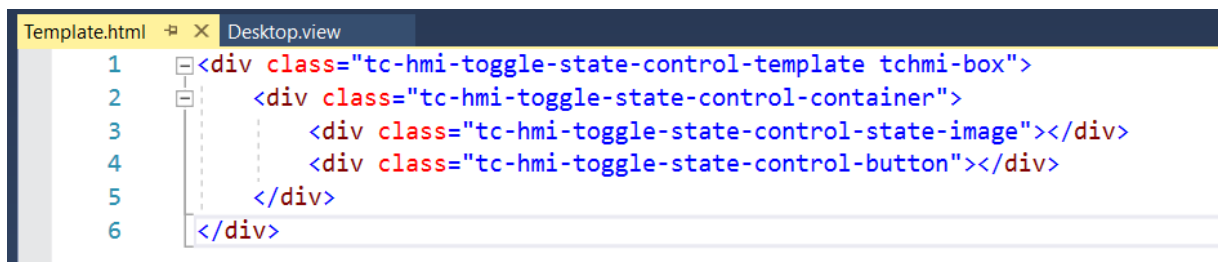


1. Introduction

- Show Power Point "Framework Control Introduction"
- Introduce the final solution, what functionalities does the control have
- Create a new Framework Control named "TcHmiToggleStateControl"
- Give an overview about all files inside the Framework Control
- Build the control
- Link the control with the TcHmiProject
- Instantiate the control on the Desktop.view

2. HTML code

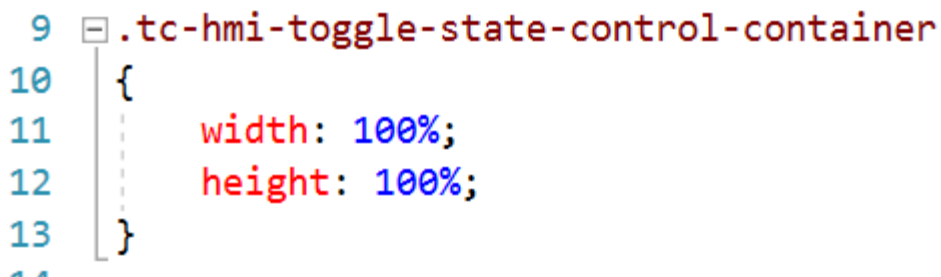
- Show Power Point "Framework Control basics"
- Open Template.html
- Optional file that is not required (elements are referenced inside Source.js)
- Don't use static IDs here (because of multiple instantiation)
- Add the following HTML code:



```
1 <div class="tc-hmi-toggle-state-control-template tchmi-box">
2   <div class="tc-hmi-toggle-state-control-container">
3     <div class="tc-hmi-toggle-state-control-state-image"></div>
4     <div class="tc-hmi-toggle-state-control-button"></div>
5   </div>
6 </div>
```

3. CSS code:

- Explain difference between Style.css on project layer and Style.css in theme folder --> Themes inside Framework Control will be introduced later
- Open Style.css on project layer
- Copy the CSS classes from the Template.html to avoid typing errors
- Add CSS class for the container element:



```
9 .tc-hmi-toggle-state-control-container
10 {
11     width: 100%;
12     height: 100%;
13 }
```

- Add CSS class for the state control button:

```

21  [ ] .tc-hmi-toggle-state-control-button {
22      width: 100%;
23      height: 38px;
24      /* line height for vertical text align*/
25      line-height: 38px;
26      bottom: 0px;
27      position: absolute;
28      text-align: center;
29      vertical-align: middle;
30      border-radius: 5px;
31      margin-top: 2px;
32  }

```

- f. Add CSS hover (mouse over) for the button:

```

31  [ ] .tc-hmi-toggle-state-control-button:hover {
32      cursor: pointer;
33      opacity: 0.7;
34  }
--

```

- g. Add CSS class for the state image:

```

48  [ ] .tc-hmi-toggle-state-control-state-image {
49      width: 100%;
50      height: calc(100% - 40px);
51      top: 0px;
52      position: absolute;
53      background-repeat: no-repeat;
54      background-size: 100% 100%;
55  }
56

```

- h. Open Style.css in base theme folder and add theme dependent style for the button:

```

8  [ ] .tc-hmi-toggle-state-control-button {
9      background-color: grey;
10     color: white;
11 }

```

- i. Build the project, reload the Desktop.view (yellow bar) and show current state in LiveView

