Creating a Textbox PCF Control

A very basic example of adding an HTML DOM input textbox can be found [here](https://www.w3schools.com/jsref/dom_obj_text.asp). The code we will add to the init function in our index.ts file:

var x = document.createElement("INPUT");

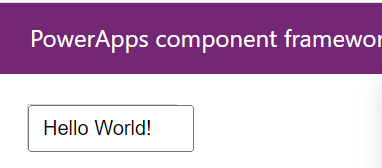
x.setAttribute("type", "text");

x.setAttribute("value", "Hello World!");

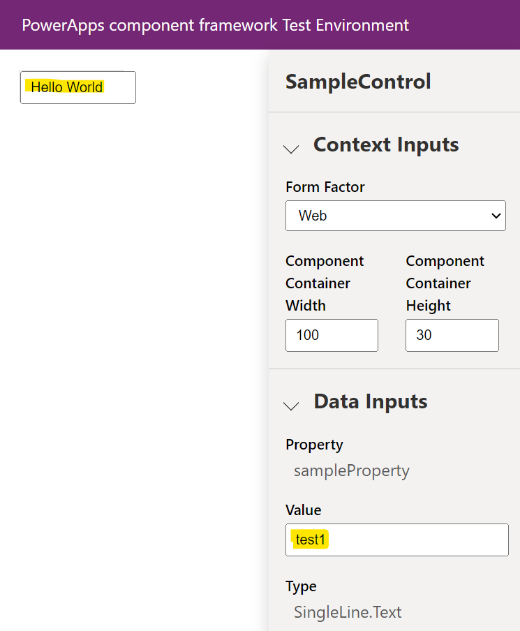
container.appendChild(x);



On saving, our test harness is now updated with a textbox control:

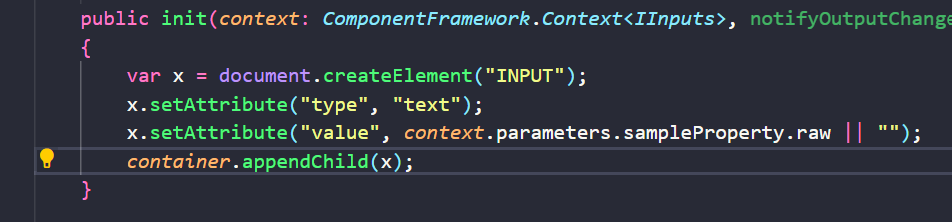


However there is a disconnect between what is in the textbox and what is the value. Changing either doesn’t change the other:



Firstly, let’s add some code to set the value on init. From our SampleProperty bound property that is currently in our manifest, we can access the value through:

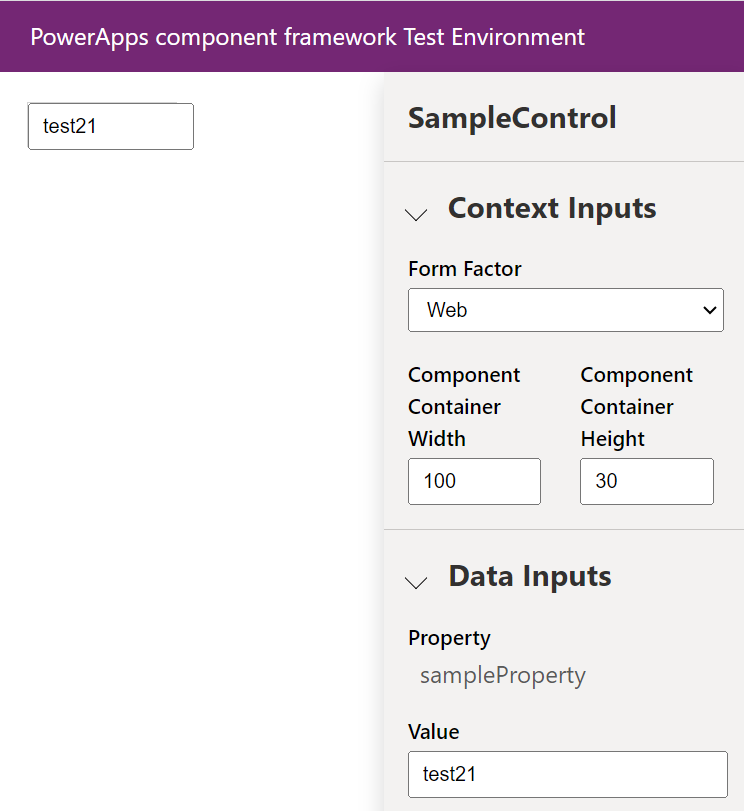
context.parameters.sampleProperty.raw



Let’s set the value of our textbox to this property:

