UI-Builder: UI bindings

Michael Heck, SICK AG, Germany - July 1, 2019, Updated: September 18, 2020

Introduction

Goal

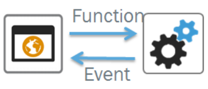
After completing this tutorial you will be able to connect user interface elements to functionality within other sources. For example to set and show a variable value from a script in the user interface

Prerequisites

- How to generate interfaces (Serves), see Tutorial "Multiple Apps"
- How to use the UI-Builder to add elements (layout elements and controls), see tutorial "Application specific user interface"

Theory

Serves (See tutorial "Multiple Apps")



- Serves are the measure to create a public interface of an App
- Serves are defining the capabilities of an App
- Serves are announced in the App Manifest
- Functions vs. Events
    - A Function served in an app can be called from another app or component. "A call into the app."
    - An Event served by an app can be notified in this app and other apps or components can register to the event. "A notification from the app."

Bindings

- A Binding connects a property / event of a control to a value served by the device. Multiple properties can be used. Note:
    - The available properties can differ for each control and address - for example - the value of a control or its visibility
- Bindings can be made to different data sources.
    - Served function (Crown-binding)
    - Served event (CrownEDPWS-binding)
    - SOPAS parameter (Sopas-binding)
    - UI internal variable (Internal-binding) - Currently only available in Code view
- A binding has a direction in which the data flows (get/set)

Binding options

A binding defines the behavior of the UI control in regards to the values of the data source.
Following use-cases are the most prominent.

- An action on the UI should trigger a (re-)action of the application.
    - Example: Crown-binding (set) on submit event of Button
- A value set in the UI should be set and applied in the application.
    - Example: Crown-binding (set) on change event of NumericField
- The current value of the application should be reflected in the UI after reload.
    - Example: Crown-binding (get) on value property of NumericField
- A value generated in the application should be shown in the UI.
    - Example: CrownEDPWS-binding (get) on value property of NumericField

Workflow overview

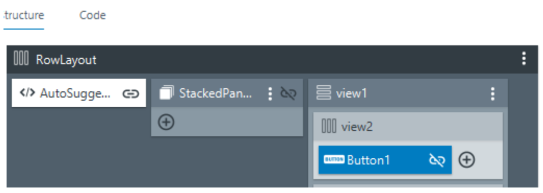
Steps:

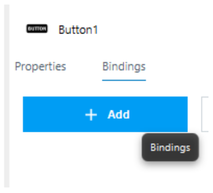
1. Serve function / event in script and app manifest
2. Select control in UI-Builder
3. Open binding dialog
4. Add the relevant bindings for the properties and use-cases
    1. Select property / event
    2. Select type
    3. Choose available matching served function / event
    4. Adapt options to suit the binding needs

Binding Dialog

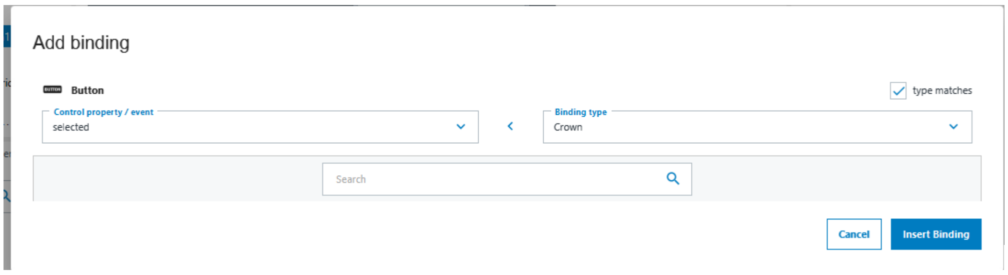
Create a new binding

Make sure, that you are connected to the device and that the apps are synchronised.
To create a new binding, select a control and go to its Bindings tab.



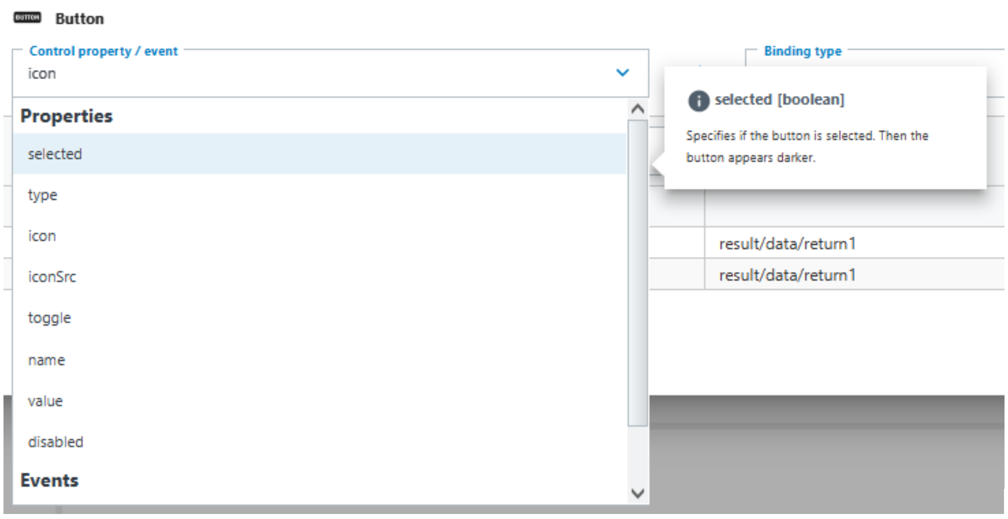


To add a new binding, click the "+ Add" button. The binding dialog will open.



Add bindings

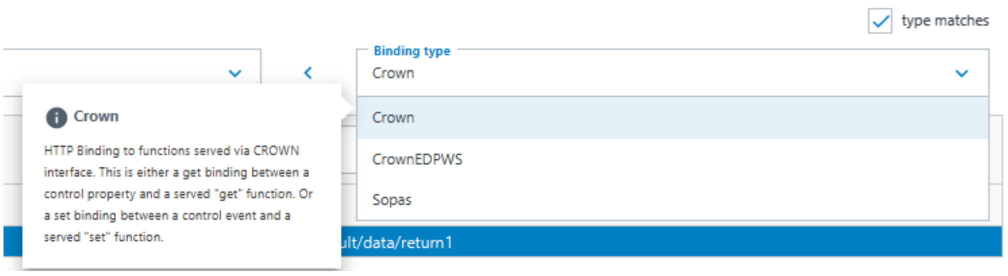
1. To select the property or event you want to bind to, open the Control property / event dropdown menu. A list showing the bindable elements is shown. They are different for each control. A tool-tip shows the description of each.



selected [boolean]

Specifies if the button is selected. Then the button appears darker.