4. Reference HTML elements inside JavaScript code:

- a. **Show Power Point "Framework Control Basics - Control lifecycle"**
- b. Open Source.js
- c. Explanations (reference to the shown graphic before)
 - i. Constructor --> Initialization of member variables
 - ii. PrevInit --> Called before the attribute setters are called
 - iii. Init --> Called after the attribute setters are called
 - iv. Attach --> Called after the control is added to the DOM (e. g. when a page is loaded)
 - v. Detach --> Called after the control is removed from the DOM (e. g. when a page is unloaded)
 - vi. Destroy --> Called if the control is destroyed (explicit called by the developer or implicit called by system if no preload is active)
- d. Scroll to __prevInit function and program the following (attention "." is required to find the elements):

```
// get reference to added html elements
this.__elementContainer = this.__elementTemplateRoot.find('.tc-hmi-toggle-state-control-container');
this.__elementStateImage = this.__elementContainer.find('.tc-hmi-toggle-state-control-state-image');
this.__elementButton = this.__elementContainer.find('.tc-hmi-toggle-state-control-button');
```

- e.

```
// set default image
this.__elementStateImage.css({
  'background-image': `url("${this.__basePath}Images/start_normal.svg")`
});
```

- f. Copy the CSS classes from the Template.html to avoid typing errors
- g. Set default state icon (just for demo, commented out later)
- h. Define base path for the state icon folder using API

```
// initialize members
this.__basePath = TcHmi.Environment.getBasePath() + '/Controls/Custom/TcHmiToggleStateControl/';
```

- i. Build project, reload Desktop.view and show changes

5. Use attributes

- a. **Show Power Point "Framework Control Basics - Attributes"**
- b. Open Description.json and scroll to attributes
- c. Copy the default type attribute and change the attributes in the following
- d. Show each property of the attribute step by step
- e. Add attribute for button text:
 - i. Description.json code:

ii.

```
{
  "name": "data-tchmi-button-text",
  "displayName": "ButtonText",
  "propertyName": "ButtonText",
  "propertySetterName": "setButtonText",
  "propertyGetterName": "getButtonText",
  "visible": true,
  "type": "tchmi:general#/definitions/String",
  "category": "Configuration",
  "description": "Text of the switch state button.",
  "requiredOnCompile": false,
  "readOnly": false,
  "bindable": true,
  "heritable": true,
  "defaultValue": null,
  "defaultValueInternal": "Switch state"
},
```

- iii. Add category "Configuration" (not required, but for prioritization of placement in property window):

```
150  [ ] "attributeCategories": [
151  [ ] {
152      "name": "Configuration",
153      "displayPriority": 100
154  }
155  ],
```

- iv. Declare member variable with undefined in the constructor of Source.js
v. Add Setter in Source.js and explain the different steps inside a setter (works also with localized texts, because the binding is resolved by the framework):

vi.

```
154 // setter function for ButtonText attribute
155 TcHmiToggleStateControl.prototype.setButtonText = function (valueNew) {
156     // convert the new value
157     var convertedValue = TcHmi.ValueConverter.toString(valueNew);
158
159     // if converted value is null, get internal default
160     if (convertedValue === null) {
161         convertedValue = this.getAttributeDefaultValueInternal('ButtonText');
162     }
163
164     // skip processing when the value has not changed
165     if (convertedValue === this.__buttonText) {
166         return;
167     }
168
169     // save new value
170     this.__buttonText = convertedValue;
171
172     // Inform the system that the function has a changed result.
173     TcHmi.EventProvider.raise(this.__id + ".onFunctionResultChanged", ["getButtonText"]);
174
175     // process the new value
176     this.__processButtonText();
177 };
```

vii. Add getter in Source.js and explain it:

viii.

```
179 // getter function for ButtonText attribute
180 TcHmiToggleStateControl.prototype.getButtonText = function () {
181     // return the current text
182     return this.__buttonText;
183 };
```

ix. Add processor in Source.js and explain it:

x.

```
185 // process function for ButtonText attribute
186 TcHmiToggleStateControl.prototype.__processButtonText = function () {
187     // set text to html element
188     if (this.__elementButton) {
189         this.__elementButton.text(this.__buttonText);
190     }
191 };
```

xi. Build project, reload Desktop.view and show the text function (also with localized text)

f. Add ShowButton attribute:

i. Description.json code:

ii.

```

116     "name": "data-tchmi-show-button",
117     "displayName": "ShowButton",
118     "propertyName": "ShowButton",
119     "propertySetterName": "setShowButton",
120     "propertyGetterName": "getShowButton",
121     "visible": true,
122     "type": "tchmi:general#/definitions/Boolean",
123     "category": "Configuration",
124     "description": "Show or hide the button.",
125     "requiredOnCompile": false,
126     "readOnly": false,
127     "bindable": true,
128     "heritable": true,
129     "defaultValue": true,
130     "defaultValueInternal": true
131 },

