Week 3

Working with Field Values in SuiteScript and Navigating Documentation

Overview

You will learn:

1. About the NetSuite SuiteScript documentation and how to use it
2. Various NetSuite script types
3. Getting and setting field values using built-in SuiteScript methods! (record.getValue(), record.setValue())
4. How to build, deploy, and test your very own client script!

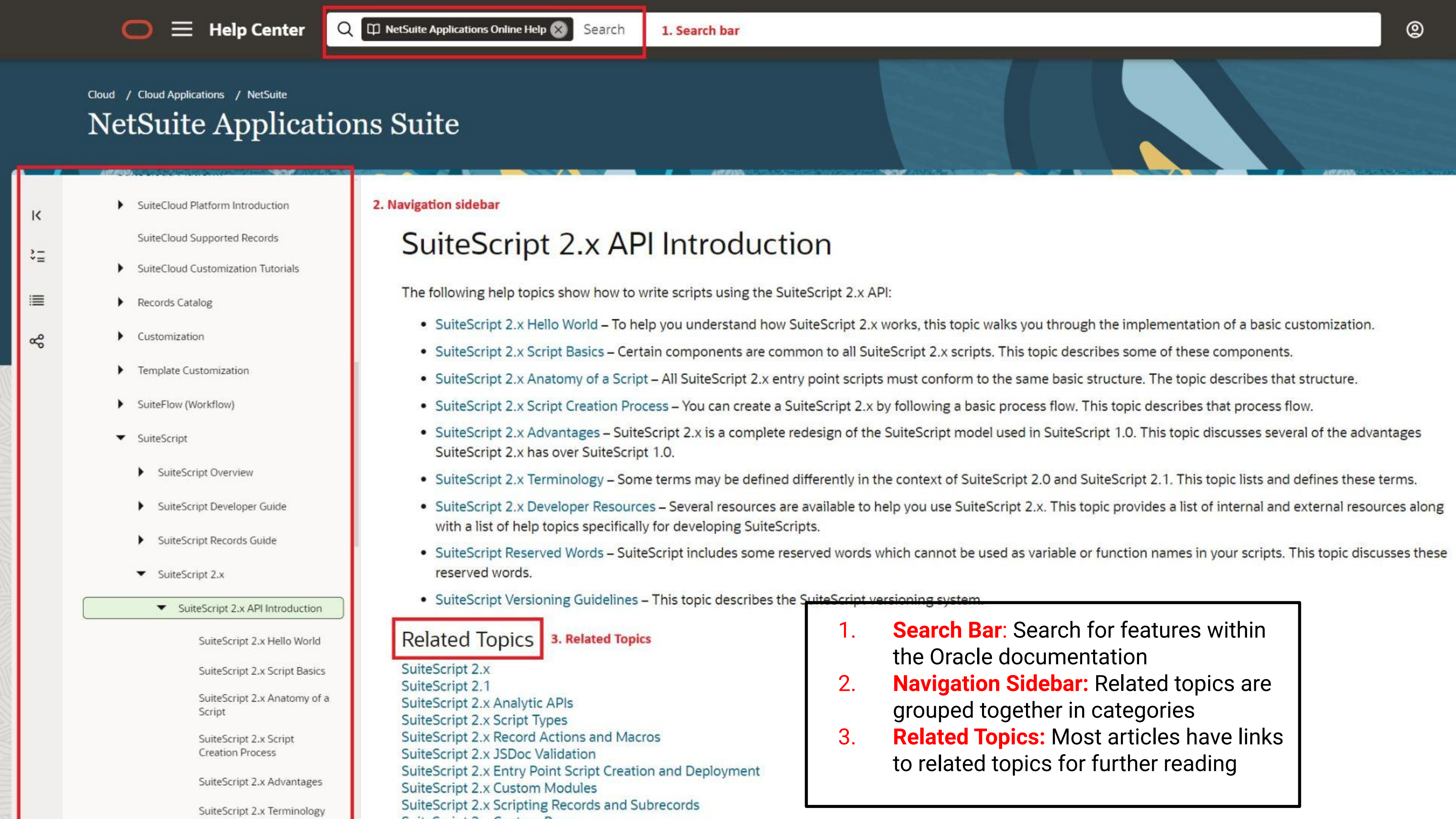
Effectively Navigating NetSuite SuiteScript Help Documentation

Overview of SuiteScript Help Documentation

- Link to the official Oracle SuiteScript documentation.