***x.setAttribute("value", context.parameters.sampleProperty.raw || "");***

On refreshing the test harness, we see the value that I enter in the sampleProperty on the right is reflected in our textbox on the left:

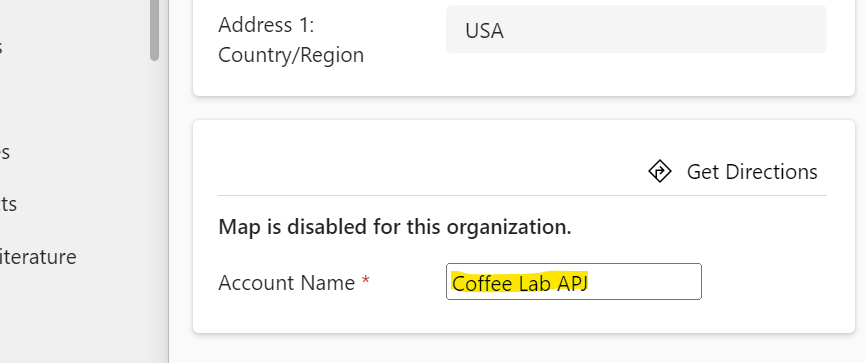


However, if I change either textbox on the right or left, the other textbox does not change. More on that shortly.

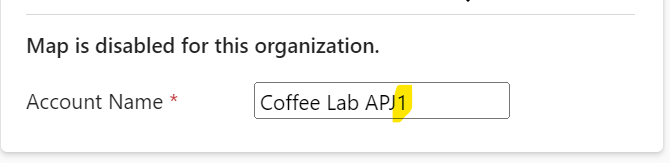
Let’s push this control up to the Power Apps environment and see what it looks like. We’re going to use the same command as before from the SampleControl folder.

**pac pcf push --publisher-prefix carl**

Now if we look in our org, we can see the field is a textbox, and the value is actually defaulting to the account name, i.e. the field our SampleProperty is bound to:

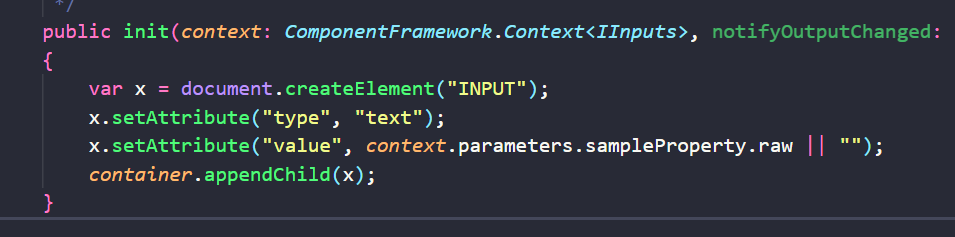


Now if we try to change the value of the field and save the record, nothing happens and the record doesn’t save. In fact, the page doesn’t even register that the form is dirty.



To resolve this, let’s go back to VS Code and take a look at some of the other methods. The one we’re interested in is updateView, which is “called when any value in the property bag has changed. This includes field values, data-sets, global values”. So we want to update our HTML element here, but we will need to change our init code a little to make our properties accessible from these other methods we’re calling.

So instead of our code looking like this:



We will create a new private member:

***private \_element: HTMLInputElement;***

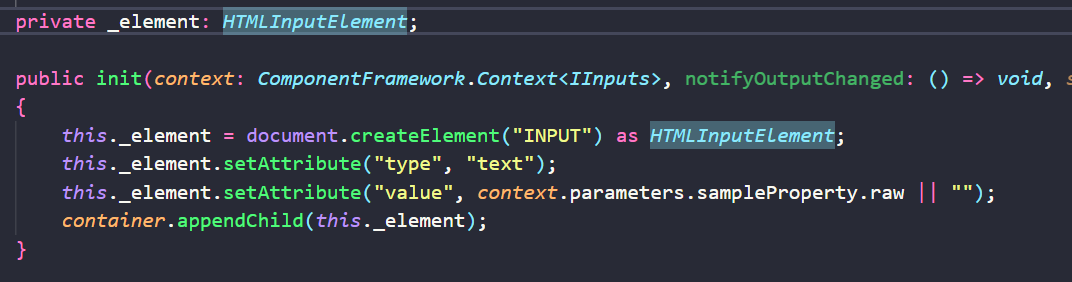
***And new code for init to use this member:***

