

# Extending Modeler with new task type

As part of the [Jenkow Jenkins Plugin](#) which leverages the Activiti workflow engine for Jenkins job orchestration, I have added a new task type "Jenkins task" to the Activiti Modeler. Because unlike the Designer, the Modeler doesn't (yet) have a mechanism for incrementally adding new task types, I had to modify the Activiti code base. This Wiki page summarizes the changes I had to do. The enhanced Activiti code base is available on [GitHub](#). My enhancement is done on top of Activiti 5.11.

My new task type is mapped onto the existing ServiceTask, using a custom task delegate class at run time.

It's good to take an existing task type as the starting point. User task or script task is a good choice.

## The artwork

The new artwork went into `modules/activiti-webapp-explorer2/src/main/webapp/editor/stencils`  
`ets/bpmn2.0/icons/jenkins`.

- Created an icon PNG file
- Created a stencil SVG file
  - I used Inkscape with "Save as Plain SVG"
    - Manually copied the relevant <g> element (SVG group) defining the stencil logo into the desired stencil SVG. The SVG saved with Inkscape didn't fully work.
    - Initially my logo <g> element had a transform attribute which confused the Modeler's calculation of the stencil's bounding box. Ungrouping & regrouping made Inkscape remove the transform attribute and place the recalculated path coordinates into the <g> element. That solved the bounding box problem.
    - I had to add an `oryx:anchors="top left"` attribute to each path element, otherwise the logo would not remain intact when the stencil got resized.

## New stencil

In `modules/activiti-webapp-explorer2/src/main/resources/stencilset.json`

- added a task base

```
{
  "name" : "mynewtaskbase",
  "properties" : [ {
    "id" : "mytaskprop1",
    "type" : "String",
    "title" : "My Task Prop One",
    "value" : "",
    "description" : "One and only property of my new task",
    "popular" : true
  } ]
}
```

- added the actual task

```

{
  "type" : "node",
  "id" : "MyNewTask",
  "title" : "My new task",
  "description" : "Does something new",
  "view" : "mynewtask/mynewtask.svg",
  "icon" : "mynewtask/mynewtask.png",
  "groups" : [ "My New Group" ],
  "propertyPackages" : [ "elementbase", "baseattributes",
"mynewtaskbase" ],
  "roles" : [ "sequence_start", "Activity", "sequence_end",
"ActivitiesMorph", "all" ]
}

```

I don't fully understand the semantics of all existing propertyPackages.

## Java Code

Declared a new stencil name constant in `modules/activiti-json-converter/src/main/java/org/activiti/editor/constants/StencilConstants.java`:

```

final String STENCIL_TASK_MYNEW =
"MyNewTask";

```

The heavy lifting is done by a new class `modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/MyNewTaskJsonConverter.java`. It is added to the existing converters in `modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/BpmnJsonConverter.java` by calling `MyNewTaskJsconConverter.fillTypes()` in the static initializer. It's important to also add the stencil name into the right `DI_...` shape map, otherwise edges to the new stencil miss a waypoint.

```

DI_RECTANGLES.add( STENCIL_TASK_MYNEW );

```

Here, the method `convertJsonToElement()` is used to populate a `ServiceTask` model object which later gets converted into XML.

Added another element in the static initializer of the diagram generator `modules/activiti-engine/src/main/java/org/activiti/engine/impl/bpmn/diagram/ProcessDiagramGenerator.java`:

```
// my new task

activityDrawInstructions.put("myNewTask",
new ActivityDrawInstruction() {
    public void draw(ProcessDiagramCanvas
processDiagramCreator, ActivityImpl
activityImpl) {

processDiagramCreator.drawScriptTask((String
) activityImpl.getProperty("name"),
activityImpl.getX(), activityImpl.getY(),
activityImpl.getWidth(),
                activityImpl.getHeight());
    }
});
```

## Import

Adding support for importing processes with a new task type is a bit more complicated in this case, because the Jenkins task is internally mapped to the Service Task.

