

Poseidon for DSLs User Guide

1. Poseidon for DSLs User Guide	1
1. Overview.....	2
2. Getting Started	3
3. Poseidon Models Guide	3
1. Diagram Model	3
1. file_extension.....	4
2. diagram_types	4
3. node	4
1. node attributes	5
1. node.metamodel_element	5
2. node.default_name.....	5
3. node.default_size	6
4. node.minimum_size.....	6
5. node.shape.....	6
6. node.name_position	7
7. node.keep_proportions.....	7
8. node.icon	7
9. node.metamodel_container.....	8
10. node.allowed_diagram_types	8
11. node.forbidden_diagram_types	9
12. node.creation_mode	9
13. node.can_contain	9
2. node properties.....	10
1. property Color.....	10
2. property FillColor.....	11
3. compartments	11
4. edge	11
1. edge attributes	12
1. edge.metamodel_element	12
2. edge.allowed_diagram_types	12
3. edge.forbidden_diagram_types	12
4. edge.default_name.....	12
5. edge.icon	13
6. edge.at_source_draw	13
7. edge.at_target_draw	13
8. edge.line_style.....	13
5. property	14
6. datatype.....	14
7. role.....	15
2. Tools model	15
1. category	15
1. category.icon.....	17
2. category.mode.....	17
2. node_tool	18
1. node_tool.diagram_node.....	19

2.	node_tool.icon	19
3.	node_tool.shown_in_add_menu_of	19
4.	node_tool.add_menu_role	20
3.	edge_tool	21
1.	edge_tool.diagram_edge	22
2.	edge_tool.icon	22
3.	edge_tool.shown_in_add_menu_of	23
4.	edge_tool.add_menu_role	23
4.	node_tool_reference	23
5.	edge_tool_reference	24
3.	Rapid Buttons model	24
1.	button	24
4.	Edge Rules Model	25
1.	rules_for	26
5.	Properties Model	26
1.	attribute	27
2.	reference	28
4.	API Reference	34
1.	Menu	34
1.	Adding new menu item to main menu.	34
2.	Adding new sub-item to existing main menu item	36
3.	"About" menu item	37
2.	Toolbar	38
1.	Customizing toolbar	38
2.	Creating tool on a toolbar	39
3.	Model Browser	40
1.	Icons in the Model Browser	40
2.	Text in the Model Browser	43
3.	Context menu in the Model Browser	44
4.	Removing nodes from Model Browser	45
4.	Diagram	46
1.	Changing the shape of a node	46
2.	Customizing the appearance of nodes and edges	48
3.	Adding new diagram types	51
5.	Icons	53
6.	Localization	54
1.	Menus	54
2.	Model Browser	55
3.	Tools palette	56

Overview

Poseidon for DSLs is a framework for developing a graphical modeling language for your domain (DSL - Domain Specific Language). To develop your own graphical modeling language with Poseidon for DSLs, you need our distribution of Eclipse and a "template" workspace to start from. The "template" workspace contains Poseidon for DSLs runtime, a set of [XText](#) models (this is where you define nodes, edges, tools etc. for you graphical modeling language) and a set of classes which can be used as a API for more detailed customization of your editor.

Getting Started

Please see this [tutorial](#) to get started with Poseidon for DSLs. It explains how to set up the environment and create a simple graphical modeling language with Poseidon for DSLs.

Poseidon Models Guide

Poseidon editor code is generated from a set of textual DSLs. There are 5 textual models, each with its' own grammar. The models can be found in project "**Poseidon**", folder "**models**"

1. Diagram model (PoseidonDiagramModel.dgm). Contains definition of diagram nodes, edges and their properties.
2. Tools model (PoseidonTools.tools). Contains definition of Poseidon tools on the Palette and in the context menu on the diagram.
3. Rapid Buttons model (Buttons.rbtn). Contains definition of rapid buttons for diagram nodes
4. Edge Rules model (EdgeRules.erule). Contains definition of some rules for edges.
5. Properties model (Properties.prt). Contains rules for properties of the semantic elements of the editor

These models are your tool to create and customize graphical editor for your domain. The "models" folder also contains model "EcoreMetamodelMapping.map", but you are not supposed to edit it because it is generated from the metamodel of your editor. The models are created with [Xtext](#) and are very convenient to work. There is syntax highlighting, code completion and validation. To invoke code completion, use Ctrl-Space key combination.

Diagram Model

Diagram Model PoseidonDiagramModel.dgm can be found in project "Poseidon", folder "models" (see how this model is used in [tutorial](#)). It contains the definition of diagram nodes, edges and their properties. Working with this model, you can:

- define nodes for you graphical editor;
- define edges
- define container nodes, which can contain other nodes;
- define compartments inside the node
- define default and minimum sizes for nodes;
- define default name for elements;
- define a set of user properties for elements (icon, fill color, line color, etc...);
- ...

By default, the diagram model contains some imports, data types and property definitions. Please do not delete them. Next goes the list of definitions you can create in the diagram model

file_extension

Syntax	<code>file_extension: "<your_desired_editor_file_extension>"</code>
Description	Define the extension of project files for your editor
Example	<code>file_extension: "p4pn"</code> Could be an extension for "Poseidon for Petri Net" files

diagram_types

Syntax	<code>diagram_types: "diagram_type_name_1" "diagram_type_name_2" ...</code>
Description	Define the type names of the diagrams which will be used in the editor. Poseidon doesn't generate any actions or menu items (except for the default diagram) to create the desired diagram types. So, if you want to add several diagram types in your editor, define the names of desired diagram types in the diagram model, generate the code and then follow instructions on how to add Poseidon menu items to create your diagrams
Example	<code>diagram_types: "class" "activity" "deployment"</code> Could be names of the diagram types for UML editor.

node

Syntax	<code>node <name_of_the_node> { <node attribute 1> ... <node attribute n> <property 1> ... }</code>
---------------	---

	<pre> <property_n> <compartments> } </pre>
Description	Define a node for the graphical editor
Example	<pre> node Place { metamodel_element: Place default_size: 4 * 4 minimum_size: 3 * 3 can_contain: Token } </pre> <p>Definition of a "Place" node with semantic element "Place", default and minimum sizes. The node can contain other nodes - Tokens.</p>

node attributes

node.metamodel_element

Syntax	metamodel_element: <semantic_element_name_from_mapping_model>
Description	Define the semantic element for the node. You don't have to guess the name of the semantic element, as there will be code completion (invoked by Ctrl-Space). It will give you the list of the semantic elements from your metamodel. Usage of this attribute is recommended, because in this case the code which creates the semantic element, will be generated. If you do not specify "metamodel_element" attribute, you will have to write an implementation of the method creating the semantic element yourself. Which is not difficult, but why bother if you can generate it?
Example	<pre> node Place { metamodel_element: Place } </pre> <p>Definition of a "Place" node with semantic element "Place"</p>

node.default_name

Syntax	default_name: "<your_desired_default_name>"
---------------	---

Description	Define the default name of the node. If this attribute is not specified, the name of the node is used as a default name.
Example	<pre>node Place { default_name: "A Place" }</pre>

node.default_size

Syntax	default_size: <width>*<height>
Description	Define the default size of the node. Size is defined in terms of the grid size of the Poseidon editor.
Example	<pre>node Place { default_size: 3*3 }</pre>

node.minimum_size

Syntax	minimum_size: <width>*<height>
Description	Define the minimum size of the node. Size is defined in terms of the grid size of the Poseidon editor.
Example	<pre>node Place { minimum_size: 2*2 }</pre>

node.shape

Syntax	shape: <shape_name_from_predefined_set_of_shapes>
Description	Define the shape for the node. There is a set of shapes you can choose from. You Ctrl-Space to see the choices
Example	<pre>node Place { shape: RHOMBUS }</pre>

node.name_position

Syntax	name_position: <name_position>
Description	Define the position of the name of the node. Possible choices are: <ul style="list-style-type: none">• ABOVE (above the node)• INSIDE (inside the node)• UNDER (under the node)• NO_NAME (name is not shown on the diagram) If this attribute is not specified, the name is placed under the node by default.
Example	<pre>node Place { name_position: ABOVE }</pre>

node.keep_proportions

Syntax	keep_proportions: true false
Description	Defines if the node has to keep its' original proportions when resized. The default is false.
Example	<pre>node Place { keep_proportions: true }</pre> See how this attribute is used in the tutorial

node.icon

Syntax	icon: "<image_name>"
Description	Defines the icon for the node. This icon will be used in Model Browser, Tools Palette, context menu. Poseidon will search the icon with the specified name in 2 locations, both in project "Resources": <ol style="list-style-type: none">1. folder "icons" (this is where you should put your images)2. library "lib/Resources.jar" (already contains a set of images you can use) IMPORTANT: only specify the name of the image, do not specify the file extension (see example)

Example	<pre>node Place { icon: "artifact" }</pre> <p>In this example the file "artifact.png" is resided in "lib/Resources.jar"</p>
----------------	---

node.metamodel_container

Syntax	metamodel_container: <ecore_reference_name_from_mapping_model>
Description	Has a meaning only for nodes which are used as compartments (see how compartments are used in tutorial). You don't have to guess the name of the reference, as there will be code completion (invoked by Ctrl-Space). It will give you the list of the available references from your metamodel. Knowing the name of the reference, poseidon generator will generate the code which will store Tokens in the "tokens" reference of the Place.
Example	<pre>node Token { metamodel_element: Token metamodel_container: Place_tokens }</pre>

node.allowed_diagram_types

Syntax	allowed_diagram_types: [< diagram_type >]
Description	The list of diagram types where it is allowed to create this node. This attribute could be only necessary if you specify any diagram_types in the model. By default, the node can be created on any diagram.
Example	<pre>diagram_types: "petri" "bpmn" node Place { metamodel_element: Place allowed_diagram_types: petri }</pre>

node.forbidden_diagram_types

Syntax	forbidden_diagram_types: [< diagram_type >]
Description	The list of diagram types where it is forbidden to create this node. This attribute could be only necessary if you specify any diagram_types in the model. By default, the node can be created on any diagram.
Example	<pre>diagram_types: "petri" "bpmn" node Place { metamodel_element: Place forbidden_diagram_types: bpmn }</pre>

node.creation_mode

Syntax	creation_mode: CLICK_CREATION DRAG_CREATION
Description	There are two possible creation modes. CLICK_CREATION and DRAG_CREATION. If you select CLICK_CREATION - the node will be created at the point on the diagram where you click the mouse, with the default size. Then you can resize the node. If you select DRAG_CREATION - you can define the size of the element in the moment of creation. You click the mouse on the diagram and then drag the mouse without releasing the mouse button. Poseidon shows the preview of the node to be created. When you are happy with the preview, you release the mouse button and Poseidon creates a node of a size defined by you via click-and-drag. The default creation mode is CLICK_CREATION.
Example	<pre>node Place { creation_mode: DRAG_CREATION }</pre>

node.can_contain

Syntax	can_contain: [< name_of_the_node name_of_the_role >]
Description	Defines the list of nodes which can be created or dragged inside the given node.
Example	<pre>node Place { can_contain: Token Place }</pre>

```
node Token {  
}
```

Such a definition means that node "Place" can contain other Places and Tokens inside, i.e you can create a node for a Token or a Place inside the Place. Or, you can drag existing Token or Place inside another Place

node properties

Syntax	property <name_of_the_property> : <value_of_the_property>
Description	Defines the value of the property for the node.
Example	<pre>node Place{ property FillColor : DARK_GRAY }</pre> <p>Such a definition means that the default fill color of the node "Place" will be dark gray.</p>

property Color

Property "Color" can be used for any node or edge. It defines the default line color for the node (or edge). There are two ways to specify the value of the color:

1. Select from the list of predefined colors (invoke code completion via ctrl-Space).
Example:

```
node Place{  
  property Color : DARK_GRAY  
}
```

2. Use construct `COLOR_RGB(<R value>,<G value>,<B value>)`. Example:

```
node Place {  
  property Color: COLOR_RGB(28,87,180)  
}
```

property FillColor

Property "FillColor" can be used for any node or edge. It defines the default fill color for the node (or edge). "FillColor" property is specified same as [property "Color"](#).

compartments

See how [compartments are used](#) in tutorial.

