

  /**
   * If used with dependency injection and @LocalInstance
   */
  public static final String DI_KEY = "localInstanceOwnerType"; //$NON-NLS-1$

}
```

@LocalInstance

@LocalInstance

BestSolution

```
@Component
class TypeProviderImpl implements TypeTypeProviderService<ModelElementFactory> {

  @Override
  public Class<? extends ModelElementFactory> getType(Type s) {
    return ModelElementFactoryImpl.class;
  }

  @Override
  public boolean test(Type t) {
    return t == ModelElementFactory.class;
  }

  public static class ModelElementFactoryImpl implements ModelElementFactory {
    @NonNull
    private final EModelService modelService;

    @Inject
    public ModelElementFactoryImpl(@NonNull EModelService modelService,
      @Named(TypeTypeProviderService.DI_KEY) @NonNull Class<?> ownerType) {
      this.modelService = modelService;
    }
  }
}
```

@LocalInstance

BestSolution

- The original code

```
@Inject
private EModelService modelService;

public void init() {
  // ...

  part = modelService.createModelElement(MPart.class);

  // ...
}
```

@LocalInstance

BestSolution

- The original code

```
@Inject
private EModelService modelService;

public void init() {
  // ...

  part = modelService.createModelElement(MPart.class);

  // ...
}
```

- Gets to

```
@Inject
@LocalInstance
private ModelElementFactory modelFactory;

public void init() {
  // ...

  part = modelFactory.createModelElement(MPart.class);

  // ...
}
```

@LocalInstance

BestSolution

- The original code

```
@Inject
private EModelService modelService;

public void init() {
  // ...

  part = modelService.createModelElement(MPart.class);

  // ...
}
```

LocalInstanceObjectSupplier

- Gets to

```
@Inject
@LocalInstance
private ModelElementFactory modelFactory;

public void init() {
  // ...

  part = modelFactory.createModelElement(MPart.class);

  // ...
}
```

@LocalInstance

BestSolution

- The original code

```
@Inject
private EModelService modelService;

public void init() {
  // ...

  part = modelService.createModelElement(MPart.class);

  // ...
}
```

- Gets to

```
@Inject
@LocalInstance
private ModelElementFactory modelFactory;

public void init() {
  // ...

  part = modelFactory.createModelElement(MPart.class);

  // ...
}
```

@LocalInstance

BestSolution

- The original code

```
@Inject
private EModelService modelService;

public void init() {
  // ...

  part = modelService.createModelElement(MPart.class);

  // ...
}
```

- Gets to

```
@Inject
@LocalInstance
private ModelElementFactory modelFactory;

public void init() {
  // ...

  part = modelFactory.createModelElement(MPart.class);

  // ...
}
```