

Nasdanika HTML

Fluent Java API for building Web UI

Overview

- Fluent Java API for building:
 - Low level – HTML elements
 - Mid level:
 - Bootstrap 4.x UI elements and 21 Bootswatch themes
 - ECharts 4.2.1 factory and option builders
 - Font Awesome 5.x icons
 - jsTree 3.3.8 nodes and context menus
 - KnockoutJS 3.4.x
 - High level – HTML Applications
 - Using abstractions of actions and property sources
 - From EMF models data and meta-data
 - Generation of documentation for Ecore models
- Dual delivery:
 - OSGi bundles – p2 repository
 - Jars - Maven repository (excluding EMF)

HTML

- API for building HTML elements
- Foundation for the other modules
- How to use:
 - Obtain HTMLFactory
 - Create HTML elements
 - Configure the elements:
 - Attributes
 - CSS Classes
 - Styles
- Output with `toString()` or `produce()`

```
HTMLFactory htmlFactory = HTMLFactory.INSTANCE;  
Tag div = htmlFactory.div("Hello")  
    .style().border("solid 1px")  
    .on(Event.click, getClass().getResource("ClickHandler.js"));  
System.out.println(div);
```



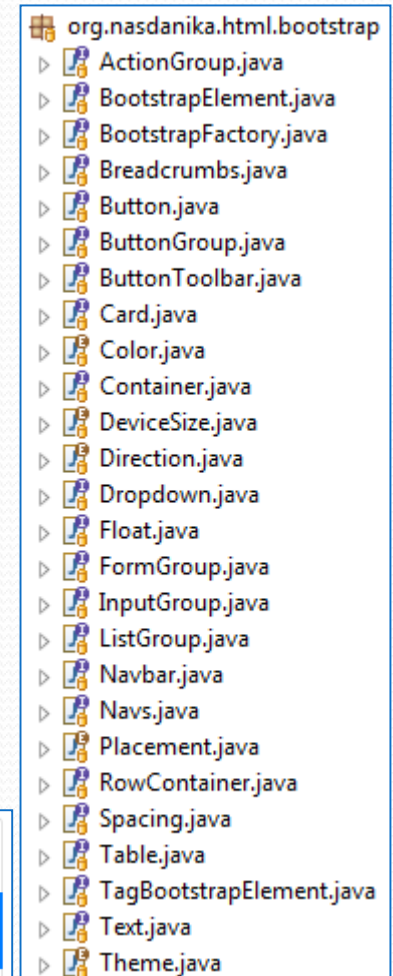
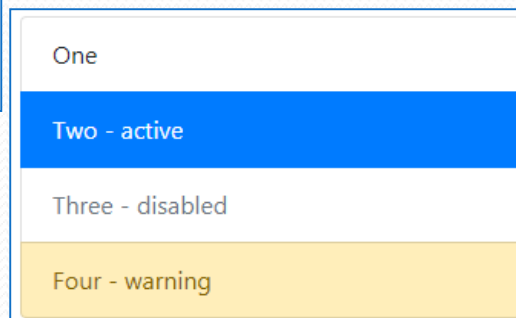
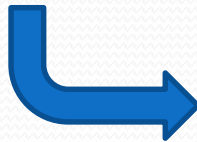
```
<div onclick="alert('Hi!')" style="border:solid 1px">Hello</div>
```

- org.nasdanika.html
 - > Button.java
 - > Color.java
 - > Container.java
 - > Event.java
 - > FieldContainer.java
 - > FieldSet.java
 - > Form.java
 - > FormFragment.java
 - > Fragment.java
 - > Function.java
 - > HTMLElement.java
 - > HTMLElementFilter.java
 - > HTMLFactory.java
 - > HTMLPage.java
 - > Input.java
 - > InputBase.java
 - > InputType.java
 - > Markup.java
 - > NamedItemsContainer.java
 - > Producer.java
 - > ProducerException.java
 - > RowContainer.java
 - > Select.java
 - > Style.java
 - > Table.java
 - > Tag.java
 - > TagName.java
 - > TextArea.java
 - > TokenSource.java

