add a new field selector 'dss-extension.json' - add a new field selector. will need to change how we combine priority/severity value.

Add a new input then use combined values from the two input values to the combine color output.

second field

intead of somevalue; somevalue+somevalue; have N number of items that map to the three colors, red, yellow, green.

third field would be the color selection

start with dss-extestion.json, there's a property section with an array of inputs in json. How to specify inputs to see the options.

have the content be something json.

1 specified field

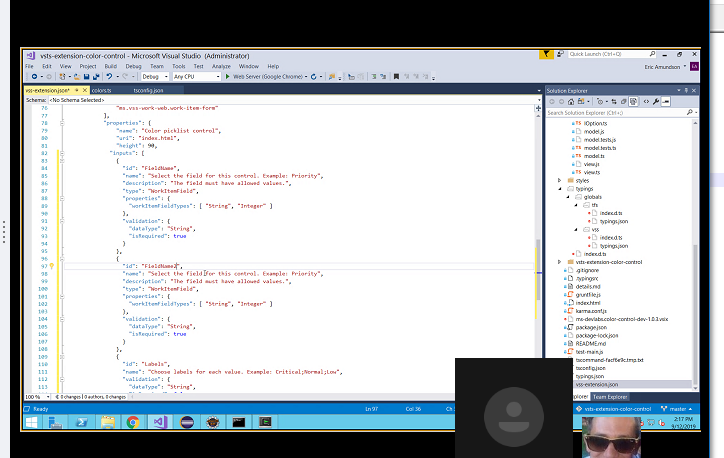
2- second specified field

3- will do the color combination based on first two

4- will be the actual color value.

Screenshot to add the new fields to the color plugin “dss-extenstions’

Include second field and a third field that will calculate a color score.



Create a new meta-data.json that will ‘replace’ the “color” section of the dss-extension file.

Will still need to add the new fields to ‘dss-extension’ as seen above, just remove the color section and replace with the meta-data.json file that will do the color mapping based on what’s in field 1 and field 2

Meta-data.json will map all the permutations. Eric will upload meta-data.json and the ‘modified’ dss-extension file into the repo. Start with making these edits

1. Add second field to vss-extension
2. Create new meta-data.json with basic color mapping code
3. Remove “color” section from vss-extion, replace with the meta-data.json mapping file
4. Update project code to account for new meta-data.json

removed the colors code block from the vss-extension json and added meta-data json. It looks like colors is referenced in "InputParser.ts"

**Changes needed to “InputParser.ts”**

import { Colors} from "./colors"

should probably replace with ./meta-data.json or meta-data.json

import { Mapping } from "./meta-data.json"

**what changes needed to the InputParser code to pickup meta-data.json?**

InputParser Parses and gets a FieldName from a dictionary. A field name such as “Priority”

It’s getting the FieldName from vss-extension.json?

|  |
| --- |
| *public static getFieldName(inputs: IDictionaryStringTo<string>): string {* |
|  |

|  |
| --- |
| *if (inputs["FieldName"]) {* |
|  |

|  |
| --- |
| *return inputs["FieldName"];* |
|  |

|  |
| --- |
| *}* |
|  |

|  |
| --- |
| *throw ("FieldName not specified.")* |
|  |

*}*

Do I need to add a block for the second field name? for the Severity field?

|  |
| --- |
| *public static getFieldName2(inputs: IDictionaryStringTo<string>): string {* |
|  |

|  |
| --- |
| *if (inputs["FieldName2"]) {* |
|  |

|  |
| --- |
| *return inputs["FieldName2"];* |
|  |

|  |
| --- |
| *}* |
|  |

|  |
| --- |
| *throw ("FieldName2 not specified.")* |
|  |

*}*

Next there’s public static getOptions – it parses inputs from IDictionary and returns an array of Interfaces of the structure – value, color, and label. Label value is ‘1 for priority’ or ‘1 – Critical’ for Severity. Color will be pulled in from meta-data.json based on values entered into field 1 and field 2

