User Task Forms

There are different types of forms which are primarily used in Tasklist. To implement a task form in your application, you have to connect the form resource with the BPMN 2.0 element in your process diagram. Suitable BPMN 2.0 elements for calling tasks forms are the [StartEvent](https://docs.camunda.org/manual/7.15/reference/bpmn20/events/start-events/) and the [UserTask](https://docs.camunda.org/manual/7.15/reference/bpmn20/tasks/user-task/).

Forms are Referenced using Form Keys and can either be embedded in Camunda Tasklist or handled by a custom application. Depending on you use-case, different Form Types can be used:

1. [Embedded Task Forms](https://docs.camunda.org/manual/7.15/user-guide/task-forms/#embedded-task-forms) allow you to embed custom HTML and JavaScript forms into Tasklist.
2. [Camunda Forms](https://docs.camunda.org/manual/7.15/user-guide/task-forms/#camunda-forms) offer visual editing of forms in the Camunda Modeler and can be used for less complex forms.
3. [External Task Forms](https://docs.camunda.org/manual/7.15/user-guide/task-forms/#external-task-forms) can be used to link to custom applications. The Form will not be embedded in Tasklist.

If no form key is present, a [Generic Task Form](https://docs.camunda.org/manual/7.15/user-guide/task-forms/#generic-task-forms) will be shown.

Form Key Details

Form keys that are used in Tasklist have the structure FORM-TYPE:LOCATION:FORM.NAME.

|  |  |
| --- | --- |
| **Name** | **Description** |
| FORM-TYPE | Can be embedded or camunda-forms depending on the form type. If no type is set, the form will be shown as an [External Task Form](https://docs.camunda.org/manual/7.15/user-guide/task-forms/#external-task-forms). |
| LOCATION | Can be either deployment or app:   * *deployment*: The file is part of your deployment (e.g., [by adding it to your process archive](https://docs.camunda.org/manual/7.15/reference/deployment-descriptors/tags/process-archive/) or [by deploying from the Camunda Modeler](https://docs.camunda.org/get-started/quick-start/deploy/#use-the-camunda-modeler-to-deploy-the-process)), which means that it is stored in the Camunda database. It can then be loaded from there. Note that this allows versioning of your form alongside the process model. * *app*: Add the file to your development project in a folder src/main/webapp/forms. The form file will be packaged into your deployment artifact (typically a WAR archive). During runtime it will be loaded from there. |
| FORM.NAME | The file name and path in the deployment, e.g. forms/startFrom.html |

To configure the form in your process, open the process with the [Camunda Modeler](http://camunda.org/bpmn/tool/) and select the desired [UserTask](https://docs.camunda.org/manual/7.15/reference/bpmn20/tasks/user-task/) or [StartEvent](https://docs.camunda.org/manual/7.15/reference/bpmn20/events/start-events/). Open the properties panel and enter the Form Key. The relevant XML tag looks like this:

<userTask id="theTask" camunda:formKey="camunda-forms:deployment:forms/userTask.form"

camunda:candidateUsers="John, Mary"

name="my Task">

Embedded Task Forms

Embedded task forms are HTML and JavaScript forms. We provide more information about the creation of embedded forms in our [Embedded Task Forms Reference](https://docs.camunda.org/manual/7.15/reference/forms/embedded-forms/).

To add an embedded form to your application, simply create an HTML file and refer to it from a [UserTask](https://docs.camunda.org/manual/7.15/reference/bpmn20/tasks/user-task/) or a [StartEvent](https://docs.camunda.org/manual/7.15/reference/bpmn20/events/start-events/) in your process model. For example, you can create a FORM\_NAME.html file containing the relevant content for your form, e.g., a simple form with two input fields:

<form role="form" name="form">

<div class="form-group">

<label for="customerId-field">Customer ID</label>

<input required

cam-variable-name="customerId"

cam-variable-type="String"

class="form-control" />

</div>

<div class="form-group">

<label for="amount-field">Amount</label>

<input cam-variable-name="amount"

cam-variable-type="Double"

class="form-control" />

</div>

</form>

The form key for this file could be embedded:deployment:FORM\_NAME.html or embedded:app:forms/FORM\_NAME.html.

