Annotations and Pragmas

Annotations

The textual SCCharts language supports several annotations to influence the visual representation of the model.

Annotation are processed in sequential order.

<table>
<tr><th>Pattern</th><th>Usage</th><th>Description</th><th>Example</th></tr>
<tr>
<td><code>@diagram [<key>] <value></code></td>
<td>
<table>
<tr><td>Location:</td><td>scchart</td></tr>
<tr><td><key></td><td>The name of the synthesis option. The given name is evaluated case-insensitive and whitespace-ignoring. The options are searched for the first matching prefix.</td></tr>
<tr><td><value></td><td>The value type depends on the option type:

CheckBox: true or false

Choice: Name of choice item

Slider: Float value</td></tr>
</table>
</td>
<td>
Sets the synthesis option identified by <key> to the given value.

The available synthesis options for a diagram are displayed in the sidebar of the diagram view.

The values from the sidebar will be ignored if a corresponding annotation is present.
<table>
<tr><td><code>initiallycollapseregions</code></td><td>Collapses all regions. Very helpful for lager models, since it fastens initial diagram rendering.</td></tr>
</table>
</td>
<td>

```
@diagram
[paper] true
scchart
Testing {
  initial
state A
  --> B;
  final state
B;
}
```

</td>
</tr>
</table>

<table>
<tr>
<td><code>@layout
[<key>]
<value></code></td>
<td>
<table>
<tr><td>Location:</td><td>scchart, state, region, transition</td></tr>
<tr><td><key></td><td>The ID of the layout option. The options are searched for the first matching postfix.</td></tr>
<tr><td><value></td><td>The value type depends on the option type. The value is parsed case-sensitive.</td></tr>
</table>
</td>
<td>
Sets the layout property identified by <key> to the given value on the annotated element.

The available layout options are documented here.

Layout options will only affect the annotated element and no underlying hierarchy levels.

If a layout direction is specified with this annotation it overrides the layout direction set by HV-/VH-Layout in any parent element for this element.

Special case: If the direction is set on the scchart element (top level) it overrides the default alternating layout.

The layout option is identified by matching a postfix. Hence the key <code>direction</code> matches both <code>org.eclipse.elk.direction</code> and <code>org.eclipse.elk.layered.priority.direction</code>.
If none or multiple options match a warning is displayed.
<table>
<tr><td><code>elk.direction</code></td><td>Layout direction</td></tr>
<tr><td><code>elk.priority</code></td><td>Influences the order of regions</td></tr>
</table>
</td>
<td>
<pre>scchart
Testing {
    @layout
[algorithm]
org.eclipse.
elk.graphviz.
circo
    region:
    initial
final state A
    --> B;
    state B
    --> C;
    state C
    --> A;
}</pre>
<pre>scchart
Testing {
    @layout
[elk.
direction] UP
    region
"up":
    initial
state A
    --> B;
    final
state B;
    @layout
[elk.
direction] LEFT
    region
"left":
    initial
state A
    --> B;
    final
state B;
}</pre>
</td>
</tr>
<tr>
<td><code>@HVLayout
@VHLayout</code></td>
<td>
<table>
<tr><td>Location:</td><td>scchart, state, region</td></tr>
</table>
</td>
<td>
Defines the order of the alternating layout directions.

The annotation can be mixed and nested in the SCChart and will only affect succeeding hierarchy levels.

The default is an implicit HVLayout starting at the top level state.
</td>
<td>
<pre>@VHLayout
scchart
Testing {
  initial
state A
    go to B;
  final state
B;
}</pre>
</td>
</tr>
</table>

`@collapse`
`@expand`
	Location: region	The annotated region will be initially collapse or expanded.	`scchart Testing {`
			`  @collapse`
			`  region {`
			`  initial state A`
			`  go to B;`
			`  final state B;`
			`  }`
			`}`

`@hide`	**Location:** scchart, state, region, transition	The annotated element will be excluded from the diagram.
		
		Transitions with a hidden source or target state will be hidden as well.
			`scchart Testing {`
			`  initial state A`
			`  go to B;`
			`  @hide`
			`  final state B;`
			`}`

Pragmas

Pragmas are annotations that are valid for the whole file in contrast to annotations that are valid for semantic model elements. They are placed in front of an .sctx.

Example

```
#pragma
scchart Testing {
  ...
}
```

Pragma	Effect
#KiCoEnv {<json>}	Configures the compiler environment.
#hostcode <code> #hostcode-[c | c-header | java] <code>	Allows hostcode additions that are placed at the beginning of the generated code file. The exact handling may depend on the used code generator. NEW IN 1.1 There are also language specific variants that will only affect the specific code generation, e.g. #hostcode-java.
#code.naming	NEW IN 1.1 Configures the code generation to use different names for generated functions. *#code.naming <TICK_FUNCTION_NAME>, <RESET_FUNCTION_NAME>, <LOGIC_FUNCTION_NAME>, <TICKDATA_STRUCT_NAME>* Sets the name for the four functions. All four parameters must be present. *#code.naming suffix* *#code.naming prefix* Code generation will use default function names but will prefix/suffix these names with the model name.

#resource <file | directory>	NEW IN 1.1
	
	The given resources (single files or directories) will be copied to the generated code folder (usually *kieler-gen*). Since this is the working directory for the compilation KIELER these files can be included via hostcode integration.
	If the compilation contains any down-stream compiler invocation (e.g. gcc) all given files and all files in given directories that match the usual source code file extension (e.g. *.c) will be included in the compilation and compile into an executable with the generated code.
	
	All non-absolute paths will be resolved relative to the model file.
	
	Example
	
	`#resource "myheader.h"`
	`#resource "mycode.c"`
	`#hostcode "#include \"myheader.h\""`
	`scchart Testing {`
	`  ...`
	`}`

#HideImportedSCCharts	NEW IN 1.1 This will hide all SCCharts that are imported from other files in the diagram, if the 'All SCCharts' synthesis option is activated.