I had to hard-code logic into the

As part of the [Jenkow Jenkins Plugin](#) which leverages the Activiti workflow engine for Jenkins job orchestration, I have added a new task type "Jenkins task" to the Activiti Modeler. Because unlike the Designer, the Modeler doesn't (yet) have a mechanism for incrementally adding new task types, I had to modify the Activiti code base. This Wiki page summarizes the changes I had to do. The enhanced Activiti code base is available on [GitHub](#). My enhancement is done on top of Activiti 5.11.

My new task type is mapped onto the existing ServiceTask, using a custom task delegate class at run time.

It's good to take an existing task type as the starting point. User task or script task is a good choice.

## The artwork

The new artwork went into `modules/activiti-webapp-explorer2/src/main/webapp/editor/stencils/ets/bpmn2.0/icons/jenkins`.

- Created an icon PNG file

- Created a stencil SVG file
  - I used Inkscape with "Save as Plain SVG"
    - Manually copied the relevant <g> element (SVG group) defining the stencil logo into the desired stencil SVG. The SVG saved with Inkscape didn't fully work.
    - Initially my logo <g> element had a transform attribute which confused the Modeler's calculation of the stencil's bounding box. Ungrouping & regrouping made Inkscape remove the transform attribute and place the recalculated path coordinates into the <g> element. That solved the bounding box problem.
    - I had to add an `oryx:anchors="top left"` attribute to each path element, otherwise the logo would not remain intact when the stencil got resized.

## New stencil

In `modules/activiti-webapp-explorer2/src/main/resources/stencilset.json`

- added a task base

```
{
  "name" : "mynewtaskbase",
  "properties" : [ {
    "id" : "mytaskprop1",
    "type" : "String",
    "title" : "My Task Prop One",
    "value" : "",
    "description" : "One and only property of my new task",
    "popular" : true
  } ]
}
```

- added the actual task

```
{
  "type" : "node",
  "id" : "MyNewTask",
  "title" : "My new task",
  "description" : "Does something new",
  "view" : "mynewtask/mynewtask.svg",
  "icon" : "mynewtask/mynewtask.png",
  "groups" : [ "My New Group" ],
  "propertyPackages" : [ "elementbase", "baseattributes",
    "mynewtaskbase" ],
  "roles" : [ "sequence_start", "Activity", "sequence_end",
    "ActivitiesMorph", "all" ]
}
```

I don't fully understand the semantics of all existing `propertyPackages`.

## Java Code

Declared a new stencil name constant in `modules/activiti-json-converter/src/main/java/org/activ`

iti/editor/constants/StencilConstants.java:

```
final String STENCIL_TASK_MYNEW =  
"MyNewTask";
```

The heavy lifting is done by a new class `modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/MyNewTaskJsonConverter.java`. It is added to the existing converters in `modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/BpmnJsonConverter.java` by calling `MyNewTaskJsonConverter.fillTypes()` in the static initializer. It's important to also add the stencil name into the right `DI_...` shape map, otherwise edges to the new stencil miss a waypoint.

```
DI_RECTANGLES.add(STENCIL_TASK_MYNEW);
```

Here, the method `convertJsonToElement()` is used to populate a `ServiceTask` model object which later gets converted into XML.

Added another element in the static initializer of the diagram generator `modules/activiti-engine/src/main/java/org/activiti/engine/impl/bpmn/diagram/ProcessDiagramGenerator.java`:

```
// my new task

activityDrawInstructions.put("myNewTask",
new ActivityDrawInstruction() {
    public void draw(ProcessDiagramCanvas
processDiagramCreator, ActivityImpl
activityImpl) {

processDiagramCreator.drawScriptTask((String
) activityImpl.getProperty("name"),
activityImpl.getX(), activityImpl.getY(),
activityImpl.getWidth(),
                activityImpl.getHeight());
    }
});
```

## Import

Adding support for importing processes with a new task type is a bit more complicated in this case, because the Jenkins task is internally mapped to the Service Task.