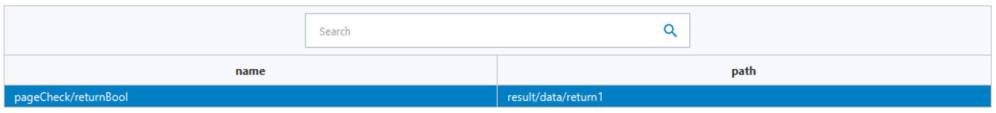
2. Choose the binding type. This is the data source/target for the binding. All available binding types for selected property / event are shown. Choose "Crown" if the source is a served function, CrownEDPWS if the source is a served event and Sopas if the source is a cid parameter.



Crown

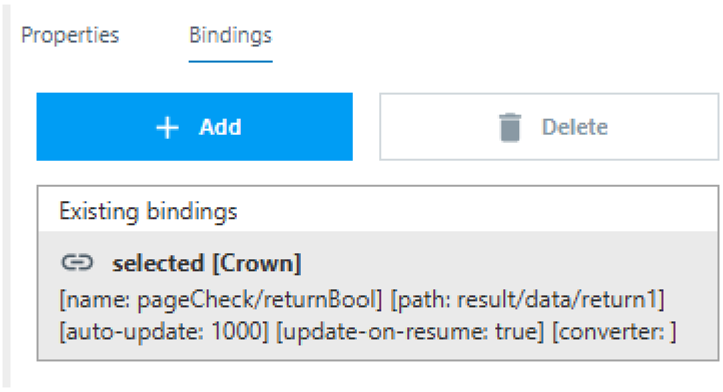
HTTP Binding to functions served via CROWN interface. This is either a get binding between a control property and a served "get" function. Or a set binding between a control event and a served "set" function.

3. Depending on the selected property and target, all available and suitable bindings are offered by clicking into the “Select ... binding” field.



name	path
pageCheck/returnBool	result/data/return1

4. After pressing "Insert Binding", a basic binding is created. You can click on it to edit it.



Existing bindings

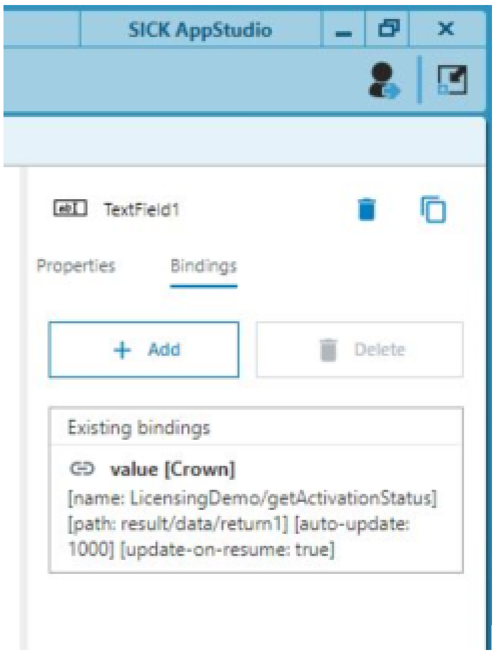
selected [Crown]
[name: pageCheck/returnBool] [path: result/data/return1] [auto-update: 1000] [update-on-resume: true] [converter:]

Note:
By default only Binding types and bindings are listed which source/target definition is matching to the selected property/event. De-select "filter by type" in order to show all types and to list all binding targets. E.g. a NumericField value property cannot be be bound to a function which returns a string directly. Therefore it is not shown in the default selection. But a binding can still be created by de-selecting the type filter and choose the according served function. As this would fail due to a mismatch in datatypes during run-time, a converter function is required in this case.

Edit bindings

You can edit the properties of the binding when you click it in the "Existing bindings" list.

Typical parameters:
1. name: name of the function, event or parameter this binding targets
2. path: path within the topic (e.g.: if a complex data structure is requested)
3, result-path: path to the result of the topic
4. param-path: path to the parameters of the topic
5. auto-update: interval in ms in which the UI element triggers this binding. Set to 0 to disable.
6. update-on-resume: enable for binding to trigger once the page is loaded in the browser
7. auto-commit: enable automatic committing of the binding target
8. converter: a javascript function to be called when the binding receives or emits a value.



Existing bindings

value [Crown]
[name: LicensingDemo/getActivationStatus] [path: result/data/return1] [auto-update: 1000] [update-on-resume: true]

Primary binding options

Get a Value - retrieving a value from the data source

- Polling source for recognizing changes
- Retrieving value once - on page (re-)load

Set a Value - sending a value to the data target

- User input in the UI should be reflected in the target

On Value changed - register to events and reflect the notification

- Value in the UI should be updated when changed in the source

Binding standard use cases

Button

Use case: pressing a button should execute a script function, or toggle a state.

