**ASSIGNMENT OF SUITESCRIPT 2.0 CLIENT SCRIPT**

***Create a SuiteScript 2.0 client script that will perform the following tasks.***

1. **Upon loading the sales order, the script should set a customer in customer field.**

Here I used the **N/currentRecord** module as I have to deal with the current record open in the UI. E.g. - A user is interacting with the customer's name field in a sales order. It also helps to interact efficiently with the record that is already loaded in the user’s browser.

Again, I used **pageInit()** entry points as it should fetch the employee's name once the page gets initialized.

***Code:***

/\*\*

\* @NApiVersion 2.0

\* @NScriptType ClientScript

\*/

define(['N/currentRecord'], function (currentRecord) {

/\*\*

\* Function to be executed when the page is initialized.

\*

\* @param {Object} scriptContext

\* @param {Record} scriptContext.currentRecord - Current form record

\* @param {string} scriptContext.mode - The mode in which the record is being accessed (create, copy, or edit)

\*/

//Q1

function pageInit(scriptContext) {

var current\_Record = scriptContext.currentRecord; // I fetch the current record upon which I have to change any field

var recordContext = current\_Record.getValue({ // Get the fieldId of that field.

fieldId: 'entity'

});

// log.debug('entity', recordContext);

current\_Record.setValue({ // Set the field value as 1126 in the fieldId of entity

fieldId: 'entity',

value: 1126 // customer's Internal Id not id

});

// alert('Page Initialized');

}

return { // Now just simply returned the object that maps the function name

pageInit: pageInit, // entry point: functionName

}

});

***Step 1*** - Now upload the Script under Customization -> Scripting -> Scripts -> New.

***Step 2*** - Now select the particular record from the browser and click on SAVE.

***Step 3*** – After this click on deploy script and choose record as Sales Order. Then click Save.

***Step 4*** – Now navigate to Transaction -> Sales -> Enter Sales Order, you will get the desired output.

***Output:***

1. **When the customer is changed, the script should retrieve the new customer's name and display it in memo field.**

Here I used **fieldChanged()** entry point as I need to change a field value after the page gets initialized.

***Code:***

function fieldChanged(scriptContext) {

var current\_record = scriptContext.currentRecord;

if (scriptContext.fieldId === 'entity' && !isItemAdded) {

var customerName = current\_record.getText({

fieldId: 'entity'

});

current\_record.setValue({

fieldId: 'memo',

value: customerName

});

}

}

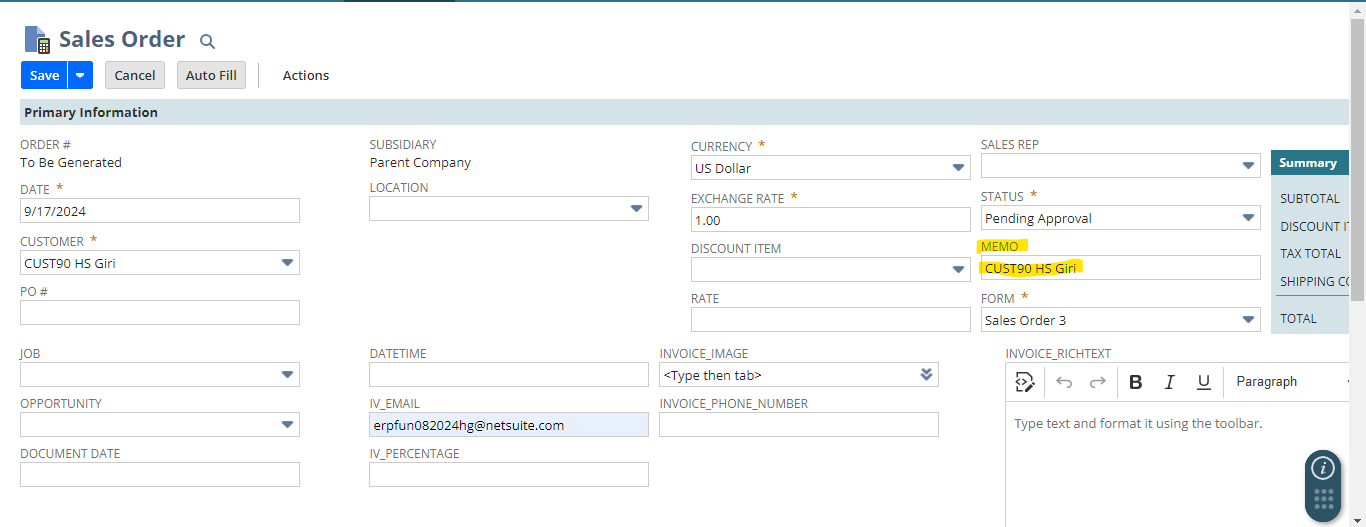
return { // Now just simply returned the object that maps the function name

pageInit: pageInit, // entry point: functionName

fieldChanged: fieldChanged

}

***Output:***



1. **While adding a new line one new item should be added to the 2nd line and with quantity 2.**

Here I used **sublistChagned**() entry point as we are dealing with a sublist item and is executed after a sublist has been inserted, removed, or edited.