***this.\_element = document.createElement("INPUT") as HTMLInputElement;***

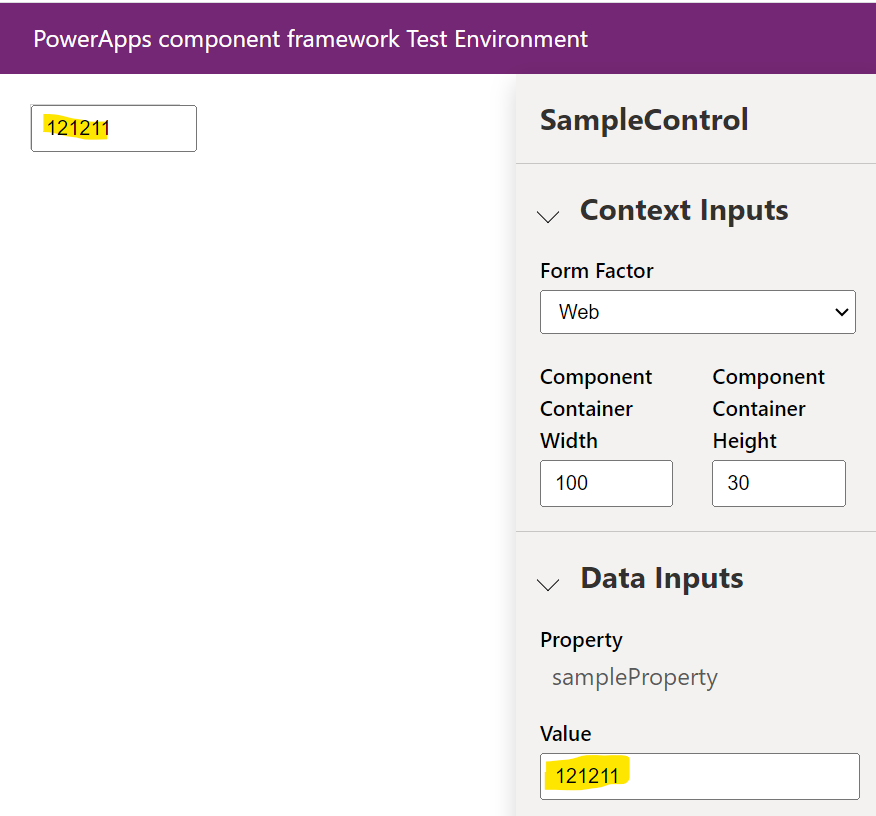
***this.\_element.setAttribute("type", "text");***

***this.\_element.setAttribute("value", context.parameters.sampleProperty.raw || "");***

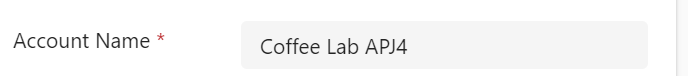
***container.appendChild(this.\_element);***



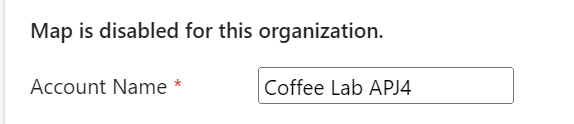
Let’s save and test this. Now as I type on the right, the control on the left is automatically updating, and without having to press enter:



Let’s push this up to our org and observe what’s happening. We have 2 fields on the Account form that are the Account Name, one is the original field with its out of the box control, and the other is the same field bound to our custom PCF control. If we change the out of the box field and tab off it:



We see our custom PCF is updated:



This is due to the updateView functionality we added.

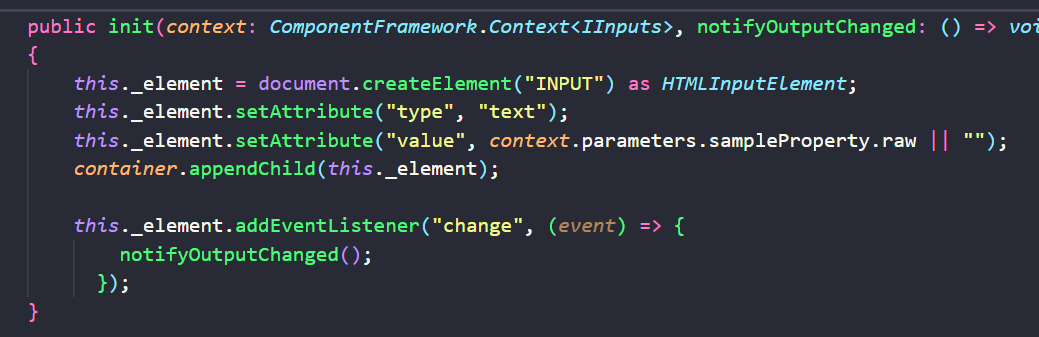
However, if we update our PCF control’s value we still don’t get a form dirty updated value of the field. Let’s now add code to do that using notifyOutputChanged(). This is a method to tell the system that a change has been made to our control, and it’s up to us to invoke it. In this example, let’s add a change event to the textbox control, and when it fires we will call our notifyOutputChanged().