I had to hard-code logic into the

As part of the [Jenkow Jenkins Plugin](#) which leverages the Activiti workflow engine for Jenkins job orchestration, I have added a new task type "Jenkins task" to the Activiti Modeler. Because unlike the Designer, the Modeler doesn't (yet) have a mechanism for incrementally adding new task types, I had to modify the Activiti code base. This Wiki page summarizes the changes I had to do. The enhanced Activiti code base is available on [GitHub](#). My enhancement is done on top of Activiti 5.11.

My new task type is mapped onto the existing ServiceTask, using a custom task delegate class at run time.

It's good to take an existing task type as the starting point. User task or script task is a good choice.

## The artwork

The new artwork went into `modules/activiti-webapp-explorer2/src/main/webapp/editor/stencils/ets/bpmn2.0/icons/jenkins`.

- Created an icon PNG file

- Created a stencil SVG file
  - I used Inkscape with "Save as Plain SVG"
    - Manually copied the relevant <g> element (SVG group) defining the stencil logo into the desired stencil SVG. The SVG saved with Inkscape didn't fully work.
    - Initially my logo <g> element had a transform attribute which confused the Modeler's calculation of the stencil's bounding box. Ungrouping & regrouping made Inkscape remove the transform attribute and place the recalculated path coordinates into the <g> element. That solved the bounding box problem.
    - I had to add an `oryx:anchors="top left"` attribute to each path element, otherwise the logo would not remain intact when the stencil got resized.

## New stencil

In `modules/activiti-webapp-explorer2/src/main/resources/stencilset.json`

- added a task base

```
{
  "name" : "mynewtaskbase",
  "properties" : [ {
    "id" : "mytaskprop1",
    "type" : "String",
    "title" : "My Task Prop One",
    "value" : "",
    "description" : "One and only property of my new task",
    "popular" : true
  } ]
}
```

- added the actual task

```
{
  "type" : "node",
  "id" : "MyNewTask",
  "title" : "My new task",
  "description" : "Does something new",
  "view" : "mynewtask/mynewtask.svg",
  "icon" : "mynewtask/mynewtask.png",
  "groups" : [ "My New Group" ],
  "propertyPackages" : [ "elementbase", "baseattributes",
    "mynewtaskbase" ],
  "roles" : [ "sequence_start", "Activity", "sequence_end",
    "ActivitiesMorph", "all" ]
}
```

I don't fully understand the semantics of all existing `propertyPackages`.

## Java Code

Declared a new stencil name constant in `modules/activiti-json-converter/src/main/java/org/activ`

iti/editor/constants/StencilConstants.java:

```
final String STENCIL_TASK_MYNEW =  
"MyNewTask";
```

The heavy lifting is done by a new class `modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/MyNewTaskJsonConverter.java`. It is added to the existing converters in `modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/BpmnJsonConverter.java` by calling `MyNewTaskJsonConverter.fillTypes()` in the static initializer. It's important to also add the stencil name into the right `DI_...` shape map, otherwise edges to the new stencil miss a waypoint.

```
DI_RECTANGLES.add(STENCIL_TASK_MYNEW);
```

Here, the method `convertJsonToElement()` is used to populate a `ServiceTask` model object which later gets converted into XML.

Added another element in the static initializer of the diagram generator `modules/activiti-engine/src/main/java/org/activiti/engine/impl/bpmn/diagram/ProcessDiagramGenerator.java`:

```
// my new task

activityDrawInstructions.put("myNewTask",
new ActivityDrawInstruction() {
    public void draw(ProcessDiagramCanvas
processDiagramCreator, ActivityImpl
activityImpl) {

processDiagramCreator.drawScriptTask((String
) activityImpl.getProperty("name"),
activityImpl.getX(), activityImpl.getY(),
activityImpl.getWidth(),
                                activityImpl.getHeight());
    }
});
```

## Import

Adding support for importing processes with a new task type is a bit more complicated in this case, because the Jenkins task is internally mapped to the Service Task.

I had to hard-code logic into  
modules/activiti-json-converter/src/main/java/org/activiti/editor/language/json/converter/ServiceTaskJsonConverter.java to get this mapping.

[details](#)