Syntax	compartments: [name of the node]
Description	Defines the list of compartments for the node. The nodes from the compartments list must have the metamodel_container attribute specified.
Example	<pre>node Place{ metamodel_element: Place compartments: Token } node Token { metamodel_element: Token metamodel_container: Place_tokens }</pre>

edge

Syntax	<pre>edge <name_of_the_edge> { sources: [name of the node name of the role] targets: [name of the node name of the role] <edge attribute 1> ... <edge attribute n> <property 1> ... <property n> }</pre>
Description	Define an edge for the graphical editor.
Example	<pre>edge Arc { sources: Place Transition targets: Place Transition }</pre>

Definition of an edge "Arc". It states that possible sources of that edge are nodes "Place" and "Transition". Possible targets are the same. When you generate the code and start Poseidon - it will only allow you to create an edge "Arc" if the source and the target of the edge are either Place or Transition. It will not allow to create an edge from (for example) a Token to a Place

edge.attributes

edge.metamodel_element

Same as [node.metamodel_element](#).

edge.allowed_diagram_types

Same as [node.allowed_diagram_types](#)

edge.forbidden_diagram_types

Same as [node.forbidden_diagram_types](#)

edge.default_name

Same as [node.default_name](#)

edge.icon

Same as [node.icon](#)

edge.at_source_draw

Syntax	at_source_draw: NOTHING OPEN_ARROW CLOSED_ARROW
Description	Defines the style of the source end of the edge. Depending on your choice at the beginning of the edge an open arrow, closed arrow or nothing will be drawn. The default value is NOTHING
Example	<pre>edge Arc { sources: Place Transition targets: Place Transition at_source_draw: NOTHING }</pre>

edge.at_target_draw

Syntax	at_target_draw: NOTHING OPEN_ARROW CLOSED_ARROW
Description	Defines the style of the target end of the edge. Depending on your choice at the end of the edge an open arrow, closed arrow or nothing will be drawn. The default value is NOTHING
Example	<pre>edge Arc { sources: Place Transition targets: Place Transition at_target_draw: CLOSED_ARROW }</pre>

edge.line_style

Syntax	line_style: SOLID DASHED
Description	Defines the appearance style of the edge. Solid line and dashed line options are available. The default value is SOLID

Example	<pre>edge Arc { sources: Place Transition targets: Place Transition line_style: SOLID }</pre>
----------------	---

property

Syntax	<pre>property <name_of_the_property> { type: datatype default_value: "<default_value_of_the_property>" variable_name: "<variable_name_for_that_property>" }</pre>
Description	Define a property which you want to store in your project file.
Example	<pre>property FillColor { type: Color default_value: COLOR_RGB(255,255,255) variable_name: "fill" }</pre> <p>This is a definition from default diagram model which is actually used by Poseidon. In Poseidon editor, you can define the fill color for an element and this property is used to store the value of the color in the project file.</p>

datatype

Syntax	<pre>datatype <datatype_name> mappedto "<java_class_fully_qualified_name>"</pre>
Description	Defines a datatype which can be used in property definition .
Example	<pre>datatype Color mappedto "java.awt.Color"</pre>

role

Syntax	<pre>role <role_name> { nodes: [<name_of_the_node>] }</pre>
Description	Defines a role, which is actually a group of nodes which can be then referenced by a role name.
Example	<p>You can define an edge Arc with sources Place and Transition like this (roles are not used):</p> <pre>edge Arc { sources: Place Transition targets: Place Transition }</pre> <p>Or like this (roles are used):</p> <pre>edge Arc { sources: ArcSources targets: Place Transition }</pre> <pre>role ArcSources{ nodes: Place Transition }</pre>

Tools model

Tools Model PoseidonTools.tools can be found in project "Poseidon", folder "models". It contains the definition of tools for creation of elements in Poseidon. Working with this model, you can:

- define categories (groups) of tools on the Poseidon tool palette
- define tools on the Poseidon tool palette

By default, the tools model contains only one import. It imports the diagram model, because tools model often has references to some elements from diagram model.

category

Tools model contains definitions of tool categories (groups). Tools are defined inside those categories, i.e every tool belongs to a category.

Syntax	<pre> category <name_of_the_category> { icon: "<image_name>" mode: maximized minimized <node_tool_1> ... <node_tool_n> <edge_tool_1> ... <edge_tool_n> <node_tool_reference_1> ... <node_tool_reference_n> <edge_tool_reference_1> ... <edge_tool_reference_n> } </pre>
Description	<p>Defines a category on the Tools Palette.</p> <p>In the generated code, project "Poseidon", there will be a property file <code>DslGenMainPaletteResourceBundle.properties</code> in package <code>com/gentleware/poseidon/gui/palette/main/impl</code>, which is responsible for screen names in the Tools Palette. This file is generated if it doesn't yet exist, but it is not overwritten if it already exists. So, you can change the screen name of the desired item. To set your own name for the category, add the following line to that property file:</p> <pre><name_of_the_category>Category = <desired_category_screen_name></pre> <p>For example, if you defined category "Petri" in the tools model</p> <pre>category Petri { }</pre> <p>you should add the following line to the properties file:</p> <pre>PetriCategory = Petri Net Tools</pre> <p>and the screen name of this category in Poseidon UI will be "Petri Net Tools"</p>
Example	<pre> category Petri { icon: "petri_diagrams_icon" mode: maximized node_tool Place{ diagram_node: Place } edge_tool Arc{ diagram_edge: Arc } } </pre>

category.icon

Syntax	icon: "<image_name>"
Description	<p>Defines the icon for the category. This icon will be used in the tools Palette. Poseidon will search the icon with the specified name in 2 locations, both in project "Resources":</p> <ol style="list-style-type: none">1. folder "icons" (this is where you should put your images)2. library "lib/Resources.jar" (already contains a set of images you can use) <p>IMPORTANT: only specify the name of the image, do not specify the file extension (see example)</p>
Example	<pre>category Petri { icon: "petri_diagrams_icon" }</pre> <p>In this example you have to put icon "petri_diagrams_icon.png" to folder "icons".</p>

category.mode

Syntax	mode: maximized minimized
Description	Defines if this category should be expanded or collapsed in the Tools Palette by default. The default value is <code>maximized</code> .
Example	<pre>category Petri { mode: maximized }</pre>

node_tool

Syntax	<pre>node_tool <name_of_the_tool> { diagram_node icon shown in add menu of add menu role }</pre>
Description	<p>Defines a tool on the Tool Palette. This tool creates on the diagram the node which you specified in the diagram_node attribute. You don't have to keep in memory the names of the nodes in your diagram model, the list of the available nodes will be provided to you if you press Ctrl-Space.</p> <p>In the generated code, project "Poseidon", there will be a property file <code>DslGenMainPaletteResourceBundle.properties</code> in package <code>com/gentleware/poseidon/gui/palette/main/impl</code>, which is responsible for screen names in the Tools Palette. This file is generated if it doesn't yet exist, but it is not overwritten if it already exists. So, you can change the screen name of the desired item. To set your own name for the tool, add the following line to that property file:</p> <pre><tool_name_in_the_model> = <desired_tool_screen_name></pre> <p>For example, if you defined tool "Place" in the tools model</p> <pre>node_tool Place { diagram_node: Place }</pre> <p>you should add the following line to the properties file:</p> <pre>Place = A Place</pre> <p>and the screen name of this tool in Poseidon UI will be "A Place"</p>
Example	<pre>node_tool Place{ diagram_node: Place }</pre>

node_tool.diagram_node

Syntax	diagram_node: < reference to diagram node definition >
Description	Defines the diagram node (from diagram model) which this tool has to create on the diagram. You don't have to keep in memory the names of the nodes in your diagram model , the list of the available nodes will be provided to you if you press Ctrl-Space.
Example	<pre>node_tool Place{ diagram_node: Place }</pre>

node_tool.icon

Syntax	icon: "<image_name>"
Description	<p>Defines the icon for the tool in the Tools Palette. Poseidon will search the icon with the specified name in 2 locations, both in project "Resources":</p> <ol style="list-style-type: none">1. folder "icons" (this is where you should put your images)2. library "lib/Resources.jar" (already contains a set of images you can use) <p>IMPORTANT: only specify the name of the image, do not specify the file extension (see example) If this attribute is not specified for the tool, the icon from the diagram node definition (which is referenced from the mandatory diagram_node attribute) will be used.</p>
Example	<pre>node_tool Place{ diagram_node: Place icon: "circle" }</pre> <p>In this example the file "circle.png" is resided in "lib/Resources.jar"</p>

node_tool.shown_in_add_menu_of

See [how this property is used](#) in tutorial.

Syntax	shown_in_add_menu_of: [Diagram element namespace "<existing_add_menu_role>"]
Description	In Poseidon, in the context menu of the diagram, there is a menu item called "add" that allows to add new model elements in the given context. There is also a short cut for this. Just hit the space bar and the add context menu will appear in the position of the mouse. If you invoke it in the empty space of a diagram, it will allow you to create elements in that position. Here is how it looks:



By default this add menu is empty. To add new elements to this add menu, the attribute "shown_in_add_menu_of" is used. For example, if we are designing an editor for Petri nets and we have a node Place, it should be in the add menu of the diagram. In this case we use the following definition for "shown_in_add_menu_of" attribute:

```
shown_in_add_menu_of: Diagram
```

There are 3 predefined choices for that attribute (press Ctrl-Space to see them):

1. Diagram. Means that the given tool will be shown in the add menu of the diagram
2. namespace. Means that the given tool will be shown in the add menu of any namespace node on the diagram
3. element. Means that the given tool will be shown in the add menu of any node on the diagram

```
shown_in_add_menu_of: Diagram namespace
```

You can also add a name of the add menu role which you defined in some other tools' [add_menu_role](#) attribute (see description for [add_menu_role](#))

Example

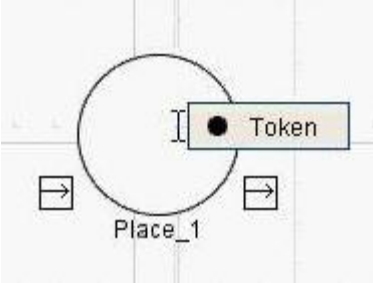
```
node_tool Place {
  diagram_node: Place
  shown_in_add_menu_of: Diagram
}
```

node_tool.add_menu_role

Syntax	add_menu_role: Diagram element namespace "<add_menu_role_name>"
Description	<p>Defines the add menu role for this tool. Say, we are designing an editor for Petri Nets. We have nodes for Places and Tokens. We want the Token to be available only in the add menu of a Place (but not in the add menu of diagram or any other element). To do that, we first add an "add_menu_role" attribute to the definition of the tool for Place:</p> <pre>node_tool Place { diagram_node: Place add_menu_role: "place" }</pre> <p>So, now node Place has an add menu role named "place".</p> <p>Now we add a shown_in_add_menu_of attribute to the definition of the Token tool.</p> <pre>node_tool Token { diagram_node: Token shown_in_add_menu_of: "place"</pre>

```
}
```

So, with such definitions for Place and Token tools, the add menu of Place (and only the add menu of Place) will have an item to add a Token



Example see **Description**

edge_tool

Syntax	<pre>edge_tool <name_of_the_tool> { diagram edge icon shown_in_add_menu_of add_menu_role }</pre>
Description	<p>Defines a tool on the Tool Palette. This tool creates on the diagram the edge which you specified in the diagram edge attribute. You don't have to keep in memory the names of all edges in your diagram model, the list of the available ed will be provided to you if you press Ctrl-Space.</p> <p>In the generated code, project "Poseidon", there will be a property file <code>DslGenMainPaletteResourceBundle.properties</code> in package <code>com/gentleware/poseidon/gui/palette/main/impl</code>, which is responsible for screen names in the Tools Palette. This file is generated if it doesn't yet exist, but it is not overwritten if it already exists. So, you can change the screen name of the desired item. To set your own name for the tool, add the following line to that property file:</p> <pre><tool_name_in_the_model> = <desired_tool_screen_name></pre> <p>For example, if you defined tool "Arc" in the tools model</p> <pre>edge_tool Arc { diagram edge: Arc }</pre>

	<p>you should add the following line to the properties file:</p> <pre>Arc = An Arc</pre> <p>and the screen name of this tool in Poseidon UI will be "An Arc"</p>
Example	<pre>edge_tool Arc { diagram_edge: Arc }</pre>

edge_tool.diagram_edge

Syntax	<pre>diagram_edge: <reference to diagram edge definition></pre>
Description	<p>Defines the diagram edge (from diagram model) which this tool has to create on the diagram. You don't have to keep in memory the names of the edges in your diagram model, the list of the available edges will be provided to you if you press Ctrl-Space.</p>
Example	<pre>edge_tool Arc { diagram_edge: Arc }</pre>

edge_tool.icon

Same as [node_tool.icon](#).