```

iii. Declare member variable with undefined in the constructor

iv. Setter in Source.js:

v.

```

193 // setter function for ShowButton
194 TcHmiToggleStateControl.prototype.setShowButton = function (valueNew) {
195     // convert the new value
196     var convertedValue = TcHmi.ValueConverter.toBoolean(valueNew);
197
198     // if converted value is null, get internal default
199     if (convertedValue === null) {
200         convertedValue = this.getAttributeDefaultValueInternal('ShowButton');
201     }
202
203     // skip processing when the value has not changed
204     if (convertedValue === this.__showButton) {
205         return;
206     }
207
208     // save new value
209     this.__showButton = convertedValue;
210
211     // Inform the system that the function has a changed result.
212     TcHmi.EventProvider.raise(this.__id + ".onFunctionResultChanged", ["getShowButton"]);
213
214     // process the new value
215     this.__processShowButton();
216 };

```

vi. Getter in Source.js:

vii.

```

218 // getter function for ShowButton
219 TcHmiToggleStateControl.prototype.getShowButton = function () {
220     // return the current value
221     return this.__showButton;
222 };

```

viii. Processor in Source.js:

ix.

```

224 // process function for ShowButton
225 TcHmiToggleStateControl.prototype.__processShowButton = function () {
226     // show or hide the button
227     if (!this.__elementButton)
228         return;
229
230     // check variable state
231     if (this.__showButton) {
232         // show the button
233         this.__elementButton.css({
234             'visibility': 'visible'
235         });
236     } else {
237         // hide the button
238         this.__elementButton.css({
239             'visibility': 'collapse'
240         });
241     }

```

x. Build project and show functionality

g. Add State attribute

- i. This attribute is a custom datatype
- ii. Navigate to Schema/TypeDefinitions and add a new JSON schema to the project
"States.Schema.json"
- iii. Schema code:

iv.

```

1  {
2      "$schema": "http://json-schema.org/draft-04/schema",
3      "definitions": {
4          "States": {
5              "$schema": "http://json-schema.org/draft-04/schema",
6              "type": "string",
7              "enum": ["Start", "Stop", "Reset"]
8          }
9      }
10 }

```

v. Description.json: Reference schema under DataTypes:

vi.

```

"dataTypes": [
    {
        "name": "tchmi:framework#/definitions/TcHmiToggleStateControl",
        "schema": "Schema/TypeDefinitions/TcHmiToggleStateControl.Schema.json"
    },
    {
        "name": "tchmi:framework#/definitions/States",
        "schema": "Schema/TypeDefinitions/States.Schema.json"
    }
]

```

- vii. Description.json: Add State attribute and use the new data type

viii.

```
132 {
133   "name": "data-tchmi-state",
134   "displayName": "State",
135   "propertyName": "State",
136   "propertySetterName": "setState",
137   "propertyGetterName": "getState",
138   "visible": true,
139   "type": "tchmi:framework#/definitions/States",
140   "category": "Configuration",
141   "description": "Start state of the control.",
142   "requiredOnCompile": false,
143   "readOnly": false,
144   "bindable": true,
145   "heritable": true,
146   "defaultValue": "Start",
147   "defaultValueInternal": "Start"
148 }
```

- ix. Declare member variable with undefined in the constructor

- x. Setter in Source.js:

```
253 // setter function for State attribute
254 TcHmiToggleStateControl.prototype.setState = function (valueNew) {
255   // convert the new value
256   var convertedValue = TcHmi.ValueConverter.toString(valueNew);
257
258   // if converted value is null, get internal default
259   if (convertedValue === null) {
260     convertedValue = this.getAttributeDefaultValueInternal('State');
261   }
262
263   // skip processing when the value has not changed
264   if (convertedValue === this.__state) {
265     return;
266   }
267
268   // save new value
269   this.__state = convertedValue;
270
271   // Inform the system that the function has a changed result.
272   TcHmi.EventProvider.raise(this.__id + ".onFunctionResultChanged", ["getState"]);
273
274   // process the new value
275   this.__processState();
276 };
```