Bootstrap

- API for building Bootstrap 4 elements
- Built on HTML
- How to use:
 - Obtain BootstrapFactory
 - Create elements
 - Configure the elements
- Output with `toString()` or `produce()`
- Fallback to HTML API's when needed

```
ListGroup listGroup = BootstrapFactory.INSTANCE.listGroup(false);  
  
listGroup.item(false, false, Color.DEFAULT, "One");  
listGroup.item(true, false, Color.DEFAULT, "Two - active");  
listGroup.item(false, true, Color.DEFAULT, "Three - disabled");  
listGroup.item(false, false, Color.WARNING, "Four - warning");
```



Font Awesome

- API for building Font Awesome 5 icons
- Built on HTML
- How to use:
 - Obtain `FontAwesomeFactory`
 - Create icons
 - Configure the icons
- Output with `toString()` or `produce()`
- Fallback to HTML API's when needed

```
org.nasdanika.html.fontawesome
├── FontAwesomeFactory.java
│   ├── FontAwesomeFactory
│   │   ├── INSTANCE
│   │   ├── cdn(P) <P extends HTMLPage> : P
│   │   ├── from(String, Style, T) <T extends HTMLElement<?>> : Icon<T>
│   │   ├── getHTMLFactory() : HTMLFactory
│   │   ├── icon(String, Style) : Icon<Tag>
│   │   └── stack() : Stack
│   └── Icon.java
│       ├── Icon<T extends HTMLElement<?>>
│       │   ├── Flip
│       │   ├── Rotate
│       │   ├── Size
│       │   ├── Stack
│       │   └── Style
│       │       ├── fixedWidth() : Icon<T>
│       │       ├── flip(Flip) : Icon<T>
│       │       ├── li() : Icon<T>
│       │       ├── pullLeft() : Icon<T>
│       │       ├── pullRight() : Icon<T>
│       │       ├── rotate(Rotate) : Icon<T>
│       │       ├── size(Size) : Icon<T>
│       │       ├── spin() : Icon<T>
│       │       ├── toHTMLElement() : T
│       │       └── ul() : Icon<T>
```

```
Icon<Tag> icon = FontAwesomeFactory.INSTANCE.icon("university", Style.SOLID)
    .size(Size.x5)
    .rotate(Rotate.R180);
```



jsTree

- API for building jsTree nodes and context menus
- Built on HTML
- How to use:
 - Obtain JsTreeFactory
 - Create nodes and menus
 - Configure the nodes and menus
- Output to JSON

```
JsTreeFactory jsTreeFactory = JsTreeFactory.INSTANCE;  
JsTreeNode rootNode = jsTreeFactory.jsTreeNode();  
rootNode.icon("far fa-user");  
rootNode.text("User");  
rootNode.id(htmlFactory.nextId());  
rootNode.hasChildren();  
JSONArray jsTreeRootNodes = new JSONArray();  
jsTreeRootNodes.put(rootNode.toJSON());
```

```
org.nasdanika.html.jstree  
├── JsTreeContextMenuItem.java  
│   ├── JsTreeContextMenuItem  
│   │   ├── action(Object) : JsTreeContextMenuItem  
│   │   ├── addSubMenuItem(String, JsTreeContextMenuItem) : JsTreeContextMenuItem  
│   │   ├── createSubMenuItem(String) : JsTreeContextMenuItem  
│   │   ├── disabled() : JsTreeContextMenuItem  
│   │   ├── disabled(boolean) : JsTreeContextMenuItem  
│   │   ├── icon(Object) : JsTreeContextMenuItem  
│   │   ├── label(Object) : JsTreeContextMenuItem  
│   │   ├── separatorAfter() : JsTreeContextMenuItem  
│   │   ├── separatorAfter(boolean) : JsTreeContextMenuItem  
│   │   ├── separatorBefore() : JsTreeContextMenuItem  
│   │   ├── separatorBefore(boolean) : JsTreeContextMenuItem  
│   │   ├── shortcut(Object) : JsTreeContextMenuItem  
│   │   ├── shortcutLabel(Object) : JsTreeContextMenuItem  
│   │   ├── subMenu(Object) : JsTreeContextMenuItem  
│   │   ├── title(Object) : JsTreeContextMenuItem  
│   │   └── toJSON() : JSONObject  
│   └── JsTreeFactory.java  
│       ├── JsTreeFactory  
│       │   └── INSTANCE  
│       │       ├── bind(HTMLElement<?>, Object) : Tag  
│       │       ├── bind(String, Object) : Tag  
│       │       ├── buildAjaxJsTree(String, String) : String  
│       │       ├── buildJsTree(Iterable<JsTreeNode>) : JSONObject  
│       │       ├── buildJsTree(JsTreeNode...) : JSONObject  
│       │       ├── cdn(P) <P extends HTMLPage> : P  
│       │       ├── getHTMLFactory() : HTMLFactory  
│       │       ├── jsTreeContextMenuItem() : JsTreeContextMenuItem  
│       │       └── jsTreeNode() : JsTreeNode  
│       └── JsTreeNode.java  
│           ├── JsTreeNode  
│           │   └── Collector<R>  
│           │       ├── accept(Collector<R>) <R> : R  
│           │       ├── anchorAttribute(String, Object) : JsTreeNode  
│           │       ├── children() : List<JsTreeNode>  
│           │       ├── createChild() : JsTreeNode  
│           │       ├── disabled() : JsTreeNode  
│           │       ├── disabled(boolean) : JsTreeNode  
│           │       ├── getData() : Object  
│           │       ├── getData(String) : Object  
│           │       ├── getId() : Object  
│           │       ├── hasChildren() : JsTreeNode  
│           │       ├── icon(String) : JsTreeNode  
│           │       ├── id(Object) : JsTreeNode  
│           │       ├── listItemAttribute(String, Object) : JsTreeNode  
│           │       ├── opened() : JsTreeNode  
│           │       ├── opened(boolean) : JsTreeNode  
│           │       ├── selected() : JsTreeNode  
│           │       ├── selected(boolean) : JsTreeNode  
│           │       ├── setData(Object) : JsTreeNode  
│           │       ├── setData(String, Object) : JsTreeNode  
│           │       ├── text(Object) : JsTreeNode  
│           │       ├── toJSON() : JSONObject  
│           │       └── toJSON(Predicate<JsTreeNode>) : JSONObject
```