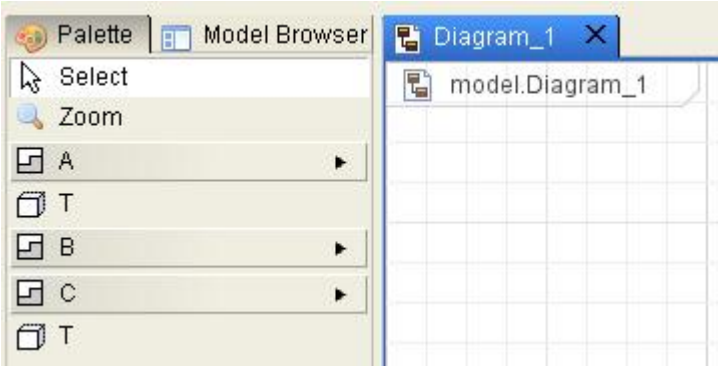
edge_tool.shown_in_add_menu_of

Same as [node_tool.shown_in_add_menu_of](#)

edge_tool.add_menu_role

Same as [node_tool.add_menu_role](#)

node_tool_reference

Syntax	node_tool_reference: < reference to node tool >
Description	Allows to use node tool definitions from another category. See example
Example	<p>Say, your editor has 3 tool categories: A, B and C. You defined tool T in category A. But you also want this tool to be present in category C. In this case, instead of creating exactly the same tool definition in category C, you can use the existing tool definition from category A.</p> <pre>category A { node_tool T { diagram_node: T } } category B { } category C { node_tool_reference T }</pre> <p>And here is how it looks in Poseidon after the code generation:</p> 

Tool T exists in categories A and C, category B is empty
--

edge_tool_reference

Syntax	edge_tool_reference: < reference to edge tool >
Description	Allows to use edge tool definitions from another category.
Example	Usage is the same as for node_tool_reference .

Rapid Buttons model

Rapid buttons model `RButtons.rbtn` can be found in project "Poseidon", folder "models". It contains the definition of rapid buttons for diagram nodes. Rapid buttons are a neat little mechanism that makes editing diagrams faster and more intuitive. Rapid buttons pop up in the diagram next to a node and provide the tools to create edges in the right context. Rapid buttons can appear in 8 different positions around a node, with names like `RIGHT_TOP` or `RIGHT_BOTTOM`.

By default, the rapid buttons model contains only one import. It imports the diagram model, because tools model often has references to some elements from diagram model.

button

Syntax	<pre>button <name_of_the_button> { anchors: [<diagram node definition reference>] edge: <diagram edge definition reference> position: [BOTTOM BOTTOM_LEFT BOTTOM_RIGHT LEFT LEFT_BOTTOM LEFT_TOP RIGHT RIGHT_BOTTOM RIGHT_TOP TOP TOP_LEFT TOP_RIGHT] icon: "<image_name>" direction: INCOMING OUTGOING }</pre>
---------------	--

Description	<p>Defines the rapid button.</p> <ul style="list-style-type: none"> • <code>anchors</code>. List of nodes from the diagram model for which this button should appear • <code>edge</code>. Reference to the definition of an edge (from diagram model) which is created when you click this rapid button. • <code>position</code>. List of positions around the node where this rapid button should appear. Possible choices are <code>BOTTOM</code>, <code>BOTTOM_LEFT</code>, <code>BOTTOM_RIGHT</code>, <code>LEFT</code>, <code>LEFT_BOTTOM</code>, <code>LEFT_TOP</code>, <code>RIGHT</code>, <code>RIGHT_BOTTOM</code>, <code>RIGHT_TOP</code>, <code>TOP</code>, <code>TOP_LEFT</code>, <code>TOP_RIGHT</code> • <code>icon</code>. Icon for the rapid button. Poseidon will search the icon with the specified name in 2 locations, both in project "Resources": folder "icons" (this is where you should put your images) and library "lib/Resources.jar" (already contains a set of images you can use). IMPORTANT: only specify the name of the image, do not specify the file extension (see example) • <code>direction</code>. The direction of the edge. Possible choices are <code>INCOMING</code> and <code>OUTGOING</code>. The default value is <code>OUTGOING</code>.
Example	<pre>button Arc { anchors: Place Transition edge: Arc position: RIGHT_TOP icon: "edge-default" direction: INCOMING }</pre> <p>In this example the file "edge-default.png" resides in "lib/Resources.jar"</p>

Edge Rules Model

Edge Rules model `EdgeRules.erule` can be found in project "Poseidon", folder "models". It contains some rules for edge creation in Poseidon. In Poseidon, you can create an edge by starting it on an existing diagram node and dropping it on an empty space of the diagram. In this case a new node will be created and used as a target for the edge. By default, Poseidon will create the same type of node as the source node. For example, if you start dragging an Arc edge from a Place node and drop it on an empty place of the diagram, Poseidon will create another Place and connect the Arc to it. If the Place is not listed in the list of this edges' [targets](#), Poseidon will create the first node type from the [targets](#) list. Sometimes you are not happy with this default behavior. You can control it with the Edge Rules model.

By default, the edge rules model contains only one import. It imports the diagram model, because tools model often has references to some elements from diagram model.

rules_for

Syntax	<pre>rules_for <reference to edge definition> { from <reference to node definition> create <reference to node definition> from <reference to node definition> create <reference to node definition> ... }</pre>
Description	Defines the rules for the specified edge. A rule "from NodeA create NodeB" means that if you start creating the specified edge from NodeA and release the mouse on an empty space of the diagram, an instance of NodeB will be created on the target end of the edge.
Example	<p>We design an editor for Petri Nets. We have nodes Place and Transition and an edge Arc. Possible sources for the Arc are Place and Transition. Possible targets are Place and Transition as well. But in Petri Nets you never connect a Place to another Place or a Transition to another Transition. The Arc always goes from Place to Transition or from Transition to Place. So, to make our Petri Net editor create correct target of the Arc, we should add following rule definition:</p> <pre>rules_for Arc { from Place create Transition from Transition create Place }</pre> <p>This example is taken from tutorial. See more detailed description in the tutorial.</p>

Properties Model

Properties Model Properties.prt can be found in project "Poseidon", folder "models". This model allows you to configure visibility and changeability of semantic element attributes and references in the Model Browser.

By default, the properties model contains only one import. It imports the diagram model, because properties model has to reference some elements from diagram model.

attribute

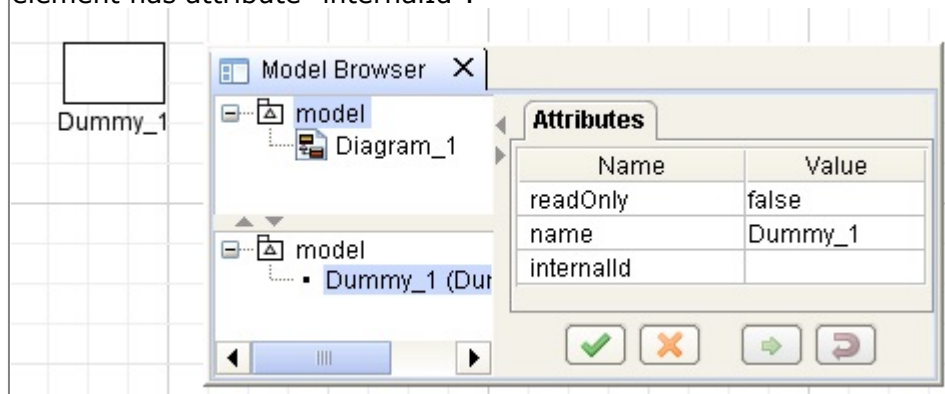
Syntax

```
attribute {
  name: "<name_of_the_attribute>"
  hidden_elements: ALL | [<reference to node or edge definition>]
  included_elements: [<reference to node or edge definition>]
  read_only_elements: ALL |
  [<reference to node or edge definition>]
}
```

Description By default, Poseidon shows all attributes of the element in the Model Browser.

hidden_elements

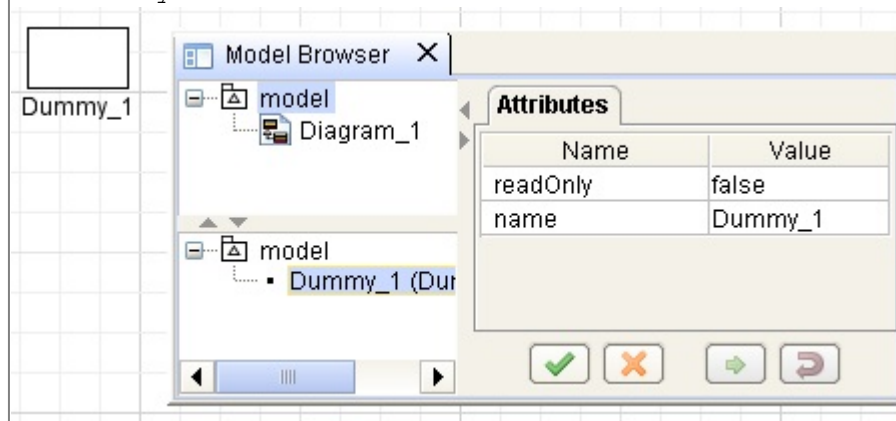
For example, you have an element "Dummy" in your metamodel and this element has attribute "internalId".



Suppose, you are using this `internalId` somehow internally in the code, but you do not want to show this attribute to users. In this case you can hide it, using "hidden_element" attribute in the properties model:

```
attribute {
  name: "internalId"
  hidden_elements: Dummy
}
```

This definition "tells" Poseidon to hide attribute "internalId" for "Dummy" node.



You can also specify a list of elements (nodes and edges) for which the attribute has to be hidden:

```
attribute {  
  name: "internalId"  
  hidden_elements: Dummy Place Token  
}
```

Or, you can use "ALL" keyword if you want to hide "internalId" for all elements:

```
attribute {  
  name: "internalId"  
  hidden_elements: ALL  
}
```

included_elements

Has the same meaning as "**hidden_elements**", but vice versa. I.e instead of specifying the list of elements for which the "internalId" must be hidden you specify the list of elements for which it must be shown. Only one of "**hidden_elements**" or "**included_elements**" attributes can be used simultaneously. Use the one which is more convenient (i.e the one with the shorter list of elements).

read_only_elements

Allows to make an attribute read-only. For example, you want to show the "internalId" to users, but you don't want the users to edit it. You can configure it as a read-only attribute:

```
attribute {  
  name: "internalId"  
  read_only_elements: Dummy  
}
```

In this case the "internalId" attribute will be visible in the Model Browser, but it will not be editable.