Camunda Forms

Camunda Forms are created as separate files using the Camunda Modeler and can be deployed together with the process models. The form schema is stored in .form files. You can find out how to build Camunda Forms in the [Camunda Modeler documentation](https://docs.camunda.org/manual/7.15/modeler/forms/) or refer to the [Camunda Forms Reference](https://docs.camunda.org/manual/7.15/reference/forms/camunda-forms/#configuration) to explore all configuration options for form elements.

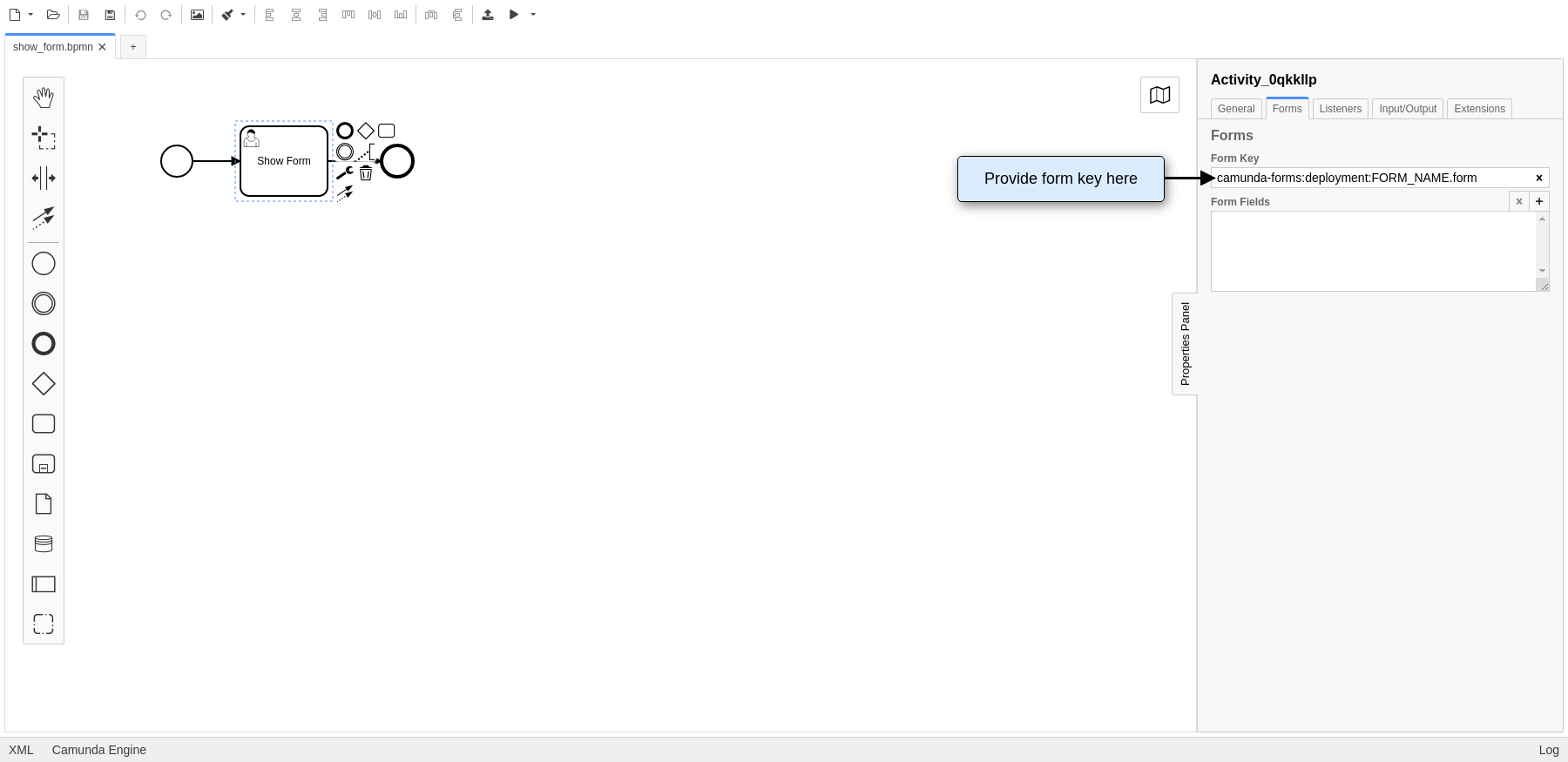
[Process variables](https://docs.camunda.org/manual/7.15/user-guide/process-engine/variables/) are mapped to form fields where the fields key matches the variable name.

