Fix Me! **This page is not fully translated, yet. Please help completing the translation.**
(remove this paragraph once the translation is finished)

DRAFT

JavaScript Quickstart

This wiki-page will describe the most imortant and basic aspects of JavaScript usage in TIM.

1. Importing a script-file

Unlike in normal web-design Javasript and HTML Code HAVE to be seperated in TIM. The HTML code is imported with the process-model while the JavaScript Code is saved in the custom.js file. The custom.js file can be uploaded in the “Administration-Client” in the “Resources”-tab. The file has to be named “custom.js”
This file will be used for the whole client. To implement process-specific functions, the smartform has to have an “initMethod” attribute, which must have a unique name. This happens in the first line of the smartform (HTML code): `<form class="example-process" initMethod="REPLACEME" > [...] </form>`

```
<form class="form-quickstart" data-bootstrap="true"
validationMethod="validateQuickstart" initMethod="initQuickstart"> [...]
</form>
```

2. Aufbau einer Scriptdatei

2.1. Init-method

The init-method must be named like this in the custom.js: `gadget.functions.REPLACEME = function(){ }`

2.1.1. Importing JQuery

Since a samrtform is not structered like a common website, a `document.getElementById("ID");` search doesn't yield the required result. To get the right result, one would have to work with the this-pointer, which can quickly get confusing. To search objects by ID, JQuery is recommended. To be used, it has to be imported first, like this:
This line must be placed at the beginning of the init-method:

```
jq = (this.form.ownerDocument.defaultView!=null) ?
```

```
this.form.ownerDocument.defaultView.jQuery :
this.form.ownerDocument.parentWindow.jQuery;
```

2.1.2. Creating own methods

The naming differs from common webdisign:
For example functions must be have the prefix `gadget.functions.` . Also important: The init-method is executed while the smartorm is loaded. This makes it a good place for returning functions, like filling a textfield with the current date. To realize this function, one can add the following code:

```
//A new date object is created
    var d = new Date;

    //The input-field with the ID "date" is filled with the value of d
(!Every time the samrtform is opened!)
    jq("#date").val(d);
```

2.2. Validate-method

The validate-method is triggered every time the smartform is safed. This happens when somebody presses the “Start process”, “Save” or “Save and finish task” button. Like the name suggests, this function should be used to validate the smartform entries before they are saved. To create a validate method for a process, the smartform needs a “validateMethod” attribute:
`<form class="example-process" initMethod="initQuickstart" validationMethod="CHANGEME"> [...] </form>`

```
<form class="form-quickstart" data-bootstrap="true"
validationMethod="validateQuickstart" initMethod="initQuickstart"> [...]
</form>
```

In this example the validation-method will compare the two email-adresses. If they don't match, the user will be informed via alert and the task will not be finished.
To achieve this, the values of the two input-fields are compared and if they don't match, an alert informs the user and the method returns false. If the validate method returns false, TIM assumes that the validation founf errors and doesn't save the smartform and doesn't finish the task.
The validate-method has to retrun a boolean.

```
gadget.functions.validateQuickstart = function(){
    //If the two adresses don't match, the user gets an alert and the task
is not finished and the changes to the smartform are not saved
    if(jq("#email_1").val() != jq("#email_2").val()){
        alert("The emails don't match!");
        return false;
    }else {
```

```
        //Method returns "true", so TIM assumes that the changes were
correct and saves the smartform/finishes task
        return true;
    }
}
```

3. Process variables

TIM has it's own variables, called process variables. This are matched to the process instance where they were created and can be used with the help of JavaScript. To avoid confusion realted to the this-pointer it is recommended to create a global variable (tp) in the init-method (see example code) which contains the this-pointer. In the example the init-method a function is bound to the button “Safe the adress to the database”. As soon as the button is clicked, the value of the input field “email_1” is saved as a process variable.
The line `tp.entity.mergeLocal(true);` is very important. It functions like a “commit” command, so it is very important for saving changes to the entity. The changes in the entity are applied to the database as soon as the smartform is saved (e.g. somebody clicks the “Save” button). One “commit”-command at the end of multiple changes to variables is enough, since it commits all the changes.

```
    //A function is bound to the button with the ID "compare_adress", that
is triggered by a click
    jq("#compare_adress").on("click", function () {
        //A process variable is created. It is named "email" and has the
value of the input field with the ID "email_1"
        tp.entity.setValue("email", (jq("#email_1").val()));
        //All changes are commited
        tp.entity.mergeLocal(true);
    })
```

The process-variable “email” can now be used for e.g. actionhandler
If one wants to get a process variable, he has to use the following command:
`tp.entity.getValue("CHANGEME");` (See also 4.)

4. Method calls

If one wants to write his own methods, which should be used in the init-method (e.g. to bind them on a button), they have to have the same prefix as the init-method: `gadget.functions.` (See also the example)

In this example the fetching of process variables will be transfered to an extra method.

```
   //A function that will be triggered by a click is bound to the button
with the ID "get_process_variable"
    jq("#get_process_variable").on("click", function () {
        gadget.functions.getVar(jq("#param").val());
    })
```

```
//This function fetches a process variable
```

```
gadget.functions.getVar = function (param) {
    alert(tp.entity.getValue(param));
}
```