Example	See Description.
----------------	------------------

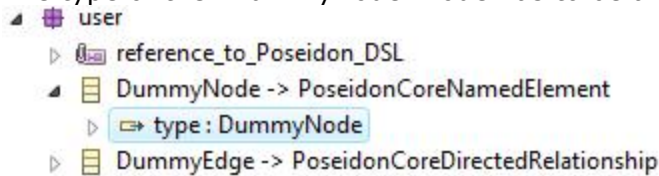
reference

Syntax

```
reference {  
  name: "<name_of_the_reference>"  
  included_elements: [<reference_to_node_or_edge_definition>]  
  read_only_elements: [<reference_to_node_or_edge_definition>]  
}
```

Description

EReferences are not shown in Poseidon Model Browser by default. So, if one of your semantic model elements has a reference and you want that reference to be editable in the Model Browser, you have to start from changing the Poseidon properties model. For example, in your metamodel you have a "DummyNode" node which has a reference "type". The type of the "DummyNode" node has to be another "DummyNode" node:



To make this reference visible in the Model Browser, add this definition in Properties model:

```
reference {
  name: "type"
  included_elements: Dummy
}
```

It "tells" Poseidon to show the "type" reference for node "Dummy". But Poseidon doesn't know how to handle this reference, so you have to implement your own viewer for that reference. After you add the lines above in the Properties model and re-generate Poseidon code, it will not be compilable:

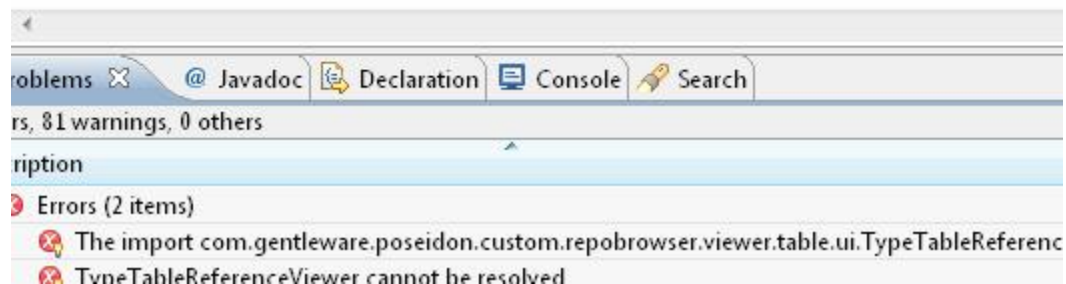
```
package com.gentleware.poseidon.repobrowser.viewer.table;

import com.gentleware.poseidon.custom.repobrowser.viewer.table.ui.

/*
 * This code is generated. All the hand-written changes will be lo
 */

public class DslGenTableViewFactoryImpl extends TableViewFacto

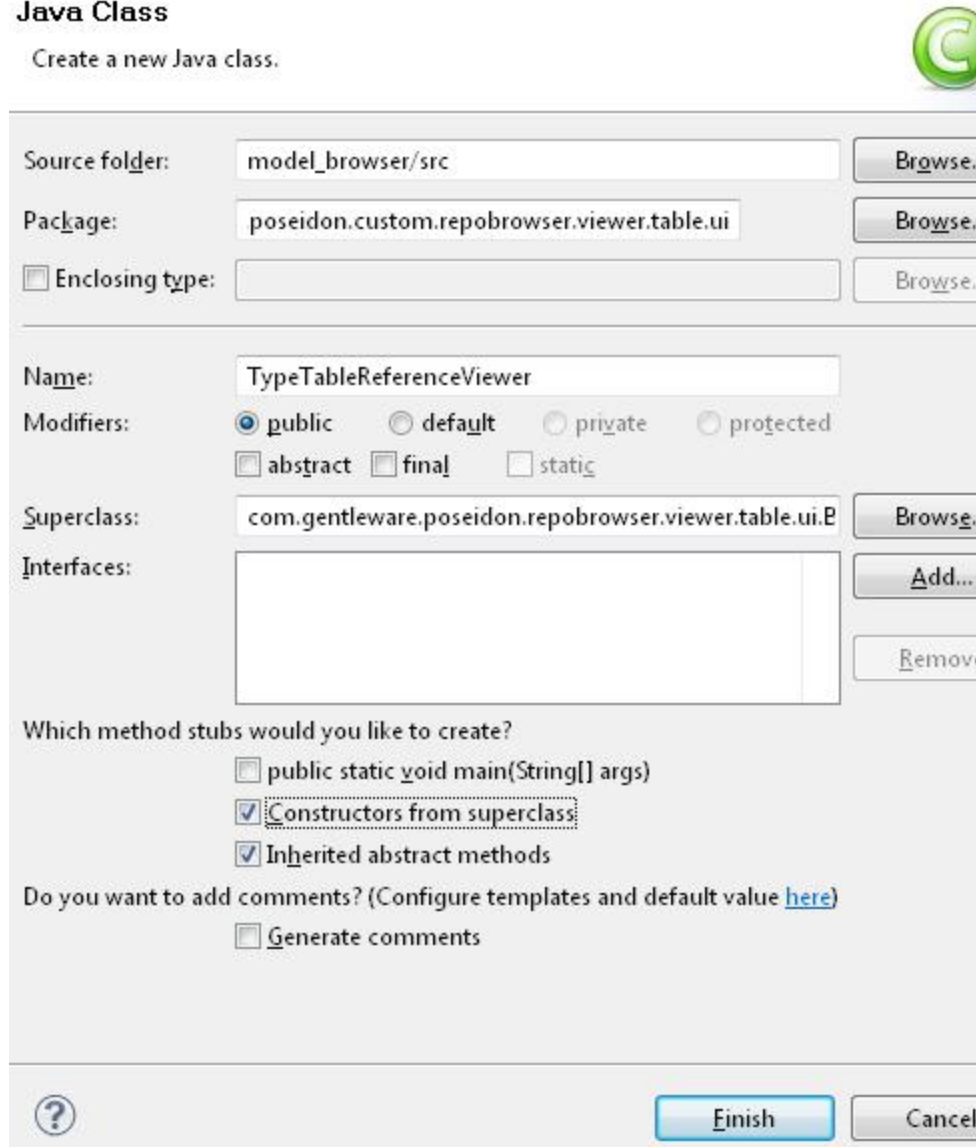
    @Override
    protected void initReferencesViewers() {
        referencesViewers.put("type", TypeTableReferenceViewer);
    }
}
```



You have to implement this missing class "TypeTableReferenceViewer" yourself. So, create a new class TypeTableReferenceViewer in project "model_browser", package com.gentleware.poseidon.custom.repobrowser.viewer.table.ui. Inherit it from class com.gentleware.poseidon.repobrowser.viewer.table.ui.BaseTableReferenceViewer. Please also check the "Constructors from superclass" item in the wizard:

Java Class

Create a new Java class.



Source folder: model_browser/src

Package: poseidon.custom.repobrowser.viewer.table.ui

Enclosing type:

Name: TypeTableReferenceViewer

Modifiers: public default private protected
 abstract final static

Superclass: com.gentleware.poseidon.repobrowser.viewer.table.ui.B

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

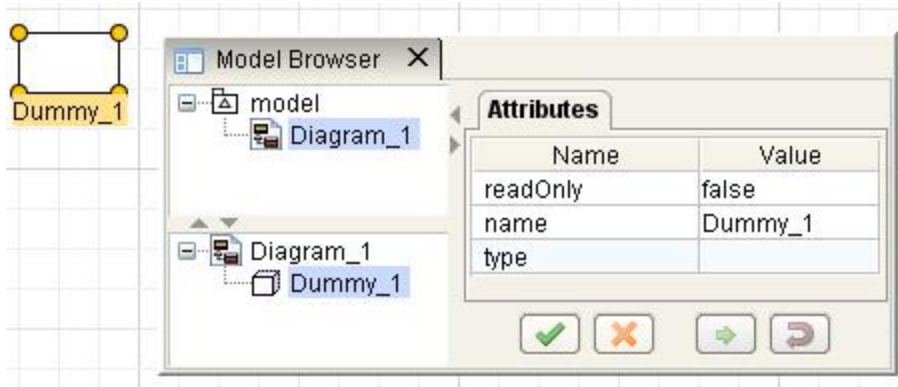
Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Finish Cancel

Once you created this class, the code becomes compilable and you can start Poseidon. You will see that there is a viewer for reference "type". But you cannot edit it so far.



To fully implement the viewer for the reference, you have to work on this `TypeTableReferenceViewer` class and implement some methods. Below is the body of `TypeTableReferenceViewer` class with implemented methods and comments with explanations:

```
// default constructor
public TypeTableReferenceViewer(PoseidonCoreElement element,
    EStructuralFeature attribute,
    DSLAttributeModificationListener listener, boolean readOnly) {
    super(element, attribute, listener, readOnly);
}

// this method creates the swing component which is used as an editor for
// your reference. We need a combo-box.
@Override
public JComponent createEditor(String advancedModelValue) {
    return new JComboBox();
}

// if you use a combo-box as an editor, than this method is responsible for
// filling up the list of possible values
@Override
protected void initListModel(JComboBox comboBox) {
    final DefaultComboBoxModel model = (DefaultComboBoxModel) comboBox
        .getModel();
    model.removeAllElements();
    final List<EModelElement> availableTypes = new ArrayList<EModelElement>();
    availableTypes.add(null);
    final SubjectRepositoryFacet repository=GlobalSubjectRepository.repository;
    if (element instanceof DummyNode) {
        final List<PoseidonCoreElement> dummyNodesFromModel = repository
            .findAllElementsOfType(DummyNode.class, false);
        availableTypes.addAll(dummyNodesFromModel);
    }

    // apply sorting
    Collections.sort(availableTypes, new NameElementComparator());
    for (EModelElement type : availableTypes) {
        model.addElement(type);
    }
}
```

```

final Object modelValue = getModelValue();
model.setSelectedItem(modelValue);
}

// this method installs the editor into the attribute panel
@Override
public JComponent installAttributeEditor(
    final ToolCoordinatorFacet coordinator,
    final JPanel insetPanel, GridBagConstraints gbcLeft,
    GridBagConstraints gbcRight) {
    // call super-method to do all the work
    JComponent referenceEditor = super.installAttributeEditor(
        coordinator, insetPanel, gbcLeft, gbcRight);

    // add the renderer we want
    ((JComboBox) getEditor())
        .setRenderer(new NamedElementComboBoxRenderer());
    return referenceEditor;
}

// this method is responsible for getting the current value of the reference
// ("type" in our example) from the model
@Override
protected Object getModelValue() {
    DummyNode type = null;
    if (element instanceof DummyNode) {
        DummyNode dummyNode = (DummyNode) element;
        type = dummyNode.undeleted_getType();
    }
    return type;
}

// this method returns the command which is executed when you apply the
// changes
@Override
public com.gentleware.poseidon.idraw.foundation.Command formApplyCommand() {
    if (!isModified())
        return null;
    return new com.gentleware.poseidon.idraw.foundation.AbstractCommand(
        "set new type", "reverted to previous type") {
        private static final long serialVersionUID = 1L;
        private final DummyNode oldValue = (DummyNode) getModelValue();
        private final DummyNode newValue = (DummyNode) getCurrentValue();

        // this method is executed when you apply changes
        public void execute(boolean isTop) {
            setNewType((DummyNode) element, newValue);
        }

        // this method is executed when you undo your changes via Ctrl-Z or
        // from the main menu
        public void unExecute() {
            setNewType((DummyNode) element, oldValue);
        }
    };
}

```

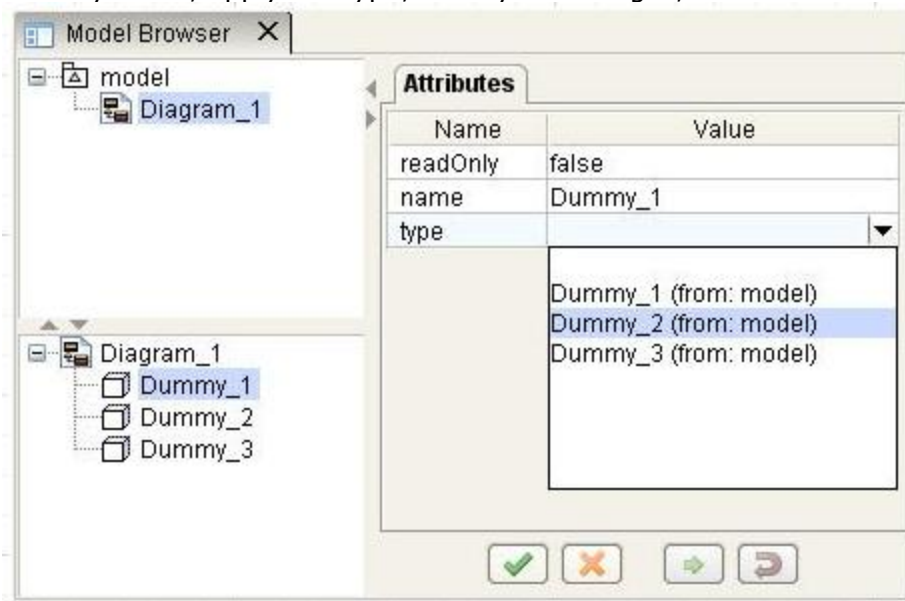
```

    }

    private void setNewType(DummyNode typedElement, DummyNode type) {
        typedElement.setType(type);
    }
};
}

```

Now the editor of the "type" reference allows you to select the type from all existing DummyNodes, apply the type, undo your changes, etc...