Most relevant serves:

- Function without parameter to be called on button click

and/or

- Function with a Boolean parameter to be called on toggle switch

and/or

- Function with a Boolean return value to update the state of the toggle switch

Specific options:

- selected: toggle state of the button
- toggle: enable/disable toggle behaviour of the button
- icon: icon to display on the button
- type: specifies button style

Exemplary bindings:

#	Property / Event	Target	Options [default]	Type (direction)	Information	Comments
1	submit	function	• Auto-commit [true]	set	Calls the target function when the button is clicked	
(2)	change	function	• Auto-commit [true]	set	Calls the target function when the button is clicked with current toggle state	Only meaningful if used along with toggle functionality
(3)	selected	function	• Auto-update [0] • Update-on-r [true]	get	Gets the toggle state of the button from the script when page is loaded	Only meaningful if used along with toggle functionality

(4)	selected	event		get	Sets the toggle state of the button from the script	Only meaningful if used along with toggle functionality

NumericField

Use case: NumericField should be used to set numerical values in the script or to display them. When reloading the page the latest value in the script should be shown in the UI.
Most relevant serves:

- Function to set value with int / float parameter to be called after UI input

and

- Function to get value with int / float return to get current value on page reload

and

- Event with int / float parameter, which is notified when the value changes in the script

Specific options:

- type: specifies control style
- group-separator: specifies whether values are separated with commas or dots.
- decimal-separator: specifies if comma or dot is used as separator
- format-pattern: specifies accuracy of the display
- unit: specifies the size of incremental changes to the value

Exemplary bindings:

#	Property / Event	Target	Options [default]	Type (direction)	Information	Comments
1	change	function	• Auto-commit [true]	set	Calls the target function including the current control value as a parameter after an input has been entered	
2	value	function	• Auto-update [0] • Update-on- [true]	get	Calls the target function when (re-)loading the page and its return value is reflected in the control	
3	value	event		get	Listens for notifications of the source event and its parameter value is reflected in the control.	

ValueDisplay

Use case: show text in UI

Most relevant serves:

- Event with string parameter, which is notified when the value changes in the script

Specific options:

- label: specifies name of the ValueDisplay
- unit: specifies unit that is displayed at the end of the text
- value: displayed text

Exemplary bindings:

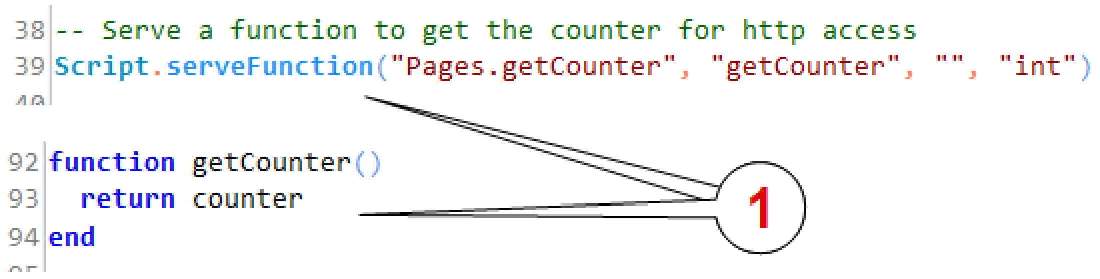
#	Property	Target	Options [default]	Type (direction)	Information	Comments
1	value	event		get	Listens for notifications of the source event and its parameter value is reflected in the control	

Understand bindings with the help of samples

Open the "Pages" sample app to understand, how to bind to *setters*, *getters* and events

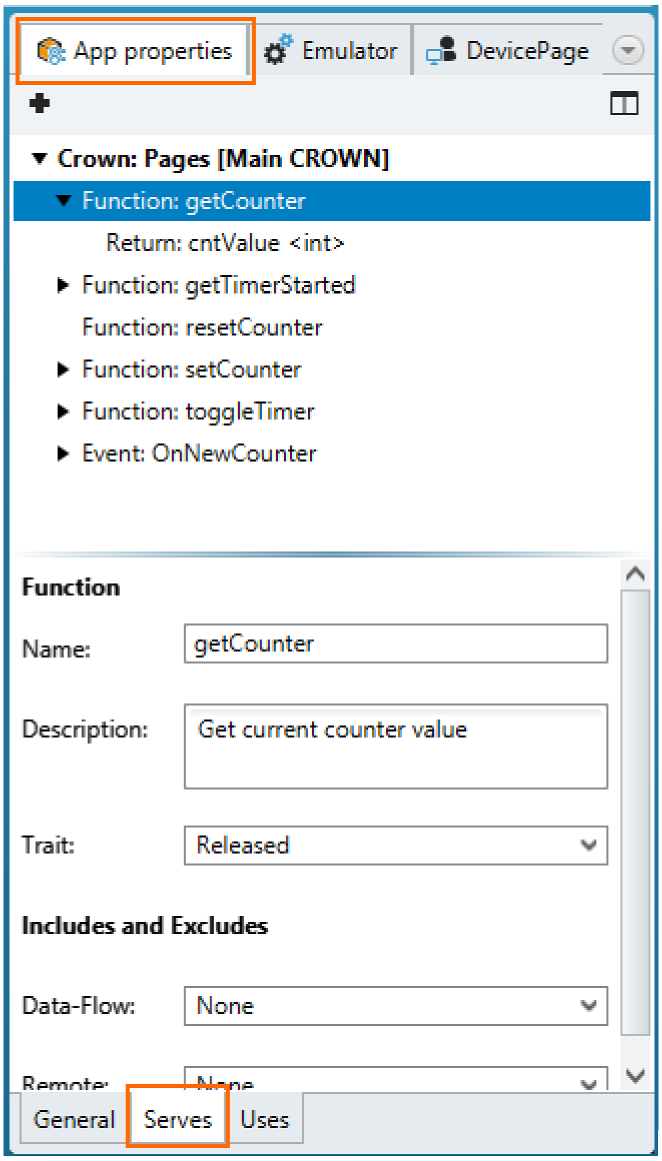
Getter – From device to web-page (1)

1. Define and serve a *getter* function in the script



```
-- Serve a function to get the counter for http access
Script.serveFunction("Pages.getCounter", "getCounter", "", "int")

function getCounter()
  return counter
end
```

Create new *serve* in the "App properties" tab to "publish" the served function



Crown: Pages [Main CROWN]
- Function: getCounter
  - Return: cntValue <int>
- Function: getTimerStarted
- Function: resetCounter
- Function: setCounter
- Function: toggleTimer
- Event: OnNewCounter

Function

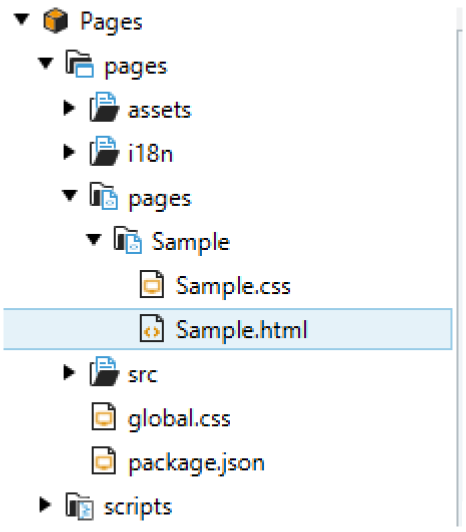
Name:	getCounter
Description:	Get current counter value
Trait:	Released