- xi. Getter in Source.js:

xii.

```

278 // getter function for State attribute
279 TcHmiToggleStateControl.prototype.getState = function () {
280     // return the current value
281     return this.__state;
282 };

```

xiii. Processor in Source.js:

xiv.

```

272 // process function for State attribute
273 TcHmiToggleStateControl.prototype.__processState = function () {
274
275     // check if the reference to image is valid
276     if (!this.__elementStateImage)
277         return;
278
279     // get the right image
280     var image = "";
281
282     if (this.__state === "Start") {
283         image = "start_normal.svg";
284     } else if (this.__state === "Stop") {
285         image = "stop_normal.svg";
286     } else {
287         image = "reset_normal.svg";
288     }
289
290     // set the image
291     this.__elementStateImage.css({
292         'background-image': `url("${this.__basePath}Images/${image}")`
293     });

```

xv. Build project and show functionality

6. Implement toggle function of the button

a. Declare StateOrder (sequence of the state machine) in the constructor:

```

// initialize members
this.__basePath = TcHmi.Environment.getBasePath() + '/Controls/Custom/TcHmiToggleStateControl/';
this.__stateOrder = ["Start", "Stop", "Reset"];

```

b. Add internal function "nextState" under all Setter/Getter/Processors (internal functions are marked with __myFunction)

c. Implement the function:

```

351 // internal function, that toggles to the next state in the state machine
352 TcHmiToggleStateControl.prototype.__nextState = function () {
353
354     // get the current state index from state order
355     var currentStateIndex = this.__stateOrder.indexOf(this.__state);
356
357     // switch to the next state
358     var nextStateIndex = currentStateIndex + 1;
359
360     // if limit is reached, set state to zero
361     if (nextStateIndex === this.__stateOrder.length + 1) {
362         nextStateIndex = 0;
363     }
364
365     // set the next state
366     this.setState(this.__stateOrder[nextStateIndex]);
367 };

```

d.

- e. Call internal function in the attach of the Framework Control
 - i. Safe reference of this (the control) for other scopes (callback functions)
 - ii. Add jQuery event listener

```

// reference to this control for callback functions with other scope
var $this = this;

```

```

// register onClick listener for the button
iii. this.__elementButton.on('click', function (e) {
    // call the state switch
    // it is required to use the saved reference here
    $this.__nextState();
});

```

- iv. Remove event listener in detach function (to avoid memory leaks):

```

// remove the onClick listener for the button
v. this.__elementButton.off('click');

```

- f. Build project, reload Desktop.view and show functions in LiveView (press the button, the states should switch regarding the state machine)

7. Events: onStateChanged

- a. Show Power Point "Framework Control Basics - Events"
- b. Open Description.json and add "onStateChanged" event:

```

186   "events": [
187     {
188       "name": ".onStateChanged",
189       "displayName": ".onStateChanged",
190       "visible": true,
191       "displayPriority": 30,
192       "category": "Control",
193       "description": "This event is fired when the state changed.",
194       "heritable": true
195     }
196   ],

```

c.

d. Open Source.js and raise event in the `__processState()` function (at the end):

```

307   // raise "onStateChanged" event
308   TcHmi.EventProvider.raise(this.getId() + ".onStateChanged");

```

e.

f. Build project, reload Desktop.view and configure a sample action for the event: Embedded JS with `"alert("Test")"`

g. Open live view and show functionality

h. Optional: Show how to raise the event with event parameters (Evaluation only possible in code behind at the moment, future also in Action and Conditions editor):

```

310   // optional: parameters in raise event
311   // parameter is the data object in callback function
312   //TcHmi.EventProvider.raise(this.getId() + ".onStateChanged", this.__state);

```

i.

8. Functions

a. **Show Power Point "Framework Control Basics - Functions"**

b. Function without transfer parameter: `toggleState`

i. This functions triggers the state change functionality of the control from outside

ii. Open Description.json and add the following code

```

156   "functions": [
157     {
158       "name": "toggleState",
159       "displayName": "toggleState",
160       "visible": true,
161       "description": "This function toggles to the next state in the state machine.",
162       "category": "Actions",
163       "params": [],
164       "type": null,
165       "heritable": false
166     }

```

iii.

iv. Add function in Source.js and call internal function `"__nextState"`:

```

315 // function "toggleState"
316 TcHmiToggleStateControl.prototype.toggleState = function () {
317
318 // proceed the same functionality as the button inside the control
319 this.__nextState();
320 };