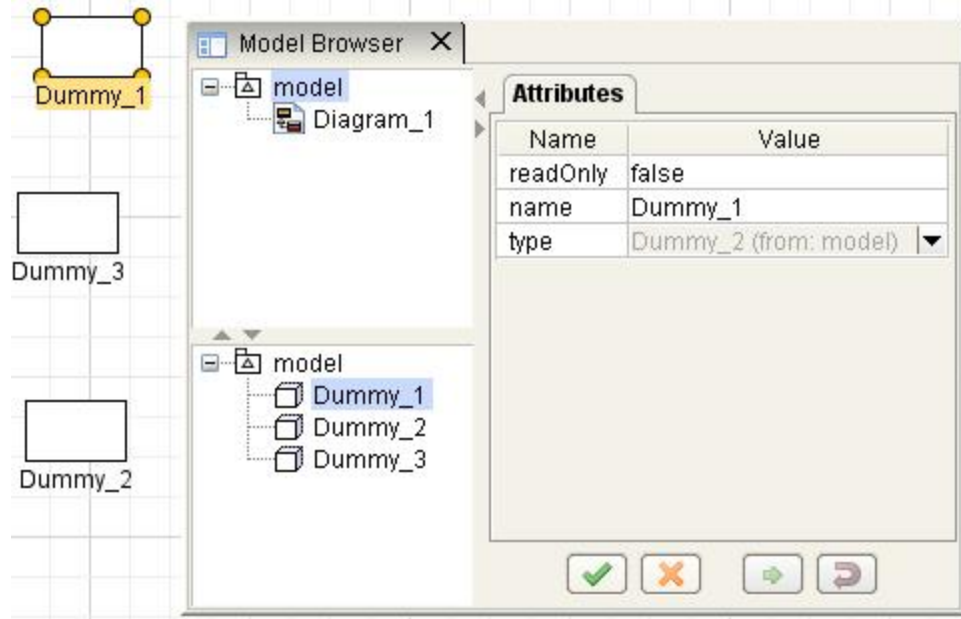


The "read_only_elements" attribute in the Properties model defines the list of diagram elements for which this reference is read-only. Example:

```

reference {
    name: "type"
    included_elements: Dummy
    read_only_elements: Dummy
}

```



API Reference

Poseidon for DSLs API allows you to extend and customize the editor according to your needs.

Menu

Adding new menu item to main menu.

Poseidon menu contains several default items: File Edit View Create Help



Poseidon API allows you to define your own menu items as well. Class `com.gentleware.poseidon.custom.gui.menu.CustomMainMenu` is responsible for that. Let's say you want to add menu "My Tools" after the "Create" menu item. Edit `CustomMainMenu` class and override method `setupMenus()`. Call the super method (which will create the default menu items), then add the items you need:

```
@Override
protected void setupMenus() {
    super.setupMenus();
    ResourceId myMenuItemResourceId =
CustomMainMenuResourceBundle.MyTools;
    factory.addMenu(myMenuItemResourceId, "e");
}
```

You also have to add resource id for your new menu item in class `com.gentleware.poseidon.custom.gui.menu.CustomMainMenuResourceBundle`. Open that class and add following line there:

```
public static ResourceId MyTools = new ResourceId(MAIN_MENU_BUNDLE,
"MyTools");
```

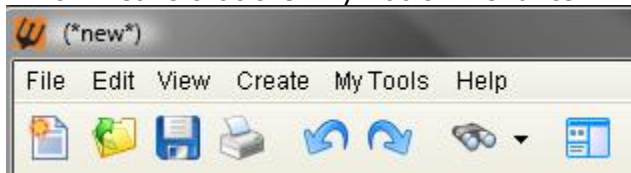
Next to `CustomMainMenuResourceBundle` class you will find `CustomMainMenuResourceBundle.properties` file. Add the localized string value for your new menu item there:

```
MyTools = My Tools
```

NOTE: the order of menu items in Poseidon main menu is defined by alphabetical order of letters assigned to each menu item. Default Poseidon menu items use letters from "a" to "d" and letter "z" for the "Help" menu item. In this example letter "e" is used for the new menu item

```
factory.addMenu(myMenuItemResourceId, "e");
```

which means that the "My Tools" menu item will be placed after "Create" menu item:



Adding new sub-item to existing main menu item

Say, you want to add your own action "Create foo" to Poseidon's "Create" menu. Go to class `com.gentleware.poseidon.custom.gui.menu.CustomMainMenu` and override method `setupMenus()` to make it create a new menu item with resource id `CustomMainMenuResourceBundle.CreateFoo` to the "Create" menu:

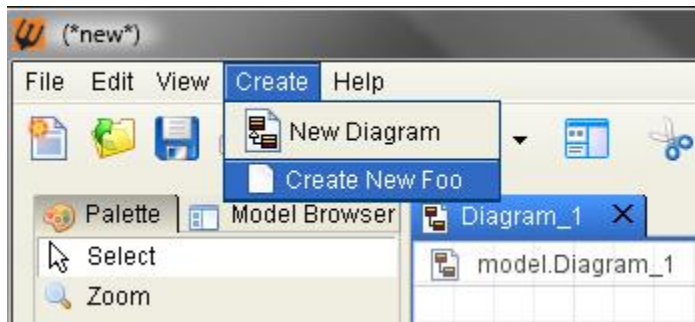
```
@Override
protected void setupMenus() {
    super.setupMenus();
    JMenuItem createFooItem =
factory.addItem(MainMenuResourceBundle.Create,
    MainMenuResourceBundle.Create,
CustomMainMenuResourceBundle.CreateFoo, "b");
}
```

Of course, you also have to create the resource id we are using. Add following line to class `com.gentleware.poseidon.custom.gui.menu.CustomMainMenuResourceBundle`

```
public static ResourceId CreateFoo = new ResourceId(MAIN_MENU_BUNDLE,
"CreateFoo");
```

and the localized string value to `CustomMainMenuResourceBundle.properties`

```
CreateFoo = Create New Foo
```



You can easily set an icon and an action listener for the new menu item you created. Here is an example:

```
@Override
protected void setupMenus() {
    super.setupMenus();
    JMenuItem createFooItem =
factory.addItem(MainMenuResourceBundle.Create,
    MainMenuResourceBundle.Create,
CustomMainMenuResourceBundle.CreateFoo, "b");
    createFooItem.setIcon(com.gentleware.poseidon.util.IconLoader
        .getIconLoader().get("foo-icon"));
    createFooItem.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {
            // create foo
        }
    });
}
```

```
    }  
  
    });  
}
```

NOTE: it is assumed that image "foo-icon.png" exists in folder "icons" of "resources" project.

"About" menu item

By default, Poseidon for DSLs shows the following "About" window.



Class responsible for the "About" window is

`com.gentleware.poseidon.custom.listeners.CustomAboutPoseidonActionListener`. If you only want to change the text and the image, you can do that by overriding methods `getAboutText` and `getAboutIcon`

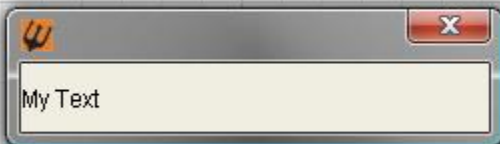
```
@Override  
protected String getAboutText() {  
    return "Poseidon for Football Tactics";  
}  
  
@Override  
protected ImageIcon getAboutIcon() {  
    return Utilities.loadImageIcon("ball");  
}
```



*

If you want to create your own about window design, you are free to do so. Override method `actionPerformed`. Below is a simple example

```
@Override
public void actionPerformed(ActionEvent e) {
    final JDialog frame = new JDialog(applicationWindow);
    frame.setSize(250, 70);
    int windowWidth = applicationWindow.getWidth();
    int windowHeight = applicationWindow.getHeight();
    frame.setLocation(windowWidth / 2 - frame.getWidth() / 2,
        windowHeight / 2 - frame.getHeight() / 2);
    JLabel label = new JLabel("My Text");
    frame.add(label, BorderLayout.CENTER);
    frame.setVisible(true);
}
```



Toolbar

Customizing toolbar

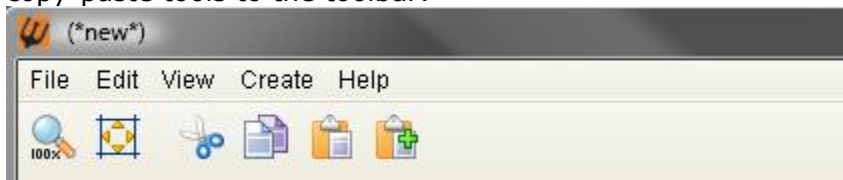
Poseidon toolbar is customizable. By default, it contains a lot of tools (see screenshot below), but you have the power to add or remove them.



The class you will use for customizing the toolbar is `com.gentleware.poseidon.custom.gui.toolbar.CustomMainToolBar`. For example, you want only zoom and copy-paste tools on your toolbar and you don't want to have the rest of the tools. Edit class `CustomMainToolBar` and override method `createPoseidonToolBar()` so that only the tools you want are created:

```
@Override
protected void createPoseidonToolBar() {
    makeZoomTools();
    addSeparator();
    makeCopyPasteTools();
}
```

So, now, instead of creating all of the default tools, this method will now add only zoom and copy-paste tools to the toolbar:



Creating tool on a toolbar

Say, you want to create tool for your "Create foo" action (see [Adding new sub-item to existing main menu item](#)). Add method `makeFooTool()` to class `com.gentleware.poseidon.custom.gui.toolbar.CustomMainToolBar` and call it from `createPoseidonToolBar()`:

```
@Override
protected void createPoseidonToolBar() {
    makeZoomTools();
    addSeparator();
    makeCopyPasteTools();
    makeFooTool();
}

private void makeFooTool() {
    final ToolBarFactory factory = ToolBarFactory.getInstance();
    final AbstractButton createFooButton = factory.createToolBarButton(
        CustomMainMenuResourceBundle.CreateFoo, this, IconLoader
            .getIconLoader().get("blank"));
}
```

```

createFooButton.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        // create foo
    }

});
}