Form Key

To reference a Camunda Forms file, you can use a deployment or app [form key](https://docs.camunda.org/manual/7.15/user-guide/task-forms/#form-key-details):

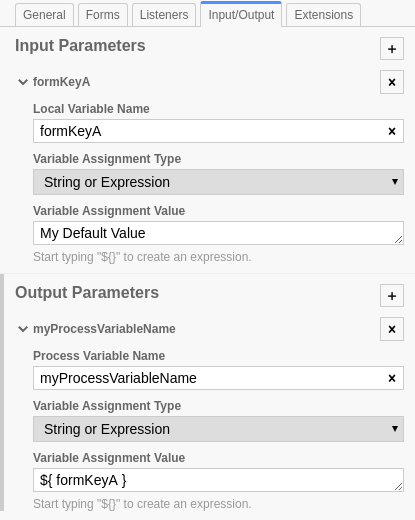
* camunda-forms:deployment:FORM\_NAME.form
* camunda-forms:app:forms/FORM\_NAME.form

When you use [Camunda Platform Run](https://docs.camunda.org/manual/7.15/user-guide/camunda-bpm-run) we recommend the camunda-forms:deployment form key for easy deployment from the Camunda Modeler.



Process Variable Binding

To define a default value for a form field, a process variable with the same name as the form field key needs to be defined. Local variables (e.g. created by defining an [Input Parameter](https://docs.camunda.org/manual/7.15/user-guide/process-engine/variables/#input-output-variable-mapping) for the User Task) take precedence over process variables.