Code highlighted in yellow to be added

Code highlighted in blue remove

|  |
| --- |
| public static getOptions(inputs: IDictionaryStringTo<string>, allowedValues: string[]): IOption[] { |
|  |

|  |
| --- |
| if (allowedValues && allowedValues.length) { |
|  |

|  |  |
| --- | --- |
| let colors: string[] = []; |  |
|  |  |

|  |
| --- |
| let inputColors: string[] = []; |
|  |

|  |
| --- |
| let labels: string[] = []; |
| let label1: string[] = []; |
| let label2: string[] = []; |
|  |
|  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| let inputLabels: string[] = [];  let inputLabel1: string[] = [];  let inputLabel2: string[] = [];   |  | | --- | | inputColors = InputParser.\_extractInputs(inputs["Colors"]); | | \*get color from “color” in meta-data.json  inputColors = InputParser.\_extractInputs(inputs["Mapping.Colors"]); |  |  | | --- | | inputLabels = InputParser.\_extractInputs(inputs["Labels"]); | | \*get priority value from meta-data.json  inputLabel1 = InputParser.\_extractInputs(inputs["Mapping.f1"]); | | \* get Severity vaue from meta-data.json  inputLabel2 = InputParser.\_extractInputs(inputs["Mapping.f2"]);  **"color": "green",**  **"mapping": [**  **{**  **"f1": "3",**  **"f2": "4 - Low "**  **},** | |  | |  |  |  | | --- | |  | |  |  |  | | --- | | colors = InputParser.\_getColors(inputColors, allowedValues); | |  |   labels = InputParser.\_getLabels(inputLabels, allowedValues);  label1 = InputParser.\_getLabels(inputLabel1, allowedValues);  label2 = InputParser.\_getLabels(inputLabel2, allowedValues); |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return InputParser.\_buildOptions(allowedValues, colors, labels); |
| return InputParser.\_buildOptions(allowedValues, colors, label1, lable2); |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| throw ("The backing field does not have allowed values."); |
|  |

|  |
| --- |
| } |
|  |

}

InputParser is also calling IOption.ts which references

export interface IOption {

value: string,

color: string,

label: string

}

control.ts also uses "InputParser"

**Meeting 2 Notes:**

Update control.ts to recognize second field name – to get the inputs

Add:

//getting info From FieldName

private \_fieldName: string = "";

//getting info from FieldName2

private \_fieldName2: string = "";

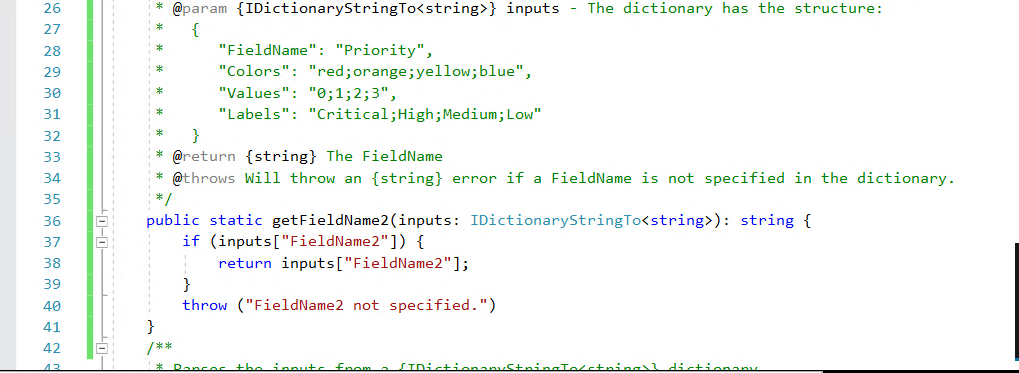
this.\_inputs = VSS.getConfiguration().witInputs;

this.\_fieldName = InputParser.getFieldName(this.\_inputs);

this.\_fieldName2 = InputParser.getFieldName2(this.\_inputs);

InputParser changes:

Add fieldName2 stuff to the InputParser.getFieldName2



Add a method for getOptions. InputParser doesn’t need the GetOptions as is – we will rewrite it in InputParser (allowed values not needed) and comment out the //If (allowed values block)

This section is parses the data from IDictionary and returns array from the interface “IOption” which originally had variables for value, color, and label. Now IOption has “color”, “mapping” and “f1” and “f2” from the IMapping section.