```

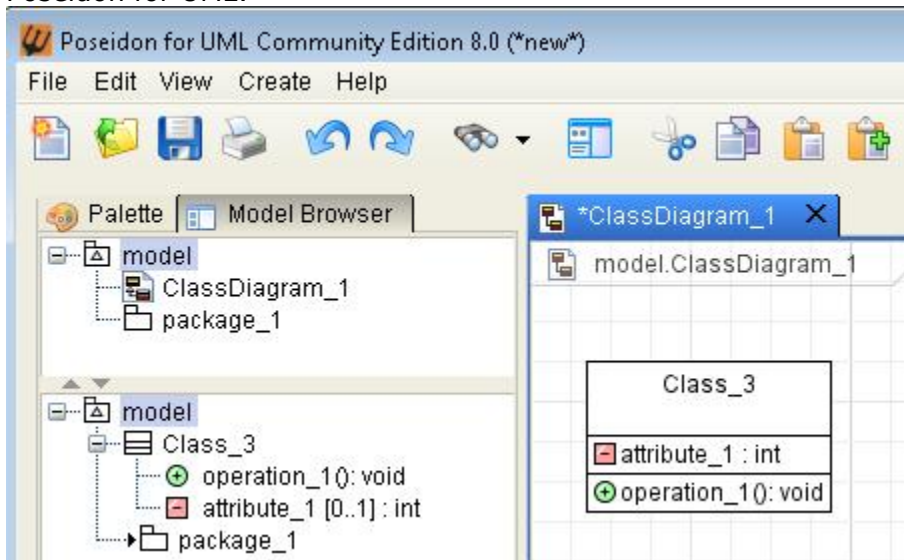
Now you have a new tool on the Poseidon toolbar:



Model Browser

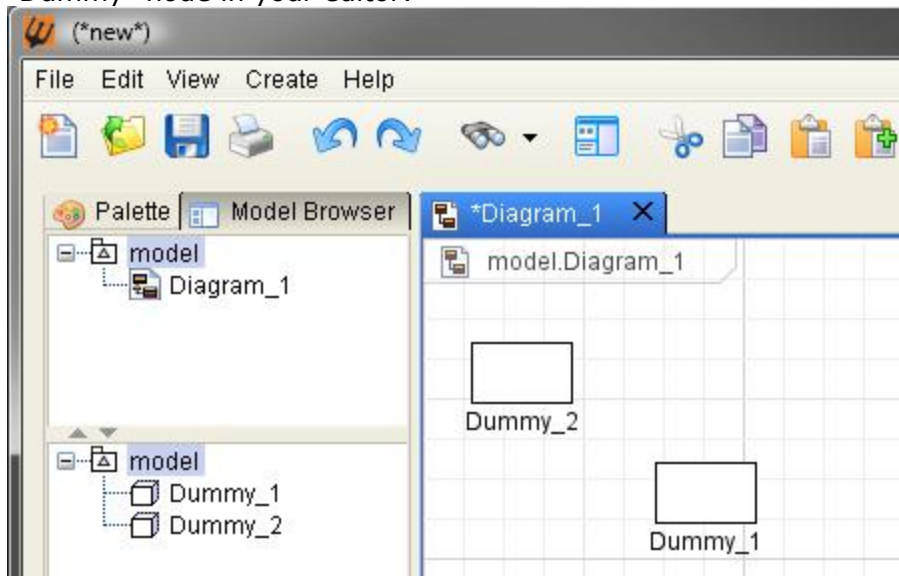
Icons in the Model Browser

In the Model Browser every node has an icon. For example, here is the Model Browser in Poseidon for UML:



By default, the icon in the Model Browser is the icon you specified in the diagram model for

the given node. But you also have possibility to manage icons. For example, you have some "Dummy" node in your editor:



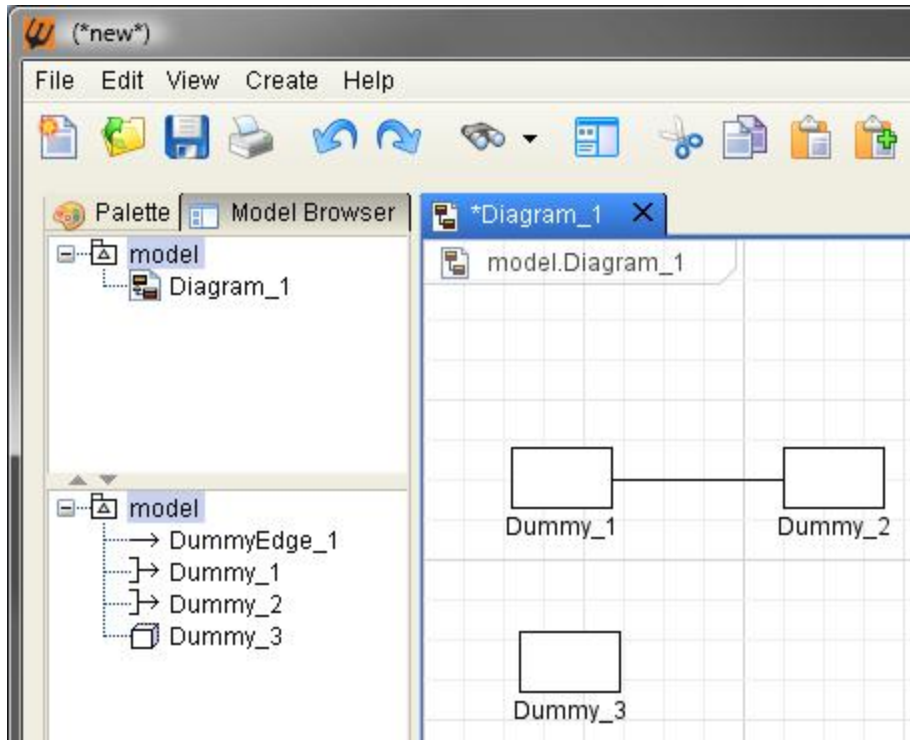
Suppose, you want to show different icons for the "Dummy" node in the Model Browser depending on context. Let's say that you want to show icon `com.gentleware.poseidon.custom.repobrowser.CustomModelBrowserNodeRenderer` and override method `setUpIcons()`:

```
@Override
protected void setUpIcons() {
    super.setUpIcons();

    AbstractDSLIconDeterminer iconDeterminer = new
AbstractDSLIconDeterminer() {

        public boolean isRelevant(PoseidonCoreElement element) {
            if (!element.get__incomingRelationships().isEmpty()
                && !element.get__outgoingRelationships().isEmpty()) {
                return true;
            }
            return false;
        }
    };
    addIcon(DummyNodeImpl.class, null, ShortCutType.NONE,
        "node-with-edge", iconDeterminer);
}
```

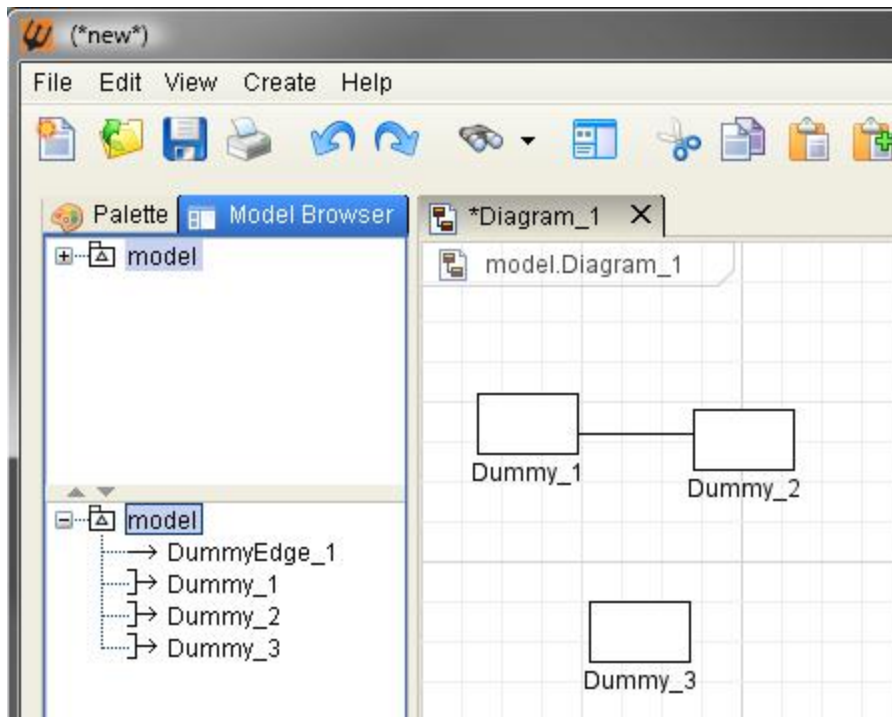
In this piece of code we add a new icon "node-with-edge" `↳ DummyNodeImpl`. We also pass an instance of `DSLIconDeterminer` class to the `addIcon` method which means that this icon will be applied only if method `isRelevant` returns true; The implementation of `isRelevant` method returns true only if the node has connected edges. Now the icon for the "Dummy" node in the Model Browser depends on context:



If you just wish to use another icon for the node, without any conditions, just pass `null` instead of the `DSLIconDeterminer` class implementation:

```
@Override
protected void setUpIcons() {
    super.setUpIcons();
    addIcon(DummyNodeImpl.class, null, ShortCutType.NONE, "node-with-
edge", null);
}
```

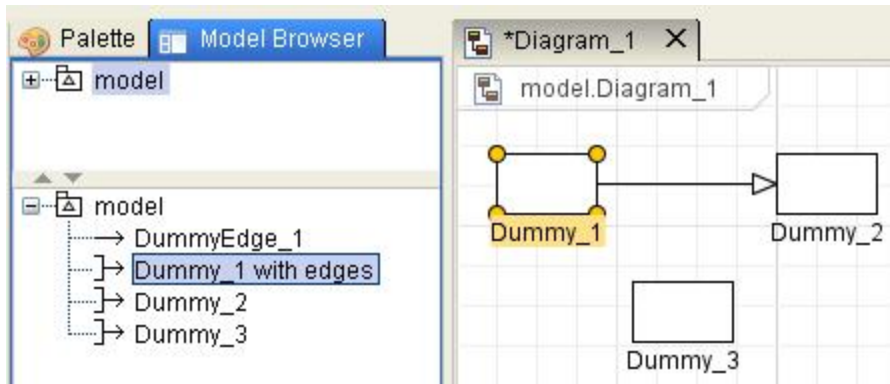
In this case the icon "node-with-edge" will be used for "Dummy" node in all cases.



Text in the Model Browser

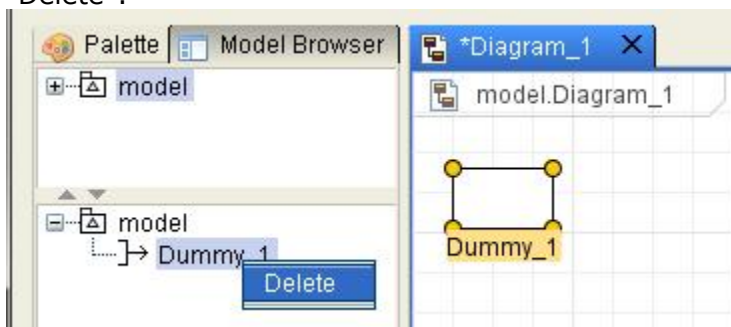
By default, the node text in the tree of the Model Browser is an element's name. You can change that in class `com.gentleware.poseidon.custom.repobrowser.CustomTreeMediator`. For example, you want to change the text for the "Dummy" node in the Model Browser if the node has outgoing edges. Just override method `getText` in class `CustomTreeMediator`.

```
@Override
protected String getText(PoseidonCoreElement element,
    CustomDSLNodeText nodeText) {
    String text = super.getText(element, nodeText);
    if (element instanceof DummyNode
        && !element.get__outgoingRelationships().isEmpty()) {
        text = text + " with edges";
    }
    return text;
}
```



Context menu in the Model Browser

By default, the context menu of elements in the Model Browser tree contains only one item "Delete":



Adding your own items to the context menu is easy. Edit class

`com.gentleware.poseidon.custom.repobrowser.CustomTreeMenuCreator` and override method `createMenu`

```
@Override
public JPopupMenu createMenu(ToolCoordinatorFacet coordinator,
    DefaultMutableTreeNode node) {
    JPopupMenu menu = super.createMenu(coordinator, node);
    DSLTreeUserObject treeUserObject = (DSLTreeUserObject)
node.getUserObject();
    PoseidonCoreElement element = treeUserObject.getElement();
    if (element instanceof DummyNode) {
        String localizedMenuItemName = Localizer
.localized(CustomRepositoryBrowserResourceBundle.CreateFoo);
        JMenuItem createFooMenuItem = new
JMenuItem(localizedMenuItemName);
        menu.add(createFooMenuItem);
    }
}
```

```

return menu;
}