***Code:***

function sublistChanged(scriptContext) {

alert('Entry point triggered');

var current\_record = scriptContext.currentRecord;

if (scriptContext.sublistId == 'item' && scriptContext.fieldId === 'item') {

var noOfLine = current\_record.getLineCount({

sublistId: 'item'

});

if (noOfLine == 1) {

current\_record.selectNewLine({

sublistId: 'item'

});

current\_record.setCurrentSublistValue({ //setting the item

sublistId: 'item',

fieldId: 'item',

value: 345

});

current\_record.setCurrentSublistValue({ //setting the quantity of that item

sublistId: 'item',

fieldId: 'quantity',

value: '2'

});

// current\_record.commitLine({ // It will Commit the currently selected line.

// sublistId: 'item'

// });

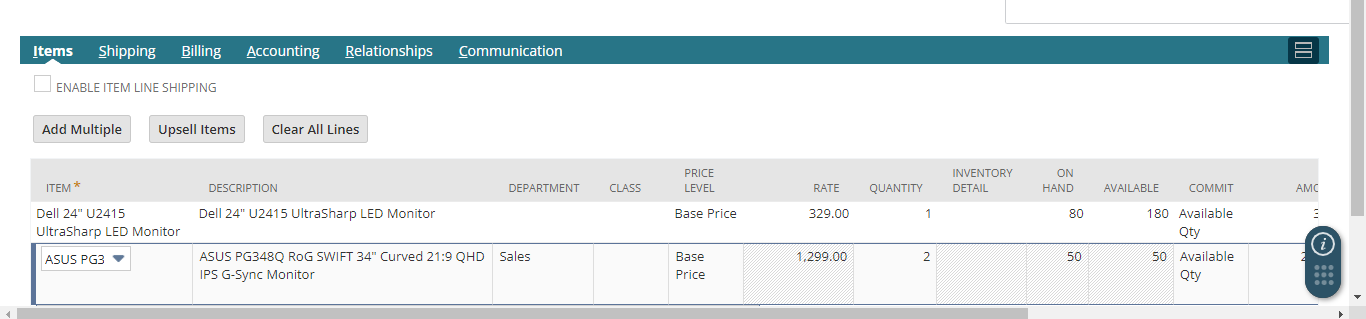
}

}

return { // Now just simply returned the object that maps the function name

sublistChanged: sublistChanged // entry point: functionName

}

***Output:*** 

1. **Do let the line be added if the quantity for the item is less than 3. (If quantity exceed 2 then you can’t add that item)**

Here I have used **validateLine**() because it validate before adding a line in a sublist.

***Code:***

//Q4 - if quantity > 2, show an alert

function validateLine(scriptContext) {

var current\_record = scriptContext.currentRecord;

var sublist = scriptContext.sublistId;

if(sublist === 'item'){

var quantity = current\_record.getCurrentSublistValue({

sublistId: 'item',

fieldId: 'quantity',

});

if(quantity > 2) { // No. Of quantity should less than or equal to 2.

dialog.alert({

title: 'Quantity Limit Exceeded',

message: 'The quantity of the item can not exceed 2'

});

return false;

}

return true;

}

}

return { // Now just simply returned the object that maps the function name

validateLine: validateLine

}

***Output:***

1. **Do not line be deleted if the quantity for the item is 4. (If the quantity is exactly 4, then item can’t be deleted.)**

I have used **validateDelete**() here because it is executed when an existing line in an edit sublist is deleted.

***Code:***

//Q5 - if quantity == 4, you can't delete it.

function validateDelete(scriptContext){

var current\_record = scriptContext.currentRecord; // record selected

var sublist = scriptContext.sublistId; // sublist selected

if(sublist == 'item') { // entry to the item sublist

var quantity = current\_record.getCurrentSublistValue({

sublistId: 'item',

fieldId: 'quantity'

});

if(quantity == 4){

dialog.alert({

title: 'Please Check Quantity Limit',

message: 'Quantity should be exactly 4. It should be greater than or less than 4.'