```

c. Function with transfer parameter: toggleToState

- i. This function sets a specific state give by the transfer parameter
(this is only a sample, the function is not really required, because you can call the setter of state instead)
- ii. Open Description.json and add the following code:

```

167 {
168   "name": "toggleToState",
169   "displayName": "toggleToState",
170   "visible": true,
171   "description": "This function toggles to a specific state.",
172   "category": "Actions",
173   "params": [
174     {
175       "name": "newState",
176       "displayName": "newState",
177       "description": "This is the parameter for the new state.",
178       "type": "tchmi:framework#/definitions/States",
179       "visible": true
180     }
181   ],
182   "type": "tchmi:general#/definitions/Boolean",
183   "heritable": false
184 }

```

iv. Open Source.js and implement the function (with check of the transfer parameter):

```

322 // function "toggleToState"
323 TcHmiToggleStateControl.prototype.toggleToState = function (newState) {
324
325 // convert the parameter to string
326 var convertedParameter = TcHmi.ValueConverter.toString(newState);
327
328 // check if the value is valid
329 if (convertedParameter === null || convertedParameter === "") {
330 // value is not valid, stop proceeding the function
331
332 // return value of the function
333 // this can be evaluated in code behind on project layer
334 // not in Actions & Conditions editor at the moment
335 return false;
336 }
337
338 // set the new state
339 this.setState(convertedParameter);
340
341 // exit function call with success
342 return true;
343 };

```

- vi. Explain return values (Evaluation only possible in code behind at the moment, in future also in Actions & Conditions editor)
- d. Build project, reload Desktop.view
 - i. Drop two TchmiButtons
 - ii. Button 1: call toggleState
 - iii. Button 2: call toggleToState (Stop)
 - iv. Open LiveView and show functionality

9. Zugriffsrechte

- a. Show Power Point "Framework Control Basics - Access Rights"
- b. Access right: allowStateChange (allows / disallows the change of the state)
- c. Description.json:

```

197   "access": [
198     {
199       "name": "allowStateChange",
200       "displayName": "allowStateChange",
201       "description": "This right allows the user to change the state.",
202       "visible": true,
203       "defaultValueInternal": null
204     }
205   ],

```

- d. Open Style.css on project layer
 - i. Comment in the code at the end of the file (default styles for operate forbidden)
 - ii. These properties are active, if the user has no operate rights for the control
 - iii. Copy the selector of the button class and add the disallowed operate styles behind it:

```

36  .tc-hmi-toggle-state-control-button.tchmi-control-operate-disallowed::after {
37      content: '';
38      position: absolute;
39      top: 0px;
40      left: 0px;
41      width: 100%;
42      height: 100%;
43      background-color: rgba(255, 255, 255, 0.5);
44      z-index: 100;
45      cursor: not-allowed;
46  }

```

- e. Open Source.js and scroll to "__processShowButton"
 - i. Add the following right query under the existing code
 - ii. Switch classes depending on the rights:

```

243
244
245
246
247
248
249
250

```

```

// check if the user is allowed to change the state
if (Tchmi.Access.checkAccess(this, 'allowStateChange') === true) {
    // user is allowed to change, remove disallowed class
    this.__elementButton.removeClass('tchmi-control-operate-disallowed');
} else {
    // user is not allowed to change, add disallowed class
    this.__elementButton.addClass('tchmi-control-operate-disallowed');
}

```

- f. Scroll to internal function "__nextState" and add the following right query at the begin:

```
351 // internal function, that toggles to the next state in the state machine
352 TcHmiToggleStateControl.prototype.__nextState = function () {
353
354     // check if the user is allowed to change the state
355     if (TcHmi.Access.checkAccess(this, 'allowStateChange') !== true) {
356         // user is not allowed to change the state
357         return;
358     }
```

- g. Scroll to ToggleToState function and add the following right query before the set of the state:

```
338 // check if the user is allowed to change the state
339 if (TcHmi.Access.checkAccess(this, 'allowStateChange') !== true) {
340     // user is not allowed to change the state
341     return false;
342 }
343
344 // set the new state
345 this.setState(convertedParameter);
346
347 // exit function call with success
348 return true;
349 };
```

- h. Build project, reload Desktop.view
- i. Add new user group "Test"
- j. Add new user "Tester" which is a member of the group "Test"
- k. Disallow the StateChange for the Test group
- l. Publish project, login as a Tester --> User has not the rights to switch the state, see also the different styles