```

This code adds a new context menu item to the "Dummy" node in the tree of the Model Browser. Of course, you also have to create the resource id we are using. Add following line to class

```

com.gentleware.poseidon.custom.repobrowser.CustomRepositoryBrowserResourceBundle
public static ResourceId CreateFoo = new ResourceId(BUNDLE, "CreateFoo");

```

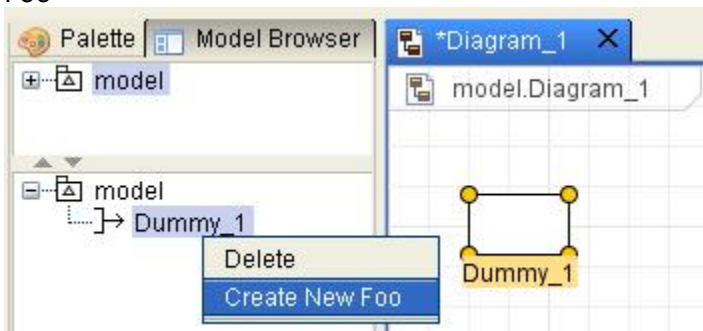
and the localized string value to `CustomRepositoryBrowserResourceBundle.properties`

```

CreateFoo = Create New Foo

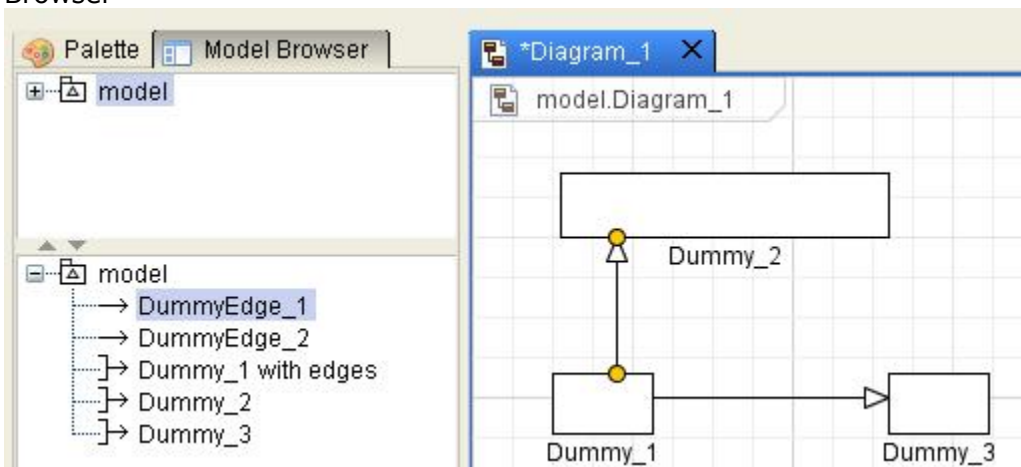
```

Now the "Dummy" node context menu in the Model Browser has new action "Create New Foo"



Removing nodes from Model Browser

By default, all elements of the model are shown in the Model Browser tree. If you want to hide some elements, you have to override method `rejectElementFromTree` of class `com.gentleware.poseidon.custom.repobrowser.CustomTreeMediator`. For example, you want to remove "DummyEdge" edges from the tree. By default, they are shown in the Model Browser

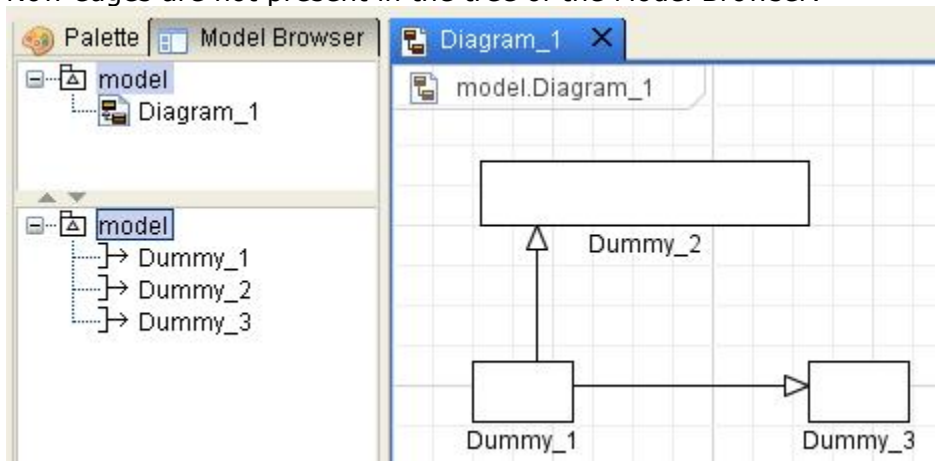


Add this method implementation to class

`com.gentleware.poseidon.custom.repobrowser.CustomTreeMediator` to remove edges from the tree:

```
@Override
protected boolean rejectElementFromTree(PoseidonCoreElement parent,
    PoseidonCoreElement child) {
    boolean rejected = super.rejectElementFromTree(parent, child);
    if (!rejected && child instanceof DummyEdge) {
        rejected = true;
    }
    return rejected;
}
```

Now edges are not present in the tree of the Model Browser:



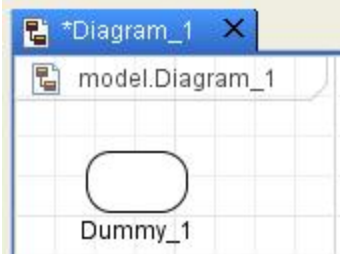
Diagram

Changing the shape of a node

You can choose the shape of the node in Poseidon diagram model from a pre-defined shapes set (Rectangle, Triangle, Star, Rhombus, ...). The choice of shapes is pretty big, but still limited. If you want to draw your own shape, you can do that. For example, you have a "Dummy" diagram element with the following definition on the diagram model:

```
node Dummy {
    metamodel_element: DummyNode
    shape: ROUNDED_RECTANGLE
}
```

On the diagram the "Dummy" node looks (surprise!) like a rounded rectangle:



The generated class which is responsible for the "Dummy" node on the diagram is `com.gentleware.poseidon.custom.diagrams.node.impl.DummyNodeGem`. To change the shape of the node, add the following method implementation to

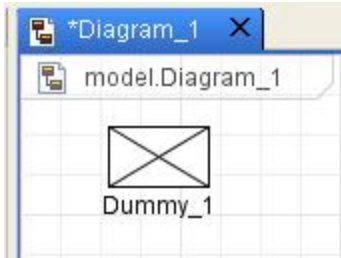
`DummyNodeGem`:

```
@Override
public NodeAppearanceFacet createBasicNodeAppearanceFacet() {
    return new BasicSeparateNameNodeAppearanceFacet(initialFillColor,
        initialFillColor, figureFacet, figureName, texttableFacet,
        subject, false) {

        public ShapeAppearanceFacet createShapeAppearanceFacet() {
            return new AbstractShapeAppearanceFacet() {
                @Override
                public ZVisualComponent[] drawShapes(UBounds bounds) {
                    ZShape rect = new ZRectangle(bounds);
                    UPoint topLeft = bounds.getTopLeftPoint();
                    UPoint bottomRight = bounds.getBottomRightPoint();
                    ZLine zLine = new ZLine(topLeft, bottomRight);
                    UPoint bottomLeft = bounds.getBottomLeftPoint();
                    UPoint topRight = bounds.getTopRightPoint();
                    ZLine zLine2 = new ZLine(bottomLeft, topRight);
                    return new ZShape[] { rect, zLine, zLine2 };
                }
            };
        }
    };
}
```

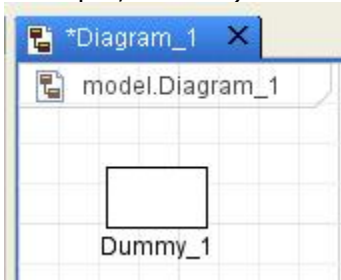
So, in this code snippet we create a new instance of `BasicSeparateNameNodeAppearanceFacet` (AppearanceFacets are responsible for the appearance of the node) with an overridden method `createShapeAppearanceFacet`. And, we create our own `ShapeAppearanceFacet` and implement method `drawShapes` so that it creates a rectangle and two diagonal lines inside the rectangle. Poseidon for DSLs uses [Jazz API](#) as a drawing engine. Please refer to the [documentation](#) if you are going to extensively create your own shapes in Poseidon (or if you don't understand the implementation of `drawShapes` method).

Now the "Dummy" node has the new custom shape:



Customizing the appearance of nodes and edges

Poseidon for DSLs API allows you to customize the appearance of the elements on the diagram. To do that, you have to override the implementation of an Appearance Facet of the appropriate element. Lets start with a node. Say, you want to add some small detail (for example, an icon) to the appearance of the simple rectangular "Dummy" node:



The generated class which is responsible for the "Dummy" node on the diagram is `com.gentleware.poseidon.custom.diagrams.node.impl.DummyNodeGem`. To change the appearance of the node, you have to override the implementation of the appearance facet of this gem. Add the following implementation to `DummyNodeGem`:

```
@Override
public NodeAppearanceFacet createBasicNodeAppearanceFacet() {
    return new BasicSeparateNameNodeAppearanceFacet(initialFillColor,
        initialFillColor, figureFacet, figureName, texttableFacet,
        subject, false) {

        @Override
        public ZNode formView() {
            // draw the default node shape
            ZGroup zGroup = (ZGroup) super.formView();

            // get bounds of the node on the diagram
            ClassifierSizes sizes = (ClassifierSizes) makeCurrentInfo()
                .makeSizes();
            Rectangle bounds = sizes.getOuter().getBounds();
            final double x = bounds.getX();
            final double y = bounds.getY();
```

```

final double width = bounds.getWidth();
final double height = bounds.getHeight();

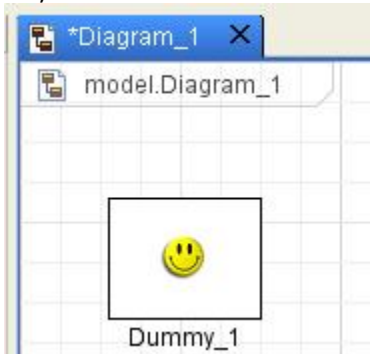
// load icon by name
String imageName = "smile-face";
ImageIcon imageIcon = Utilities.loadImageIcon(imageName);
Image image = imageIcon.getImage();
ZImage icon = new ZImage(image);

// set coordinates for the icon
double iconWidth = icon.getWidth();
double iconHeight = icon.getHeight();
double iconX = x + (width - iconWidth) / 2;
double iconY = y + (height - iconHeight) / 2;
icon.setTranslation(iconX, iconY);

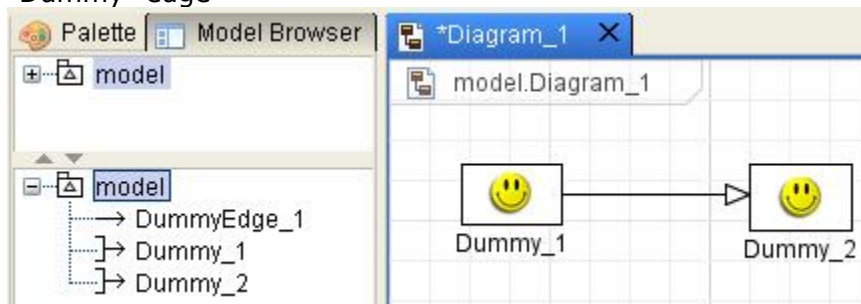
// add icon to view
zGroup.addChild(new ZVisualLeaf(icon));
return zGroup;
}
};
}

```

So, the code above adds a smile face icon to the center of the node:



