Batch Level Field Change Script

Earlier, there was no mechanism to increase/decrease the number of value options available for a batch class field drop-down list if the value of another batch class field drop-down list was changed.

With Ephesoft v4.1.0.0, you can now customize the script to control the list of value options available for a batch class field drop-down list based on value options that have already been selected for other batch class field drop-down lists.

For example, if you select a value from a list of value options available in the drop-down list for a batch class field, Ephesoft will be able to control other drop-down list of options based on the selected value to this script. The script may increase/decrease the options based on the selection.

This script can be invoked either from the **Upload Batch** screen or from the **Web Scanner** screen.

Custom Script to Control List of Batch Level Field Options

A new method has been added in **utility.js** where you can write custom script to add batch class field values to existing field-value list of options.

This method returns a JSON string representation of batch level field values in format as below:

```
[
   {
      "key":"field1",
      "options":"value1, value2, value3"
   },
   {
      "key":"field2",
      "options":"value1, value2, value3"
   },
   {
      "key":"field3",
      "options":"value1, value2, value3"
   }
]
```


The **key** used in custom script should match **Description** of batch class fields defined in grid view.

Value options are automatically updated in drop-down lists of **Web Scanner** or **Upload Batch** screens.

Custom Script to Control Batch Level Field Options, When Value is Changed

A new method has been created in **utility.js** (to avoid modifying signature of existing method) with following signature:

Existing Method

```
/*
 * Method to pre populate batch class fields on basis of batch class identifier,
 * batch class field, value typed and description in web scanner and upload
 * batch.
 */
function getBatchClassFieldsWhileFieldValueChange(batchClassIdentifier,
                batchClassDescription, key, value) {
     // Custom code
}
```

Expected JSON Response:

```
[
   {
      "key":"field1",
      "value":"value1"
   },
   {
      "key":"field2",
      "value":"value2"
   },
   {
      "key":"field3",
      "value":"value3"
   }
]
```

New Method

```
/*
 * Method to pre populate batch class fields on basis of existing jsonFieldValues, batch class identifier,
 * batch class field, value typed and description in web scanner and upload
 * batch.
 */
function getBatchClassFieldOptionsOnValueChange(jsonFieldValues, batchClassIdentifier,
                batchClassDescription, key, value) {
     // Custom code
}
```

Expected JSON Response:

```
 [
   {
```

```
        "key":"field1",
        "value":"value1",
        "options":"value1, value2, value3"
    },
    {
        "key":"field2",
        "value":"value2",
        "options":"value1, value2"
    },
    {
        "key":"field3",
        "value":"value3",
        "options":"value1, value2"
    }
]
```

The new method, by default, invokes the existing method, to create JSON response. This is done to ensure existing custom implementation can be used on upgrade releases.

The **key** used in custom script should match **Description** of batch class fields defined in grid view.

Value from the JSON response is set as the value of the batch level field.

Options from the JSON response are set as list of values in combo box.

If **Options** is undefined in JSON response, then list of values remains unchanged.

Since, on Ephesoft upgrade, **utility.js** gets replaced, any custom implementation batch level fields in existing methods has to be manually migrated.

Detailed Explanation of Default Implementation of custom script code

```
/*
 * Method to override batch class field options, on the basis of json list of field options,
 * batch class identifier, batch class field, value typed and description in web scanner
 * and upload batch.
 */
function getBatchClassFieldOptionsOnValueChange(jsonFieldValues, batchClassIdentifier,
              batchClassDescription, key, value) {

       // Using existing method's implementation, to fetch the JSON response. If user do not
need to use existing method, then user can initialize bcf as empty array of batch class fields
       // var bcf = '[]';
```

```
	var bcf = getBatchClassFieldsWhileFieldValueChange(batchClassIdentifier,
		batchClassDescription, key, value);

	// 'key' is the description in batch level field, for which value change is triggered
	// 'value' is the changed value, for which script is invoked
	// ‘jsonFieldValues’ contains the JSON envelope of batch class fields
	if (key != undefined && jsonFieldValues != undefined) {

		// Converting bcf to JSON object representation
		var jsonBCFArr = JSON.parse(bcf);
		var jsonFieldValuesArr = JSON.parse(jsonFieldValues);

		// Copying list of 'options' from jsonFieldValues to bcf
		// If key is not present in BCF, then adding the key from jsonFieldValues to
jsonBCFArr array representation
		for(var k = 0; k < jsonFieldValuesArr.length; k++) {
			var isKeyPresent = false;
			var jsonFieldValuesObj = jsonFieldValuesArr[k];
			for(var m = 0; m < jsonBCFArr.length; m++) {
				var jsonBCFObj = jsonBCFArr[m];
				if(jsonFieldValuesObj.key == jsonBCFObj.key) {
					jsonBCFObj["options"] = jsonFieldValuesObj.options;
					isKeyPresent = true;
					break;
				}
			}
			if(!isKeyPresent) {
				jsonBCFArr.push(jsonFieldValuesObj);
			}
		}

		// Default implementation (Customize, if required): Removing the selected value
from other dropdowns.
		for (var k = 0; k < jsonBCFArr.length; k++) {
			var jsonObj = jsonBCFArr[k];
			// Note: jsonObj is a JSON JavaScript Object and not String value. Hence
this can be simply being iterated as <jsonObj.key> for fetching the key or <jsonObj.options>
for fetching the options from JSON object
			if(jsonObj.key != key && (jsonObjOptions = jsonObj.options) != undefined
&& (index = jsonObjOptions.indexOf(value)) != -1) {
				jsonObjOptions.splice(index, 1);
			}
		}

		// Converting JSON array object to String response
		bcf = JSON.stringify(jsonBCFArr);
	}

	return bcf;
}
```

Sample Custom Implementations

#1: Sample implementation showing how to decrease the dropdown options:

When user selected ‘Messi’ from dropdown for a Batch Class Field named as ‘Name’, then all other Batch Class Fields should be made blank and remove all options from dropdown for other Batch class fields.

```
if (key == "Name" && value=="Messi") {
	for(var k = 0; k < jsonBCFArr.length; k++) {
		var jsonObj = jsonBCFArr[k];
		if(jsonObj.key != key && (jsonObjOptions = jsonObj.options) != undefined) {
			jsonObj.options = [];
			jsonObj.value="";

		}
	}
}
```

#2: Sample implementation showing how to increase the dropdown options:

When user selected ‘Messi’ from dropdown for a Batch Class Field named as ‘Name’, then all other Batch Class Fields should have only ‘Football’ as the dropdown option and value should be set as blank.

```
if (key == "Name" && value=="Messi") {
	for(var k = 0; k < jsonBCFArr.length; k++) {
		var jsonObj = jsonBCFArr[k];
		if(jsonObj.key != key && (jsonObjOptions = jsonObj.options) != undefined) {
			jsonObj.options = [];
			jsonObj.options.push("Football");
			jsonObj.value="";
		}
	}
}
```