Includes and Excludes

Data-Flow:	None
Remote:	None

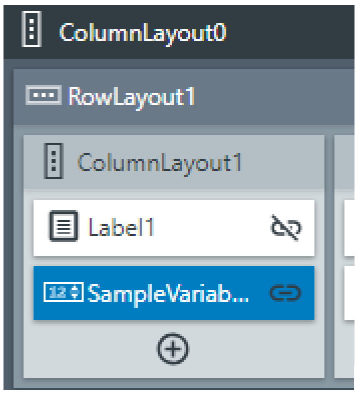
Getter – From device to web-page (2)

1. Open the page of the sample

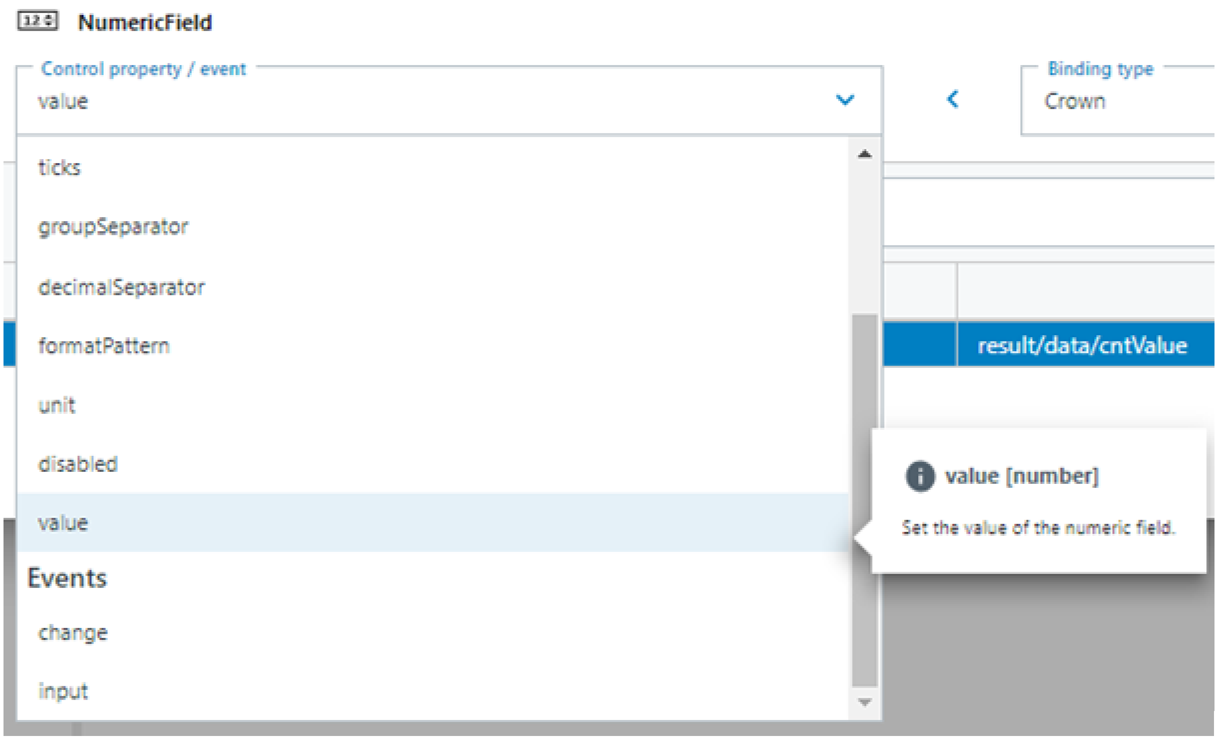


```
Pages
  pages
    assets
    i18n
    pages
      Sample
        Sample.css
        Sample.html
    src
    global.css
    package.json
  scripts
```

2. To bind the *getter* value from the device, click on the "chain link" button to add a binding or go through the bindings tab (see previous paragraphs)



3. In the *Add Binding* dialog, first select the property, to which the control should be bound to


NumericField

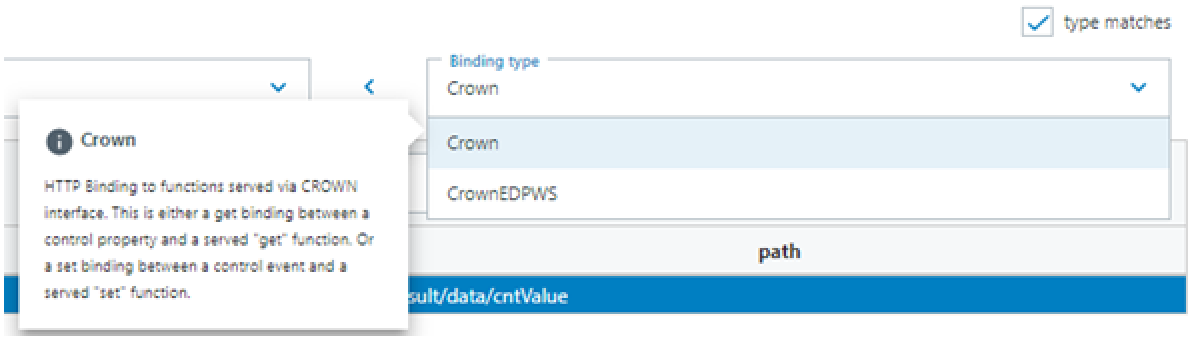
Control property / event: value

Binding type: Crown

value [number]

Set the value of the numeric field.