ECharts

- API for building ECharts charts
- Built on HTML
- How to use:
 - Obtain EChartsFactory
 - Create OptionBuilder
 - Configure option and then create a chart

```
Card card = BootstrapFactory.INSTANCE.card().border(Color.SUCCESS);
card.getTitle().toHTMLElement().content("ECharts demo");
OptionBuilder optionBuilder = EChartsFactory.INSTANCE.createOptionBuilder();

optionBuilder.titleBuilder().text("Nasdanika ECharts example");

JSONArray legendData = new JSONArray();
legendData.put("Sales");
optionBuilder.legend().put("data", legendData);

JSONArray xAxisData = new JSONArray();
xAxisData.put("shirt");
xAxisData.put("cardign");
xAxisData.put("chiffon shirt");
xAxisData.put("pants");
xAxisData.put("heels");
xAxisData.put("socks");
optionBuilder.xAxis().put("data", xAxisData);

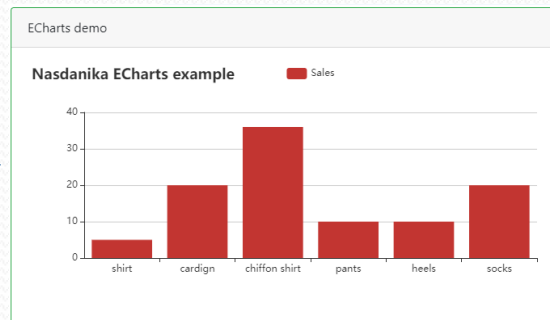
// yAxis object must be present,
// so we call this method to create an empty object.
optionBuilder.yAxis();

JSONObject salesSeries = new JSONObject();
salesSeries.put("name", "Sales");
salesSeries.put("type", "bar");

JSONArray salesData = new JSONArray();
salesData.put(5);
salesData.put(20);
salesData.put(36);
salesData.put(10);
salesData.put(10);
salesData.put(20);
salesSeries.put("data", salesData);

optionBuilder.series().put(salesSeries);

card.getBody().toHTMLElement().content(optionBuilder.create("700px", "300px"));
writeThemedPage("echarts/card.html", "Bootstrap card", card);
```



```
org.nasdanika.html.echarts
├── EChartsFactory.java
│   └── EChartsFactory
│       └── INSTANCE
│           ├── cdn(P) <P extends HTMLPage> : P
│           ├── create(JSONObject, Object, Object) : Fragment
│           ├── create(JSONObject, Object, Object, Consumer<Object>) : void
│           ├── create(Supplier<JSONObject>, Object, Object) : Fragment
│           ├── create(Supplier<JSONObject>, Object, Object, Consumer<Object>) : void
│           ├── createOptionBuilder() : OptionBuilder
│           ├── getHTMLFactory() : HTMLFactory
│           ├── init(HTMLElement<?>, JSONObject) : String
│           ├── init(HTMLElement<?>, Supplier<JSONObject>) : String
│           ├── init(String, JSONObject) : String
│           └── init(String, Supplier<JSONObject>) : String
└── OptionBuilder.java
    └── OptionBuilder
        ├── angleAxis() : JSONObject
        ├── animation() : JSONObject
        ├── animationDelay() : JSONObject
        ├── animationDelayUpdate() : JSONObject
        ├── animationDuration() : JSONObject
        ├── animationDurationUpdate() : JSONObject
        ├── animationEasing() : JSONObject
        ├── animationEasingUpdate() : JSONObject
        ├── animationThreshold() : JSONObject
        ├── aria() : JSONObject
        ├── axisPointer() : JSONObject
        ├── backgroundColor() : JSONObject
        ├── brush() : JSONObject
        ├── calendar() : JSONObject
        ├── color() : JSONObject
        ├── create(Object, Object) : Fragment
        ├── create(Object, Object, Consumer<Object>) : void
        ├── dataset() : JSONObject
        ├── dataZoom() : JSONArray
        ├── geo() : JSONObject
        ├── getFactory() : EChartsFactory
        ├── graphic() : JSONObject
        ├── grid() : JSONObject
        ├── init(HTMLElement<?>) : String
        ├── init(String) : String
        ├── legend() : JSONObject
        ├── parallel() : JSONObject
        ├── parallelAxis() : JSONObject
        ├── polar() : JSONObject
        ├── radar() : JSONObject
        ├── radiusAxis() : JSONObject
        ├── series() : JSONArray
        ├── singleAxis() : JSONObject
        ├── textStyle() : JSONObject
        ├── timeline() : JSONObject
        ├── title() : JSONObject
        ├── titleBuilder() : TitleBuilder
        ├── toolbox() : JSONObject
        ├── tooltip() : JSONObject
        ├── useUTC() : JSONObject
        ├── visualMap() : JSONArray
        ├── xAxis() : JSONObject
        └── yAxis() : JSONObject
```