We will add this to our init:

this.\_element.addEventListener("change", (event) => {

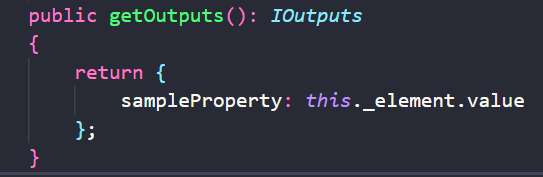
notifyOutputChanged();

});

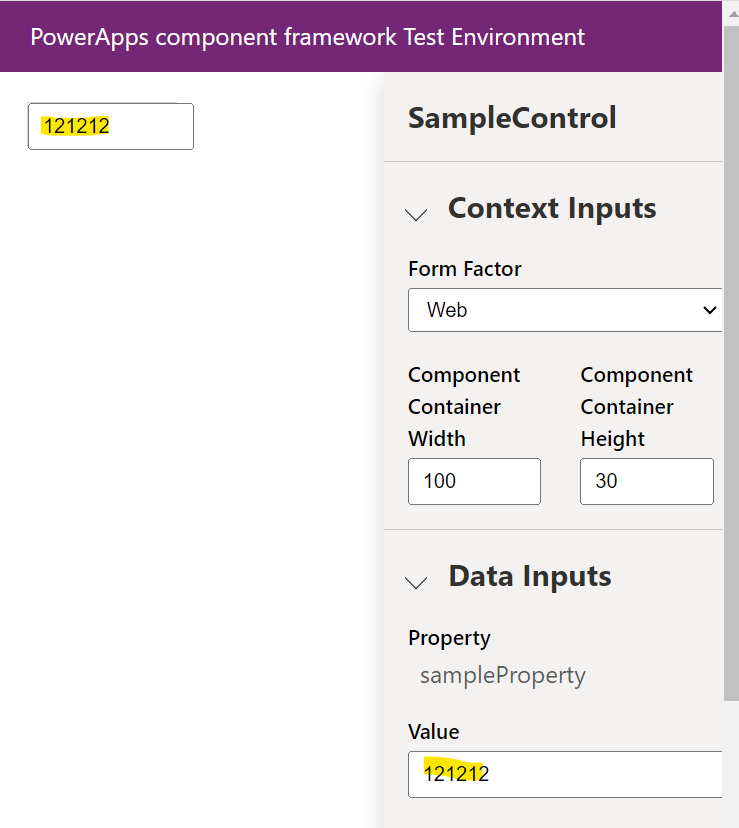


Now when the text in our control changes and the user tabs off the control, our notifyOutputChanged() runs, and this in turn natively invokes the getOutputs() function. In this function we want to set our SampleProperty to be the value of our element:

**sampleProperty: this.\_element.value**

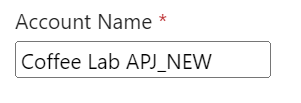


Now when we make a change to the value of our control on the left and tab off the field, our value gets updated on the right:

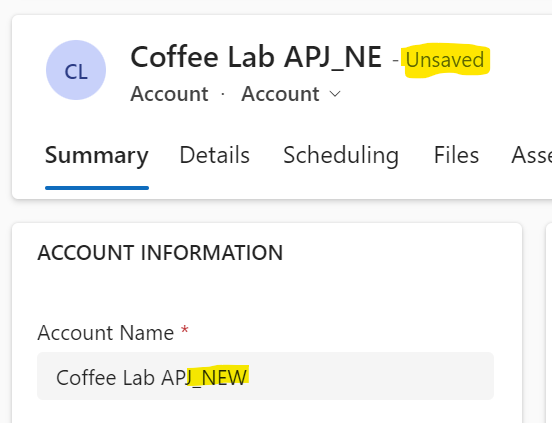


Let’s publish this latest control to our org and see how it looks.

When we change the value in our PCF control, we see now the out of the box Account Name field also changes its value, and the form becomes dirty. Our control:



Form is now dirty with our change, out of the box field updated:



This is a simple example of how these pieces fit together.

Finally, instead of updating the value when the user tabs off the control, we can do it when any text changes on the control. For this we would use “input” instead of change:

this.\_element.addEventListener("input", (event) => {

notifyOutputChanged();

});

That’s it! You can now build a text PCF control.