4. Next, select “Crown” to get the list of served functions

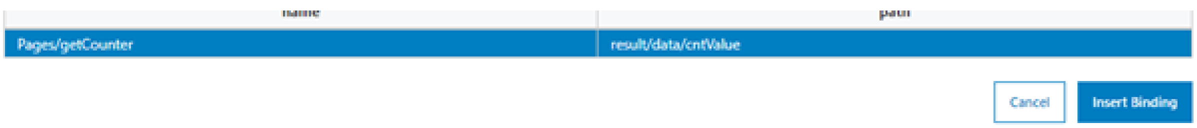

Crown

HTTP Binding to functions served via CROWN interface. This is either a get binding between a control property and a served "get" function. Or a set binding between a control event and a served "set" function.

Binding type: Crown

✓ type matches

5. Select the respective served function e.g. "Pages / getCounter" and click "Insert binding"

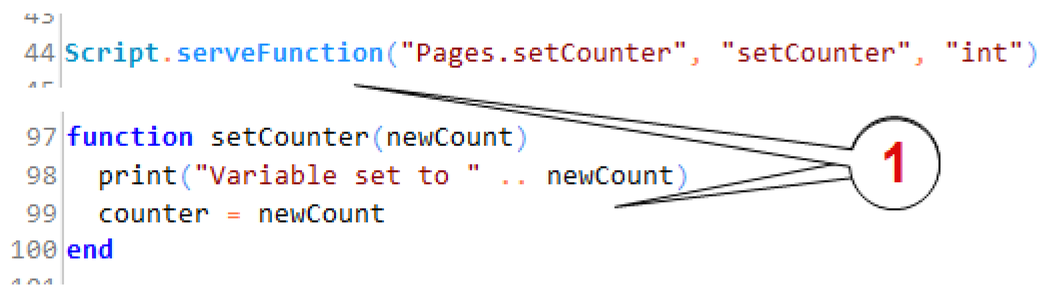


name	path
Pages/getCounter	result/data/cntValue

Note: if the serve is not listed, check the connection and uncheck the *Type matches* checkbox. The app will be restarted after "publishing" the serve.

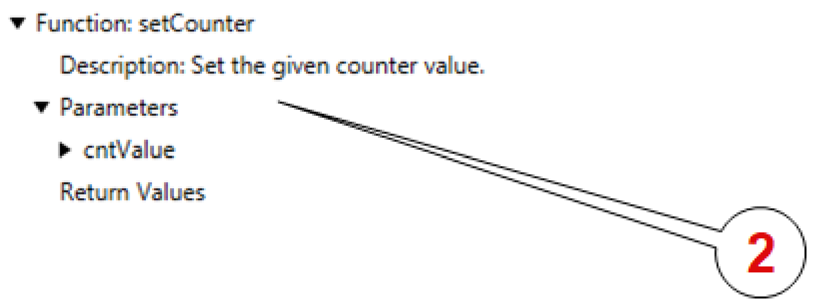
Setter – From web-page to device

1. Define and serve a *setter* function in the script



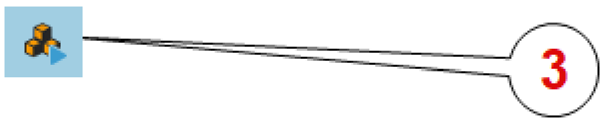
```
Script.serveFunction("Pages.setCounter", "setCounter", "int")

function setCounter(newCount)
  print("Variable set to " .. newCount)
  counter = newCount
end
```

2. Create a new serve in the "App properties" dialog to "publish" the served function

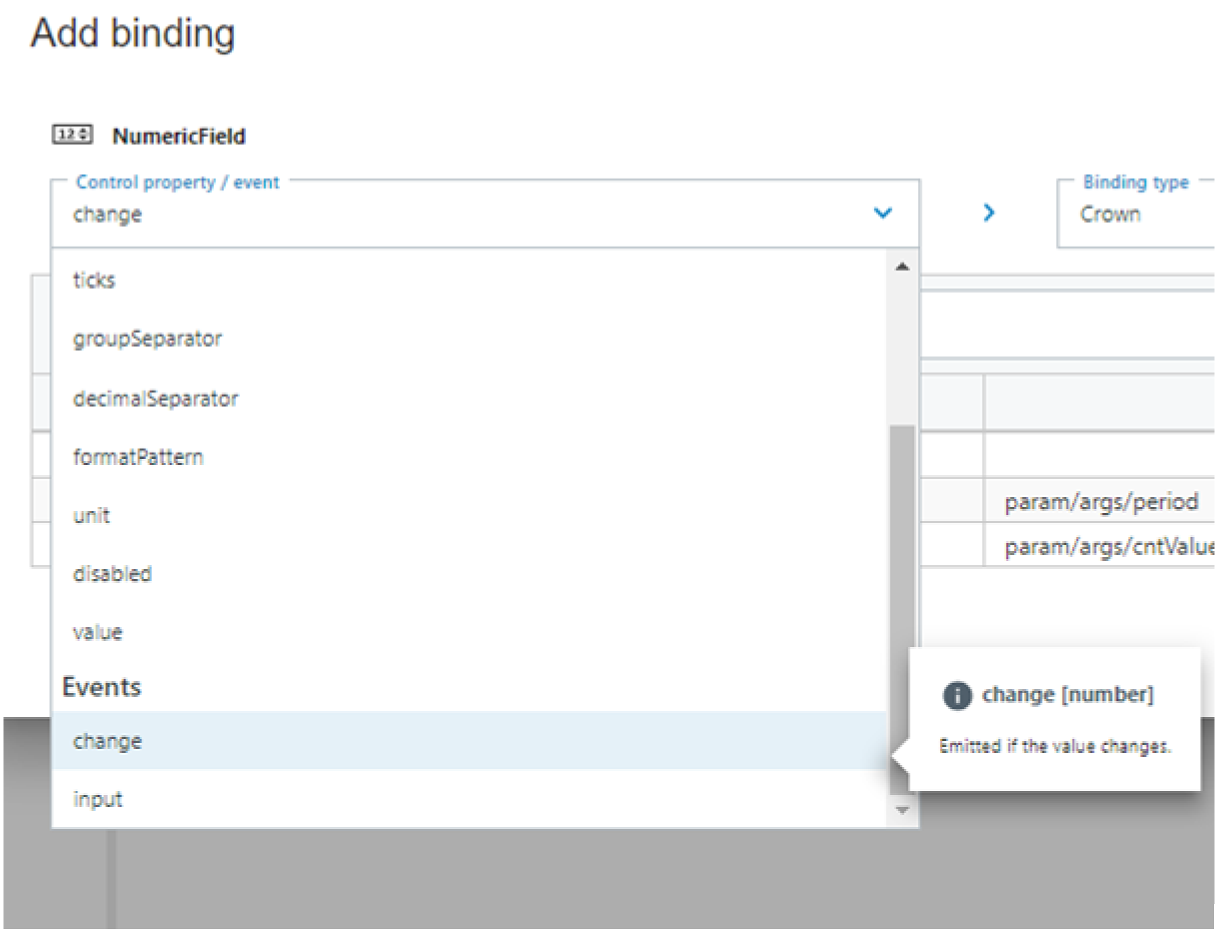


```
▼ Function: setCounter
    Description: Set the given counter value.
  ▼ Parameters
    ▶ cntValue
    Return Values
```