KnockoutJS

- API for building KnockoutJS bindings
- Built on HTML

```
KnockoutBindingsSource.java
└─ KnockoutBindingsSource
   └─ Binding
      └─ getAllBindings(): Collection<Binding>
KnockoutControlFlow.java
└─ KnockoutControlFlow<T>
   └─ bind(String, Object, Object): T
      └─ component(Object): T
         └─ foreach(Object): T
            └─ foreach(Object, Object): T
               └─ generateObservables(String...): String
                  └─ if_(Object): T
                     └─ if_(Object, Object): T
                        └─ ifnot(Object): T
                           └─ ifnot(Object, Object): T
                              └─ with(Object): T
KnockoutFactory.java
└─ KnockoutFactory
   └─ INSTANCE
      └─ cdn(P) <P extends HTMLPage> : P
         └─ from(H) <H extends HTMLElement<?>> : Knockout<H>
            └─ getHTMLFactory(): HTMLFactory
               └─ virtualElement(Object...): KnockoutVirtualElement
KnockoutFilter.java
KnockoutVirtualElement.java
└─ KnockoutVirtualElement
   └─ getContent(): List<Object>
      └─ isEmpty(): boolean
```

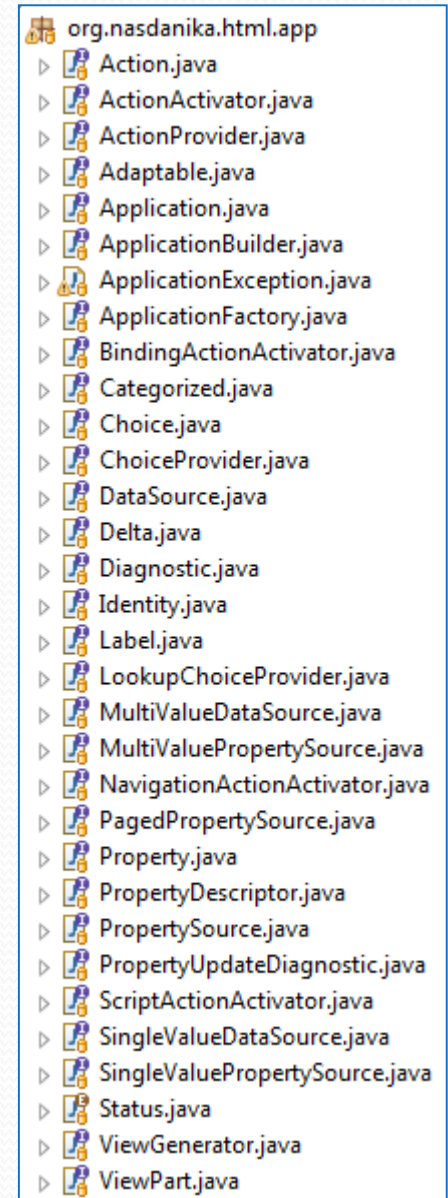
```
Knockout<H extends HTMLElement<?>>
└─ attr(Object): Knockout<H>
   └─ checked(Object): Knockout<H>
      └─ checked(Object, Object): Knockout<H>
         └─ click(Object): Knockout<H>
            └─ css(Object): Knockout<H>
               └─ disable(Object): Knockout<H>
                  └─ disable(Object, Object): Knockout<H>
                     └─ enable(Object): Knockout<H>
                        └─ enable(Object, Object): Knockout<H>
                           └─ event(Object): Knockout<H>
                              └─ hasFocus(Object): Knockout<H>
                                 └─ hasFocus(Object, Object): Knockout<H>
                                    └─ html(Object): Knockout<H>
                                       └─ html(Object, Object): Knockout<H>
                                          └─ options(Object): Knockout<H>
                                             └─ options(Object, Object): Knockout<H>
                                                └─ selectedOptions(Object): Knockout<H>
                                                   └─ selectedOptions(Object, Object): Knockout<H>
                                                      └─ style(Object): Knockout<H>
                                                         └─ submit(Object): Knockout<H>
                                                            └─ template(Object): Knockout<H>
                                                               └─ text(Object): Knockout<H>
                                                                  └─ text(Object, Object): Knockout<H>
                                                                     └─ textInput(Object): Knockout<H>
                                                                        └─ textInput(Object, Object): Knockout<H>