|  |
| --- |
| \* @return an array of Interfaces of the structure: { |
|  |

|  |
| --- |
| \* value: values[i], |
|  |

|  |
| --- |
| \* color: colors[i], |
|  |

\* label: labels[i]

|  |
| --- |
| \* @return an array of Interfaces of the structure: { |
|  |

|  |
| --- |
|  |

|  |
| --- |
| \* color: colors[i], |
|  |

\* f1: field1[i],

\* f2: field2[i],

|  |
| --- |
| public static getOptions(inputs: IDictionaryStringTo<string>, allowedValues: string[]): IOption[] { |
| public static getOptions(inputs: IDictionaryStringTo<string>, IOption[] { |
|  |

|  |
| --- |
| /\*if (allowedValues && allowedValues.length) { |
|  |

|  |
| --- |
| let colors: string[] = []; |
|  |

|  |
| --- |
| let inputColors: string[] = []; |
|  |

|  |
| --- |
| let labels: string[] = []; |
| Let field1: string[]:  Let field2: string[]; |
|  |

|  |
| --- |
| let inputLabels: string[] = []; |
| \*/ |

|  |
| --- |
|  |
|  |

|  |
| --- |
| inputColors = InputParser.\_extractInputs(inputs["Colors"]); |
|  |

|  |
| --- |
| inputLabels = InputParser.\_extractInputs(inputs["Labels"]); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| colors = InputParser.\_getColors(inputColors, allowedValues); |
|  |

|  |
| --- |
| labels = InputParser.\_getLabels(inputLabels, allowedValues); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return InputParser.\_buildOptions(allowedValues, colors, labels); |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| throw ("The backing field does not have allowed values."); |
|  |

|  |
| --- |
| } |
|  |

}

Change to

|  |
| --- |
| \* @return an array of Interfaces of the structure: { |
|  |

|  |
| --- |
|  |

|  |
| --- |
| \* color: colors[i], |
|  |

\* f1: field1[i],

\* f2: field2[i],

|  |
| --- |
| public static getOptions(inputs: IDictionaryStringTo<string>, allowedValues: string[]): IOption[] { |
| public static getOptions(inputs: IDictionaryStringTo<string>, IOption[] { |
|  |

|  |
| --- |
|  |

|  |
| --- |
| let colors: string[] = []; |
|  |

|  |
| --- |
| let inputColors: string[] = []; |
|  |

|  |
| --- |
| Let field1: string[]:  Let field2: string[]; |
|  |

|  |
| --- |
| let inputFields1: string[] = []; |
| let inputFields2: string[] = []: |
| \*/ |

|  |
| --- |
|  |
|  |

|  |
| --- |
| inputColors = InputParser.\_extractInputs(inputs["Colors"]); |
|  |

|  |
| --- |
| inputFields1 = InputParser.\_extractInputs(inputs["Mapping?"]); |
| inputFields2 = InputParser.\_extractInputs(inputs["IMapping?"]); |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| colors = InputParser.\_getColors(inputColors, allowedValues); |
|  |

|  |
| --- |
| labels = InputParser.\_getLabels(inputLabels, allowedValues); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return InputParser.\_buildOptions(allowedValues, colors, labels); |
|  |

|  |
| --- |
| } else { |
|  |

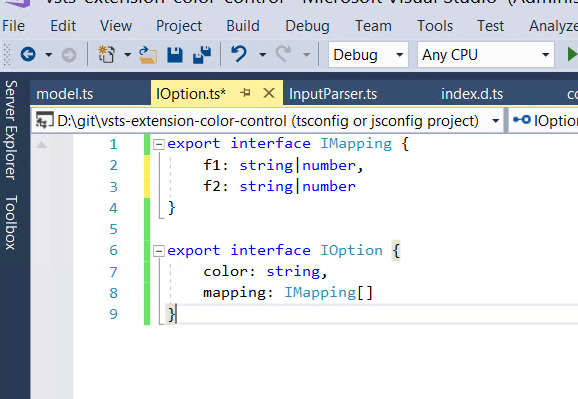
|  |
| --- |
| throw ("The backing field does not have allowed values."); |
|  |

|  |
| --- |
| } |
|  |

}

IOptions needs to be fixed to pull in the full input structure.

Changes needed to the IOption code so it pull in the F1 and F2 values



Next we have to get to the color that get’s output.