In a similar way you can change the appearance of an edge. Say, you have a simple "Dummy" edge



and you want to add a circle at the starting point of the edge. You have to override the

implementation of the appearance facet of the "Dummy" edge. The class is `com.gentleware.poseidon.custom.diagrams.edge.impl.DummyEdgeArcGem`. Here is the code you need to add:

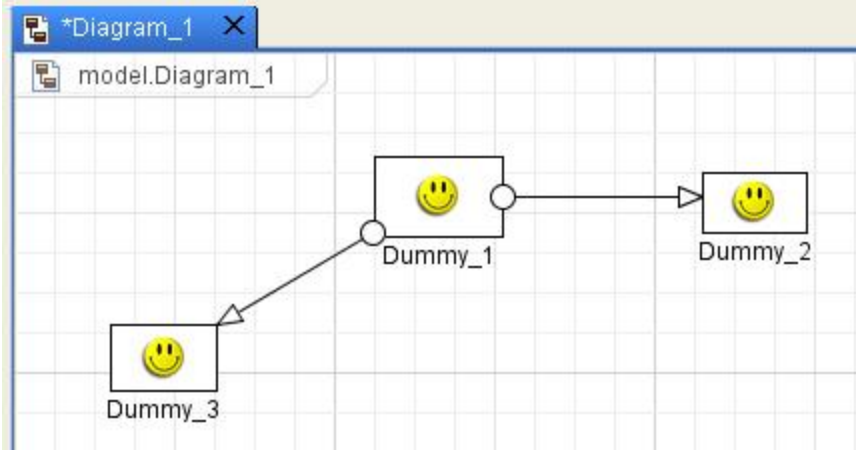
```
@Override
protected BasicArcAppearanceGem createBasicEdgeAppearanceFacet() {
    return new CustomBasicArcAppearanceFacet(fillColor, lineColor,
        figureFacet, figureName, subject) {

        public ArcEndAppearanceFacet createTargetEndAppearanceFacet() {
            return new ClosedArrowArcEndAppearanceFacet(false);
        }

        @Override
        public ZNode formAppearance(ZShape mainArc, UPoint start,
            UPoint second, UPoint secondLast, UPoint last,
            CalculatedArcPoints calculated, boolean curved) {
            // create the default appearance
            ZNode formAppearance = super.formAppearance(mainArc, start,
                second, secondLast, last, calculated, curved);

            // create and add a circle to the starting point of the edge
            if (mainArc instanceof ZPolyline) {
                ZPolyline mainLine = (ZPolyline) mainArc;
                double radius = 6;
                double diameter = radius * 2;
                ZEllipse zEllipse = new ZEllipse(mainLine.getX(0) -
radius,
                                mainLine.getY(0) - radius, diameter, diameter);
                ((ZGroup) formAppearance).addChild(new
ZVisualLeaf(zEllipse));
            }

            return formAppearance;
        }
    };
}
```



Adding new diagram types

In some editors you would want to create several diagram types. For example, in UML editor you want to have Class diagram, Use Case diagram, Sequence diagram, etc... First, you [add the desired diagram types to diagram model](#) and generate the code. Then you want to add some actions to Poseidon menus which will allow you to create the diagrams in the editor. For example, you added new diagram type "my" to diagram model and generated the code. Now you want to add an action which creates this type of diagram to Poseidon "Create" menu. First, let us create an action which creates the diagram. Go to project "Poseidon", package `com.gentleware.poseidon.custom.gui.menu` and create a new class `NewMyDiagramActionListener` in that package (right-click on the package and use wizard "New->Class"). As a superclass, select `com.gentleware.poseidon.diagrams.listeners.NewDefaultDiagramActionListener`, please also check "Constructors from superclass" checkbox:

Java Class

Create a new Java class.

Source folder: Poseidon/src

Package: com.gentleware.poseidon.custom.gui.menu

Enclosing type:

Name: NewMyDiagramActionListener

Modifiers: public default private protected
 abstract final static



Superclass: com.gentleware.poseidon.diagrams.listeners.NewDefaultDiagramActionListener

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
 Generate comments

Now add this method to the class you just created:

```
@Override
protected DslGenDiagramType getDiagramType() {
    return DslGenDiagramType.MY_LITERAL;
}
```

Now you want to add a menu item which actually creates the diagram to the Poseidon "Create" menu. Open class `com.gentleware.poseidon.custom.gui.menu.CustomMainMenu` in the project "Poseidon" and add the following method there:

```
@Override
protected void createPoseidonCreateMenu() {
    super.createPoseidonCreateMenu();
    ResourceId createMenuResourceId = CustomMainMenuResourceBundle.Create;
    JMenuItem myDiagramItem = factory.addMenuItem(createMenuResourceId,
```

```

        createMenuResourceId,
        CustomMainMenuResourceBundle.NewMyDiagram, "b");
    NewMyDiagramActionListener actionListener = new
NewMyDiagramActionListener(
        this.applicationWindow);
    myDiagramItem.addActionListener(actionListener);
}

```

You also have to add the following field declaration to class

`com.gentleware.poseidon.custom.gui.menu.CustomMainMenuResourceBundle:`

```
public static final ResourceId NewMyDiagram = new ResourceId(MAIN_MENU_BUNDLE, "NewMyDiagram");
```

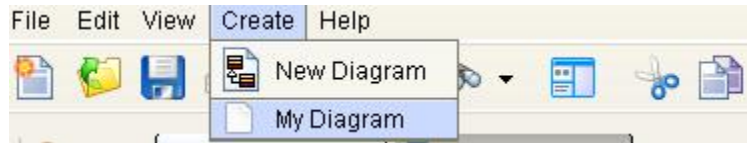
and the following line to property file

`com.gentleware.poseidon.custom.gui.menu.CustomMainMenuResourceBundle.properties`

:

```
NewMyDiagram=My Diagram
```

Now start Poseidon to check the result:



You can now create diagrams of your own "MyDiagram" type. By default, the set of tools in the tools palette for the new diagram type is the same as for the default diagram, but you can [customize which tools are available for the given diagram type](#).

Icons

You can use your own icons in Poseidon for DSLs. Just copy your *.png file to project "**resources**", folder "**icons**". This folder is in the classpath, so Poseidon will find the icon at runtime. To load an icon by name, use method `loadImageIcon` of class `com.gentleware.poseidon.swingx.Utilities`.

```
ImageIcon imageIcon = Utilities.loadImageIcon("smile-face");
```

IMPORTANT: Do not include file extension into the image name. The code above is correct. The code below is incorrect, it will NOT find the icon.

```
ImageIcon imageIcon = Utilities.loadImageIcon("smile-face.png");
```

Localization

Poseidon for DSLs has a bunch of property files which allow you to customize text in Poseidon GUI.

Menus

Files `com.gentleware.poseidon.custom.gui.menu.CustomMainMenuResourceBundle.java` and `com.gentleware.poseidon.custom.gui.menu.CustomMainMenuResourceBundle.properties` are responsible for:

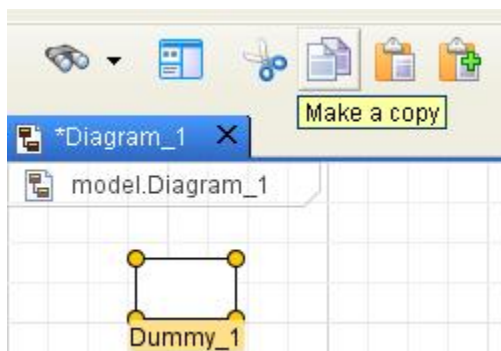
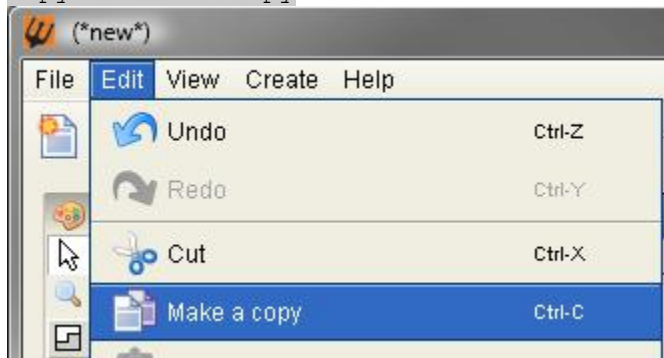
- Poseidon main menu texts
- context menu on the diagram
- toolbar hints

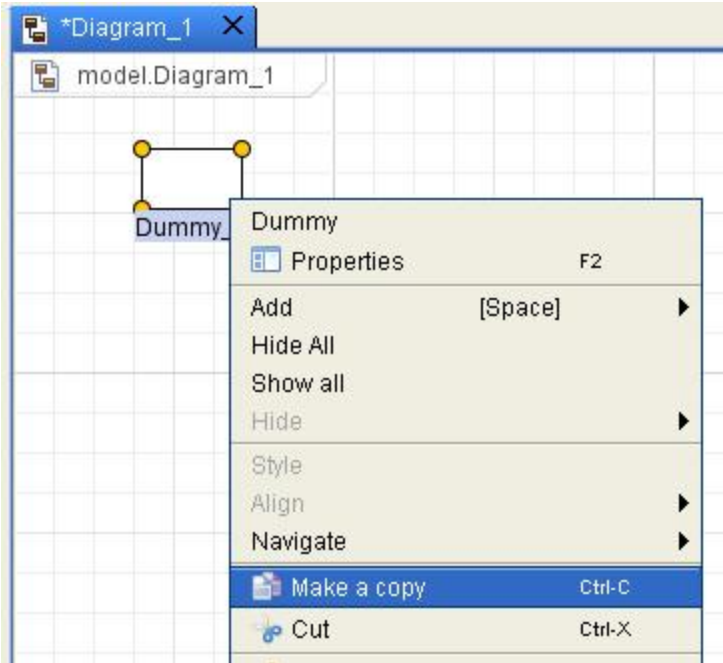
For example, you can change the caption of the "Cope" menu. Go to `CustomMainMenuResourceBundle.properties` and change line

`Copy = Copy`

to

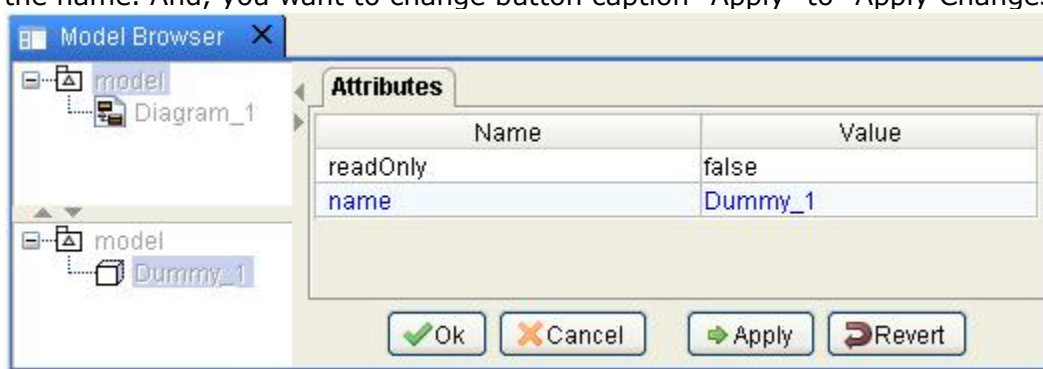
`Copy = Make a copy`





Model Browser

Files `CustomRepositoryBrowserResourceBundle.java` and `CustomRepositoryBrowserResourceBundle.properties` from package `com.gentleware.poseidon.custom.repobrowser` are responsible for Model Browser. You can change button captions or attribute names in the Model Browser. For example, if you want to change the caption of the "Apply" button, you suppose, you have an attribute "readOnly" for the "Dummy" node, but you don't really like the name. And, you want to change button caption "Apply" to "Apply Changes".



Open file `CustomRepositoryBrowserResourceBundle.properties` and change line `Apply = Apply` to `Apply = Apply Changes`. Then add line

readOnly = Not Changeable

Now the Model Browser looks like this:



Tools palette

See [Tools model](#).