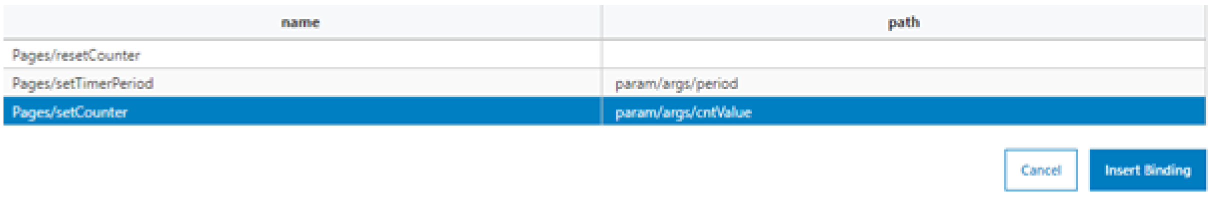
3. To make the serve available, restart the app



4. Open the control's binding dialog, select the property



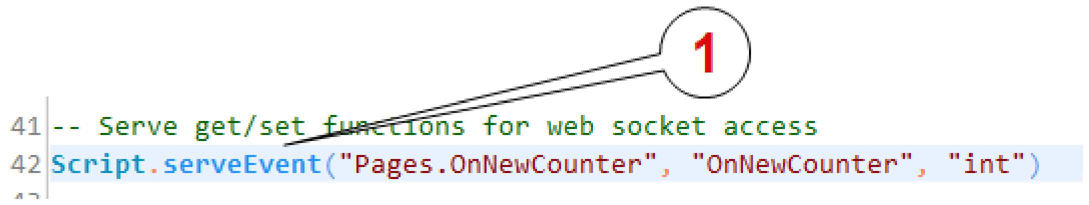
5. … and then select a binding target and click "Insert binding"



name	path
Pages/resetCounter	
Pages/setTimerPeriod	param/args/period
Pages/setCounter	param/args/cntValue

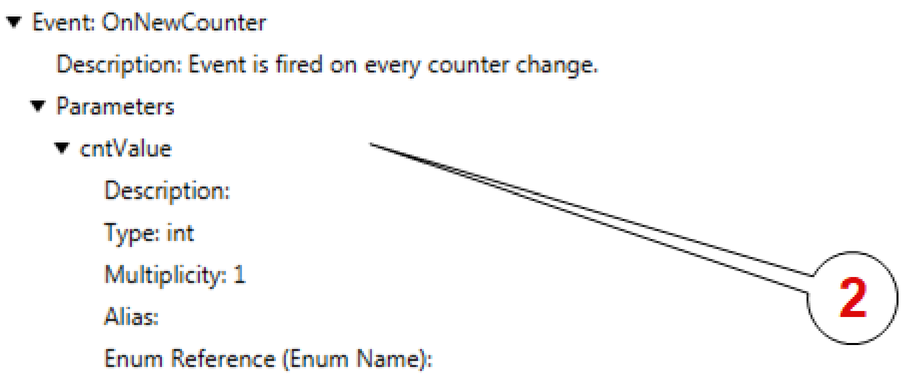
Event - from device to web-page

1. The served event “OnNewCounter” is notified periodically by a timer and publishes the new counter value (see "Pages" app for details)

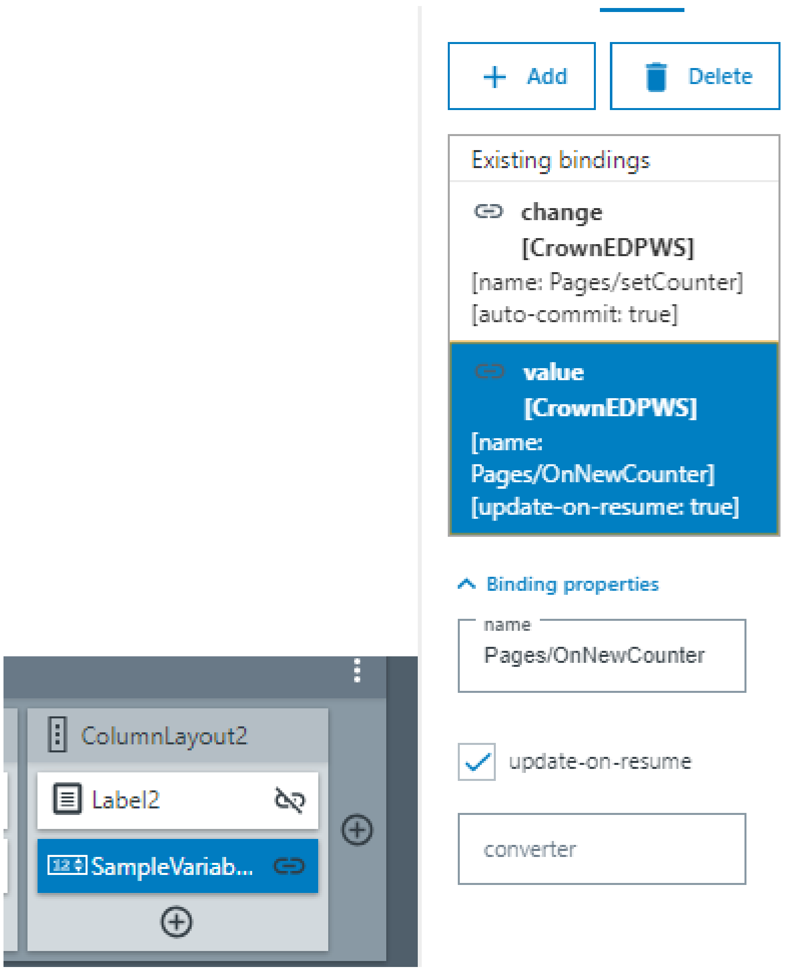


```
-- Serve get/set functions for web socket access
Script.serveEvent("Pages.OnNewCounter", "OnNewCounter", "int")
```