                                                                           └─ toHTMLElement(): H
                                                                              └─ uniqueName(Object): Knockout<H>
                                                                                 └─ value(Object): Knockout<H>
                                                                                    └─ value(Object, Object): Knockout<H>
                                                                                       └─ visible(Object): Knockout<H>
                                                                                          └─ visible(Object, Object): Knockout<H>
```


Application - philosophy

- Abstractions to think of user-system interaction as:
 - System invokes a user by passing them a callback (user) interface with actions to activate
 - Actions form a vocabulary of system-user interactions
 - The framework takes care of generating an HTML UI from actions and property sources
- Subject - Verb - Object => User - Action - Property source:
 - "Customer views account details":
 - Customer - user
 - Views account details - view action for "account" property source
- If it can be articulated, it can be automated

Application – key abstractions

- Label – something with a text and an icon
- Action – a label which can be activated by a user
- Data sources and properties – low-level data access
- Property sources and property descriptors – higher-level data access abstractions with UI attributes and actions
- Application – header, navigation bar, navigation panel, content panel, footer
- Application Builder – builds an application
- Action Application Builder – builds an application from an action tree
- ViewPart – a contributor to UI construction
- ViewGenerator – provides common generation methods and access to factories – HTML, Bootstrap, ...



EMF

- EMF adapters to the application abstractions
- Use default implementations or customize
- Register with a resource set:

```
ComposedAdapterFactory composedAdapterFactory = new ComposedAdapterFactory();

composedAdapterFactory.registerAdapterFactory(
    new SupplierAdapterFactory<ApplicationFactory>(
        ApplicationFactory.class,
        this.getClass().getClassLoader(),
        BootstrapContainerApplicationFactory::new));

composedAdapterFactory.registerAdapterFactory(
    new FunctionAdapterFactory<ApplicationBuilder, EObject>(
        ApplicationBuilder.class,
        this.getClass().getClassLoader(),
        ViewActionApplicationBuilder::new));

composedAdapterFactory.registerAdapterFactory(
    new FunctionAdapterFactory<ViewAction, EObject>(
        ViewAction.class,
        this.getClass().getClassLoader(),
        EObjectViewAction::new));

resourceSet.getAdapterFactories().add(composedAdapterFactory);
```