});

return false;

}

return true;

}

}

return { // Now just simply returned the object that maps the function name

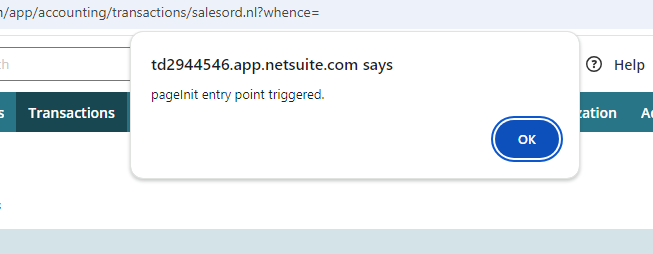
validateDelete: validateDelete

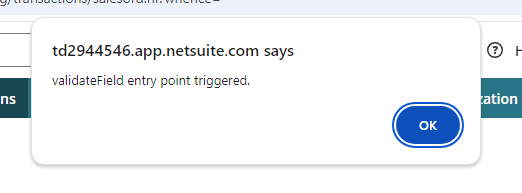
}

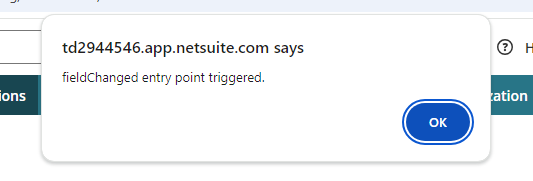
***Output:***

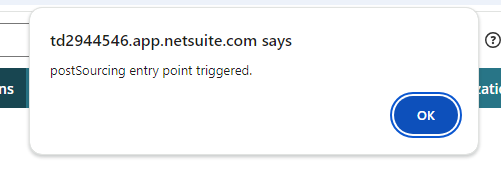
1. **Write all trigger points and show trigger name in alert function.**

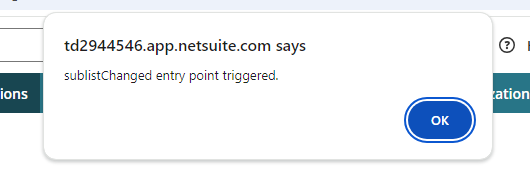
***Screenshots of all entry points when it gets triggered in alert:***

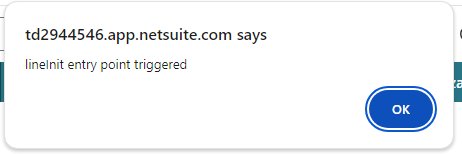


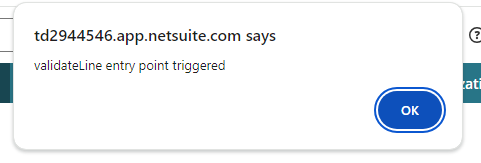


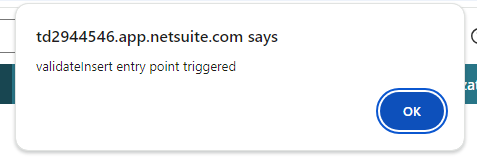


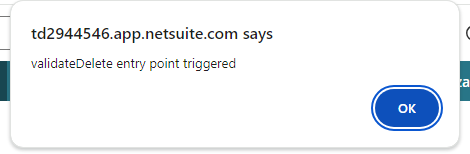


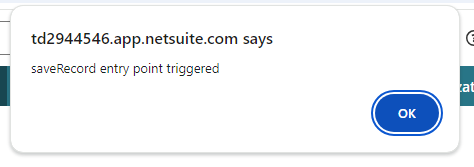












***Code of all entry points all together:***

/\*\*

\* @NApiVersion 2.x

\* @NScriptType ClientScript

\*/

define([], function() {

// Triggered when the form is loaded

function pageInit(scriptContext) {

alert('pageInit entry point triggered');

}

// Triggered when a field is changed

function fieldChanged(scriptContext) {

alert('fieldChanged entry point triggered');

}

// Triggered when a field's source is changed (post sourcing)

function postSourcing(scriptContext) {

alert('postSourcing entry point triggered');

}

// Triggered when the sublist is changed

function sublistChanged(scriptContext) {

alert('sublistChanged entry point triggered');

}

// Triggered when a line in a sublist is initialized

function lineInit(scriptContext) {

alert('lineInit entry point triggered');

}

// Triggered when a field's value is being validated

function validateField(scriptContext) {

alert('validateField entry point triggered');

return true; // Return true to allow the field change to proceed

}

// Triggered when a line in the sublist is being validated

function validateLine(scriptContext) {

alert('validateLine entry point triggered');

return true; // Return true to allow the line to be saved

}

// Triggered when an insert action is being validated

function validateInsert(scriptContext) {

alert('validateInsert entry point triggered');

return true; // Return true to allow the insert to proceed

}

// Triggered when a delete action is being validated

function validateDelete(scriptContext) {

alert('validateDelete entry point triggered');

return true; // Return true to allow the delete to proceed

}

// Triggered when the record is being saved

function saveRecord(scriptContext) {

alert('saveRecord entry point triggered');

return true; // Return true to allow the record to be saved

}

// Returning each function separately as per SuiteScript ClientScript structure

return {

pageInit: pageInit,

fieldChanged: fieldChanged,

postSourcing: postSourcing,

sublistChanged: sublistChanged,

lineInit: lineInit,

validateField: validateField,

validateLine: validateLine,

validateInsert: validateInsert,

validateDelete: validateDelete,

saveRecord: saveRecord

};

});