2. It is added as a serve to the app properties ...


▼ Event: OnNewCounter
Description: Event is fired on every counter change.
▼ Parameters
▼ cntValue
Description:
Type: int
Multiplicity: 1
Alias:
Enum Reference (Enum Name):

3. ... and defined as a binding to the text-field "Sample Variable (event)"



change [CrownEDPWS] [name: Pages/setCounter] [auto-commit: true]

value [CrownEDPWS] [name: Pages/OnNewCounter] [update-on-resume: true]

name: Pages/OnNewCounter

✓ update-on-resume

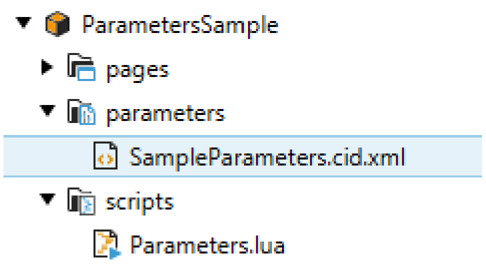
converter

Note: if events are used, which are triggered only occasionally, no default value will be presented when loading the web page until the first event is fired. To avoid having no initial value, add a second binding to a getter function to the same property with update-on-resume selected.

SOPAS parameters - Binding directly to parameters specified in cid parameters

Open the "Parameters" sample app to understand how to bind to SOPAS variables

1. App specific parameters are defined in the "SampleParameters.cid.xml" file

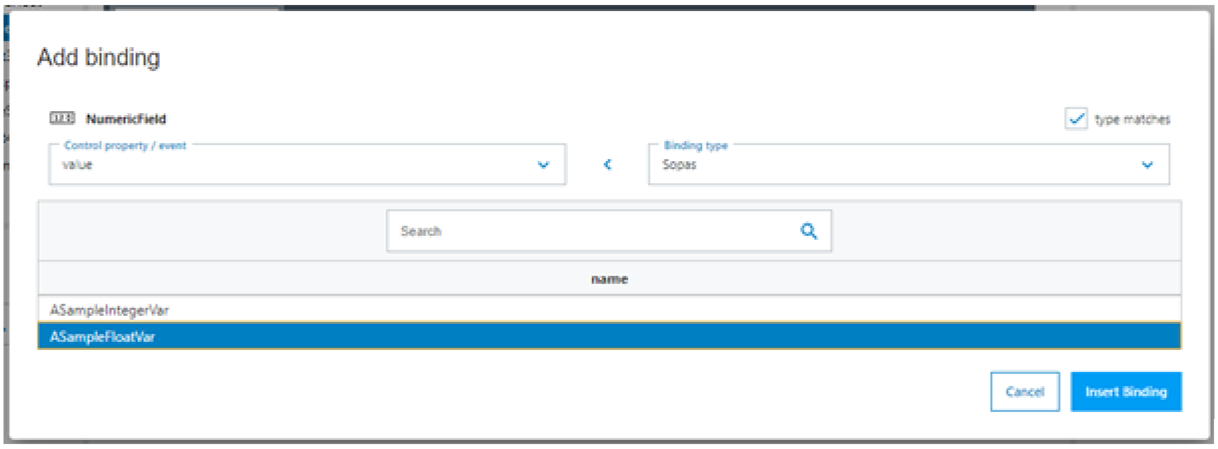


```
▼ ParametersSample
  ▶ pages
  ▼ parameters
      SampleParameters.cid.xml
  ▼ scripts
      Parameters.lua
```

2. Run the app using the Emulator

3. Open the binding dialog of the respective control which should bind the parameter to (e.g. the **NumericField**), then select the Sopas binding

4. Select "ASampleFloatVar" and click "Insert binding"



Add binding

NumericField ✓ type matches

Control property / event: value

Binding type: Sopas

Search

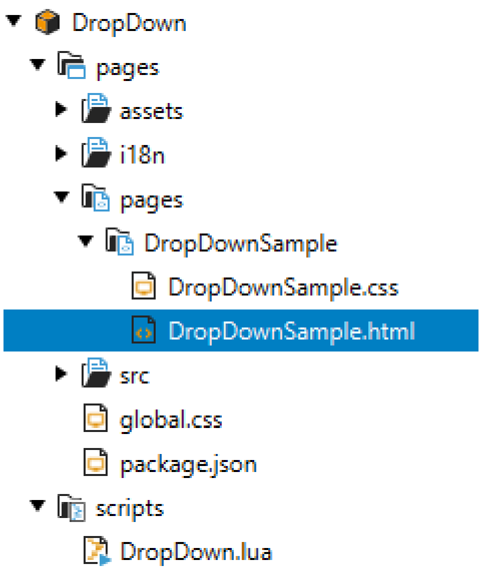
name
ASampleIntegerVar
ASampleFloatVar

Internal variables - Binding to UI internal variables

Internal bindings are direct bindings between two UI elements locally in the UI.
These can currently only be specified manually in code view.