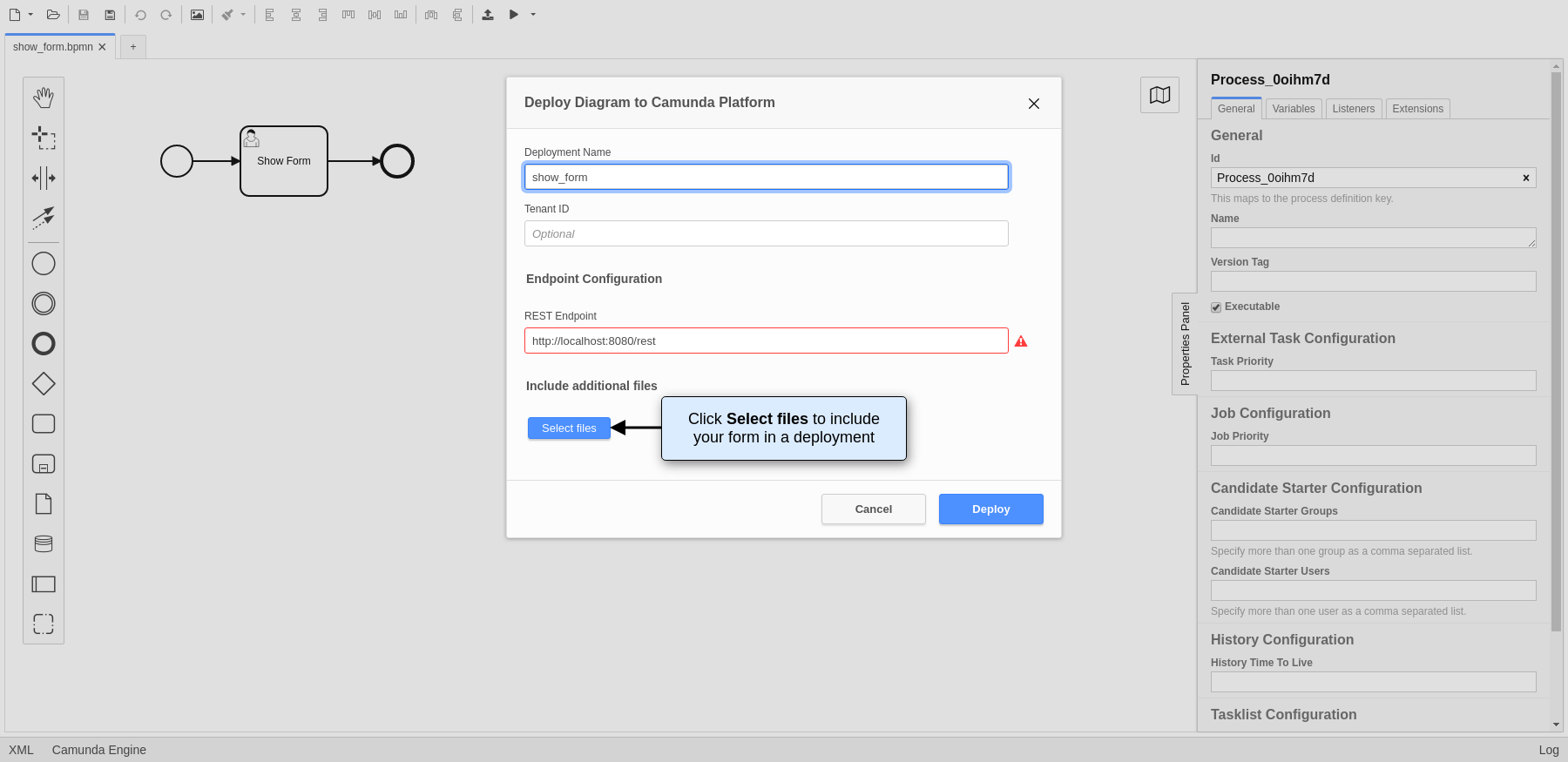


The submitted values of a form are returned as variables to the process engine:

* When a process variable with the same name as the form field key already existed, then its value will be overwritten by the form submission.
* When the User Task has an Input Parameter defined with the same name as the form field key, then this local variable will be used. In this case, an [Output Parameter](https://docs.camunda.org/manual/7.15/user-guide/process-engine/variables/#input-output-variable-mapping) needs to be defined to map the local variable to a process variable for usage in other process elements.
* When no variable exists with the same name as the form field key, then a new process variable will be created and gets the value from the submission.

Deployment

If you want to include your Camunda Form as part of the deployment, then you need to deploy the .form file in the same deployment as the respective .bpmn diagram - for example using the Camunda Modeler.



External Task Forms

Using Task Forms outside of Camunda Tasklist

When embedding the process engine into a custom application, you can use any value in the form key property as a reference to your custom form. This way, your front-end can ensure to render the correct form for each user task.

If you want to call a task form that is not part of your application, you can add a reference to the desired form. The referenced task form will be configured in a way similar to the embedded task form. Open the properties view and enter FORM\_NAME.html as form key. The relevant XML tag looks like this:

<userTask id="theTask" camunda:formKey="app:FORM\_NAME.html"

camunda:candidateUsers="John, Mary"

name="my Task">

Tasklist creates the URL by the pattern:

"../.." + contextPath (of process application) + "/" + "app" + formKey (from BPMN 2.0 XML) + "processDefinitionKey=" + processDefinitionKey + "&callbackUrl=" + callbackUrl;

When you have completed the task, the call back URL will be called.