- Adapt EObject to generate HTML UI:

```
Application application = EObjectAdaptable.adaptTo(eObj, ApplicationFactory.class).createApplication();
ApplicationBuilder applicationBuilder = EObjectAdaptable.adaptTo(eObj, ApplicationBuilder.class);
applicationBuilder.build(application);
```

- Can be used to generate static content and in dynamic Web applications

```
org.nasdanika.html.emf
> ContentPanelViewPart.java
> EClassLabel.java
> EClassPropertySource.java
> EditAction.java
> ENamedElementLabel.java
> EObjectAdaptable.java
> EObjectSingleValueDataSource.java
> EObjectSingleValuePropertySource.java
> EObjectViewAction.java
> EReferenceMultiValuePropertySource.java
> EReferenceMultiValuePropertySourceViewAction.java
> EReferenceSingleValuePropertySource.java
> EReferenceSingleValuePropertySourceViewAction.java
> EStructuralFeatureLabel.java
> EStructuralFeatureMultiValueDataSource.java
> EStructuralFeatureMultiValuePropertySource.java
> EStructuralFeatureProperty.java
> EStructuralFeaturePropertyDescriptor.java
> EStructuralFeatureSingleValueDataSource.java
> EStructuralFeatureSingleValuePropertySource.java
> ETypedElementProperty.java
> FooterViewPart.java
> HeaderViewPart.java
> NavigationBarViewPart.java
> NavigationPanelViewPart.java
> ViewAction.java
> ViewActionActivator.java
> ViewActionApplicationBuilder.java
```

ECore

- ViewAction adapters for Ecore model elements
- Documentation generator and helper classes

```
EcoreDocumentationGenerator generator = new EcoreDocumentationGenerator("Nasdanika Bank Model", null);  
generator.loadGenModel("urn:org.nasdanika.bank");  
generator.generate(new File("target/test-dumps/ecore"));
```



Nasdanika Bank Model

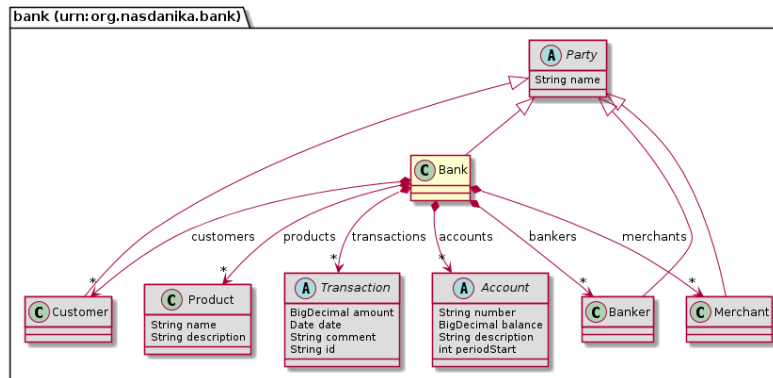
- Bank
 - Account
 - Bank
 - Accounts
 - Bankers
 - Customers
 - Merchants
 - Products
 - Transactions
 - Banker
 - Banker transaction
 - Card
 - Contact method
 - Customer
 - Customer account
 - Device
 - Device transaction
 - E mail
 - Internal account
 - Merchant
 - Mobile phone
 - Online session
 - Online transaction
 - Party
 - Phone
 - Point of sale
 - Postal address
 - Product
 - Statement

Bank / Bank

Bank

Bank is the root of the domain model. Bank has customers and bankers - employees performing banking operations. Bank tracks merchants its customers interact with in order to provide customers information about their spending habits.

Diagram Contents Supertypes



- org.nasdanika.html.ecore
 - DependencyTracer.java
 - EAttributeViewAction.java
 - EClassifierViewAction.java
 - EClassViewAction.java
 - EcoreDocumentationApplication.java
 - EcoreDocumentationGenerator.java
 - EcoreDocumentationViewGenerator.java
 - EcoreViewActionAdapterFactory.java
 - EDataTypeViewAction.java
 - EEnumLiteralViewAction.java
 - EEnumViewAction.java
 - ENamedElementViewAction.java
 - EOperationViewAction.java
 - EPackageViewAction.java
 - EParameterViewAction.java
 - EReferenceViewAction.java
 - EStructuralFeatureViewAction.java
 - ETypedElementViewAction.java
 - GenModelResourceSet.java
 - PlantUmlTextGenerator.java
 - content-router.js