Open the "DropDown" sample app to understand how to bind to internal variables

1. Open the page file

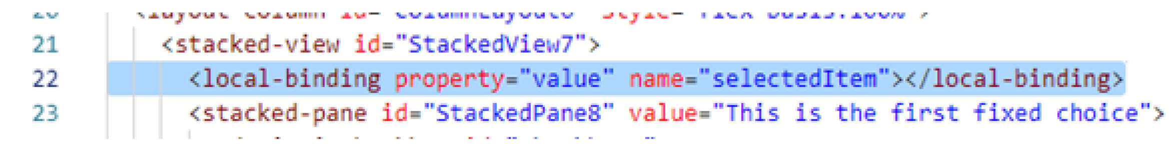


```
▼ DropDown
  ▼ pages
    ▶ assets
    ▶ i18n
    ▼ pages
      ▼ DropDownSample
          DropDownSample.css
          DropDownSample.html
    ▶ src
      global.css
      package.json
  ▼ scripts
      DropDown.lua
```

2. Open the UI-Builder in Code view and find the code of the **StackedView** element.

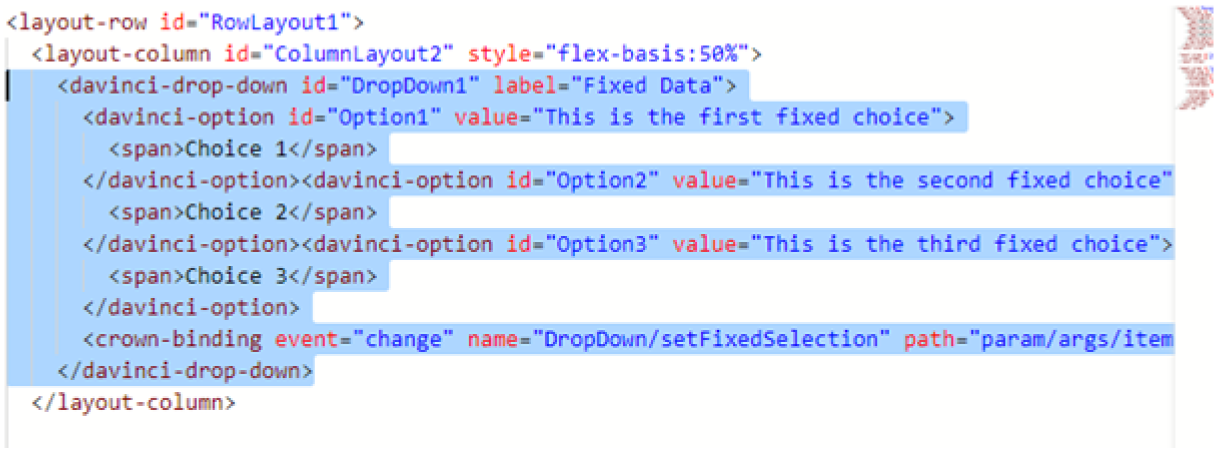
3. Adding a binding to the internal variable, has to be done manually. It can be done by inserting the following line in the **stacked-view** element code:
<local-binding property="value" name="<name of the variable>"></local-binding>

where <name of the variable> is the internal variable used to change the displayed element, e.g. *selectedItem*.



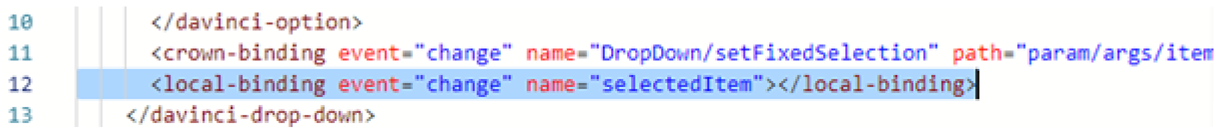
```
  <stacked-view id="StackedView7">
    <local-binding property="value" name="selectedItem"></local-binding>
    <stacked-pane id="StackedPane8" value="This is the first fixed choice">
```

4. To provide data for this internal variable, a control (e.g. **DropDown**) can be bound to it. Therefore, find the code of the **DropDown** control “Fixed Data”



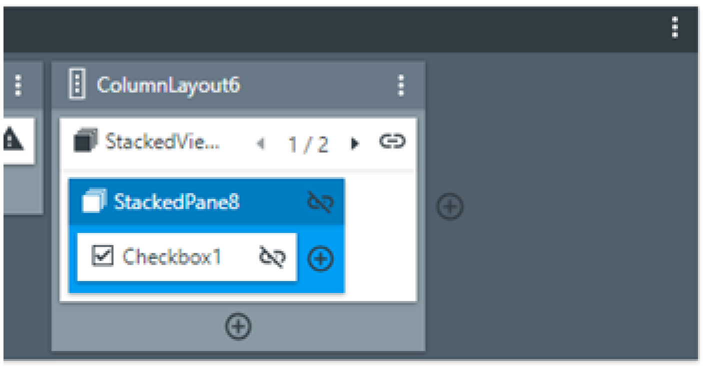
```
<layout-row id="RowLayout1">
  <layout-column id="ColumnLayout2" style="flex-basis:50%">
    <davinci-drop-down id="DropDown1" label="Fixed Data">
      <davinci-option id="Option1" value="This is the first fixed choice">
        <span>Choice 1</span>
      </davinci-option><davinci-option id="Option2" value="This is the second fixed choice"
        <span>Choice 2</span>
      </davinci-option><davinci-option id="Option3" value="This is the third fixed choice">
        <span>Choice 3</span>
      </davinci-option>
      <crown-binding event="change" name="DropDown/setFixedSelection" path="param/args/item
    </davinci-drop-down>
  </layout-column>
```

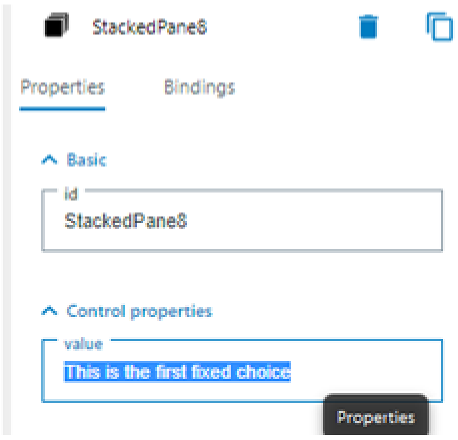
5. … and add the following to dropdown code:
<local-binding property="value" name="selectedItem"></local-binding>



```
      </davinci-option>
      <crown-binding event="change" name="DropDown/setFixedSelection" path="param/args/item
      <local-binding event="change" name="selectedItem"></local-binding>
    </davinci-drop-down>
```

6. The different values of the **DropDown** control can then be defined on the **StackedPanes** in the **StackedView**. Selecting the value in the **DropDown** will then show the corresponding **StackedView**





id: StackedPane8

value: This is the first fixed choice