How To

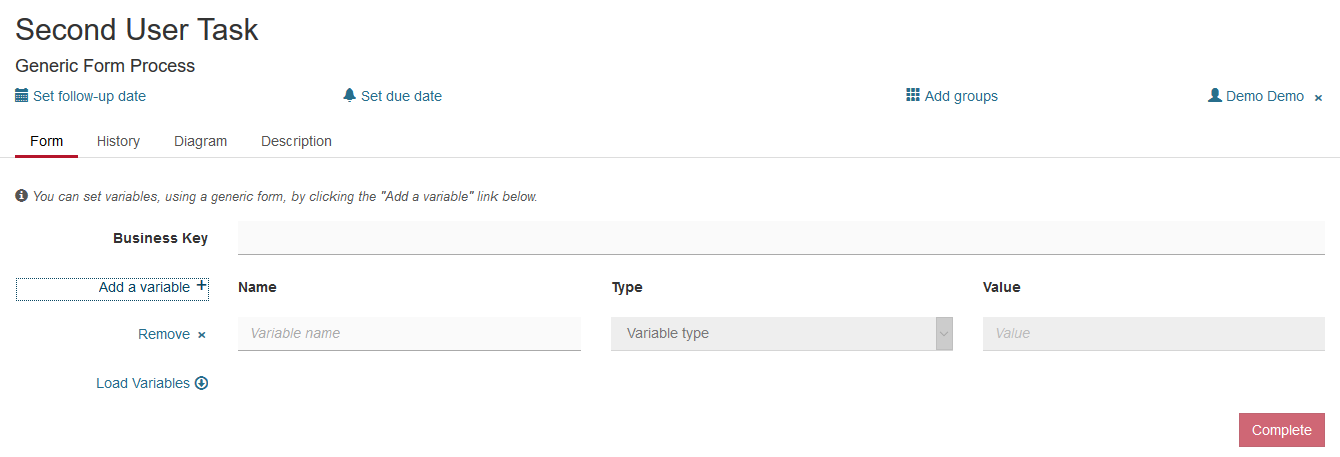
[How to add JSF Forms to your process application](https://docs.camunda.org/manual/7.15/user-guide/task-forms/jsf-task-forms/)

Other Task Forms

These Task forms do not use the form-key attribute to be referenced. They are not recommended for production use and are intended for testing and development purposes.

Generic Task Forms

The generic form will be used whenever you have not added a dedicated form for a [UserTask](https://docs.camunda.org/manual/7.15/reference/bpmn20/tasks/user-task/) or a [StartEvent](https://docs.camunda.org/manual/7.15/reference/bpmn20/events/start-events/).



Hit the *Add a variable* button to add a variable that will be passed to the process instance upon task completion. State a variable name, select the type and enter the desired value. Enter as many variables as you need. After hitting the *Complete* button, the process instance contains the entered values. Generic task forms can be very helpful during the development stage, so you do not need to implement all task forms before you can run a workflow. For debugging and testing this concept has many benefits as well.

You can also retrieve already existing variables of the process instance by clicking the *Load Variables* button.

Generated Task Forms

Camunda Forms or Generated Task Forms?

The feature set of Camunda Forms and Generated Task Forms are similar. For new projects, we recommend to use Camunda Forms, as they offer more flexibility and are easier to create.

The Camunda process engine supports generating HTML task forms based on Form Data Metadata provided in the BPMN 2.0 XML. Form Data Metadata is a set of BPMN 2.0 vendor extensions provided by Camunda, allowing you to define form fields directly in the BPMN 2.0 XML:

<userTask id="usertask" name="Task">

<extensionElements>

<camunda:formData>

<camunda:formField

id="firstname" label="First Name" type="string">

<camunda:validation>

<camunda:constraint name="maxlength" config="25" />

<camunda:constraint name="required" />

</camunda:validation>

</camunda:formField>

<camunda:formField

id="lastname" label="Last Name" type="string">

<camunda:validation>

<camunda:constraint name="maxlength" config="25" />

<camunda:constraint name="required" />

</camunda:validation>

</camunda:formField>

<camunda:formField

id="dateOfBirth" label="Date of Birth" type="date" />

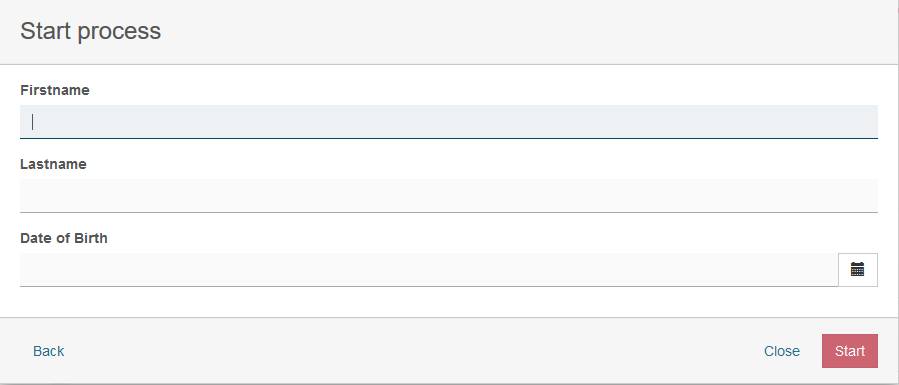
</camunda:formData>

</extensionElements>

</userTask>

Form metadata can be graphically edited using the [Camunda Modeler](http://camunda.org/bpmn/tool/).

This form would look like this in Tasklist:



As you can see, the <camunda:formData ... /> element is provided as a child element of the BPMN <extensionElements> element. Form metadata consists of multiple form fields which represent individual input fields where a user has to provide some value or selection.

A form data can have following attributes:

|  |  |
| --- | --- |
| **Attribute** | **Explanation** |
| businessKey | Id of a form field that will be marked as cam-business-key |

Form Fields

A form field can have the following attributes:

|  |  |
| --- | --- |
| **Attribute** | **Explanation** |
| id | Unique id of the form field, corresponding to the name of the process variable to which the value of the form field is added when the form is submitted. |
| label | The label to be displayed next to the form field. |
| type | The data type of the form field. The following types are supported out of the box:   * string * long * date * boolean * enum |
| defaultValue | Value to be used as a default (pre-selection) for the field. |

Form Field Validation

Validation can be used for specifying frontend and backend validation of form fields. Camunda Platform provides a set of built-in form field validators and an extension point for plugging in custom validators.

Validation can be configured for each form field in the BPMN 2.0 XML:

<camunda:formField

id="firstname" label="First Name" type="string">

<camunda:validation>

<camunda:constraint name="maxlength" config="25" />

<camunda:constraint name="required" />

</camunda:validation>

</camunda:formField>

As you can see, you can provide a list of validation constraints for each form field.

The following built-in validators are supported out of the box:

|  |  |
| --- | --- |
| **Validator** | **Explanation** |
| required | Applicable to all types. Validates that a value is provided for the form field. Rejects 'null' values and empty strings.  <camunda:constraint name="required" /> |
| minlength | Applicable to string fields. Validates the minimum length of text content. Accepts 'null' values.  <camunda:constraint name="minlength" config="4" /> |
| maxlength | Applicable to string fields. Validates the maximum length of text content. Accepts 'null' values.  <camunda:constraint name="maxlength" config="25" /> |
| min | Applicable to numeric fields. Validates the minimum value of a number. Accepts 'null' values.  <camunda:constraint name="min" config="1000" /> |
| max | Applicable to numeric fields. Validates the maximum value of a number. Accepts 'null' values.  <camunda:constraint name="max" config="10000" /> |
| readonly | Applicable to all types. Makes sure no input is submitted for the given form field.  <camunda:constraint name="readonly" /> |

Camunda Platform supports custom validators. Custom validators are referenced using their fully qualified classname or an expression. Expressions can be used for resolving Spring or CDI @Named beans:

<camunda:formField

id="firstname" label="First Name" type="string">

<camunda:validation>

<camunda:constraint name="validator" config="com.asdf.MyCustomValidator" />

<camunda:constraint name="validator" config="${validatorBean}" />

</camunda:validation>

</camunda:formField>

The name attribute must be set to “validator” in order to use custom form field validator.

A custom validator implements the org.camunda.bpm.engine.impl.form.validator.FormFieldValidator interface:

public class CustomValidator implements FormFieldValidator {

public boolean validate(Object submittedValue, FormFieldValidatorContext validatorContext) {

// ... do some custom validation of the submittedValue

// get access to the current execution

DelegateExecution e = validatorContext.getExecution();

// get access to all form fields submitted in the form submit

Map<String,Object> completeSubmit = validatorContext.getSubmittedValues();

}

}

If the process definition is deployed as part of a ProcessApplication deployment, the validator instance is resolved using the process application classloader and / or the process application Spring Application Context / CDI Bean Manager, in case of an expression.