  Go-to resource for:

  - Script Methods: (getValue, setValue, etc…)
  - Objects: Definitions of key NetSuite objects and how to work with them (records, search, sublists, etc…).
  - Examples: Code Snippets and examples to demonstrate real-world usage.



Help Center

NetSuite Applications Online Help

Search

1. Search bar

Cloud / Cloud Applications / NetSuite

NetSuite Applications Suite

- SuiteCloud Platform Introduction
- SuiteCloud Supported Records
- SuiteCloud Customization Tutorials
- Records Catalog
- Customization
- Template Customization
- SuiteFlow (Workflow)
- SuiteScript
  - SuiteScript Overview
  - SuiteScript Developer Guide
  - SuiteScript Records Guide
  - SuiteScript 2.x
    - SuiteScript 2.x API Introduction
      - SuiteScript 2.x Hello World
      - SuiteScript 2.x Script Basics
      - SuiteScript 2.x Anatomy of a Script
      - SuiteScript 2.x Script Creation Process
      - SuiteScript 2.x Advantages
      - SuiteScript 2.x Terminology

2. Navigation sidebar

SuiteScript 2.x API Introduction

The following help topics show how to write scripts using the SuiteScript 2.x API:

- SuiteScript 2.x Hello World – To help you understand how SuiteScript 2.x works, this topic walks you through the implementation of a basic customization.
- SuiteScript 2.x Script Basics – Certain components are common to all SuiteScript 2.x scripts. This topic describes some of these components.
- SuiteScript 2.x Anatomy of a Script – All SuiteScript 2.x entry point scripts must conform to the same basic structure. The topic describes that structure.
- SuiteScript 2.x Script Creation Process – You can create a SuiteScript 2.x by following a basic process flow. This topic describes that process flow.
- SuiteScript 2.x Advantages – SuiteScript 2.x is a complete redesign of the SuiteScript model used in SuiteScript 1.0. This topic discusses several of the advantages SuiteScript 2.x has over SuiteScript 1.0.
- SuiteScript 2.x Terminology – Some terms may be defined differently in the context of SuiteScript 2.0 and SuiteScript 2.1. This topic lists and defines these terms.
- SuiteScript 2.x Developer Resources – Several resources are available to help you use SuiteScript 2.x. This topic provides a list of internal and external resources along with a list of help topics specifically for developing SuiteScripts.
- SuiteScript Reserved Words – SuiteScript includes some reserved words which cannot be used as variable or function names in your scripts. This topic discusses these reserved words.
- SuiteScript Versioning Guidelines – This topic describes the SuiteScript versioning system.

Related Topics

3. Related Topics

SuiteScript 2.x
SuiteScript 2.1
SuiteScript 2.x Analytic APIs
SuiteScript 2.x Script Types
SuiteScript 2.x Record Actions and Macros
SuiteScript 2.x JSDoc Validation
SuiteScript 2.x Entry Point Script Creation and Deployment
SuiteScript 2.x Custom Modules
SuiteScript 2.x Scripting Records and Subrecords

1. **Search Bar**: Search for features within the Oracle documentation
2. **Navigation Sidebar:** Related topics are grouped together in categories
3. **Related Topics:** Most articles have links to related topics for further reading

Use the documentation!

Use the documentation for when you need help figuring out how to do various functions.

If you need to get a value, set a value, load records, run searches, or perform any host of actions in SuiteScript, the documentation will likely tell you how to do it!

Understanding the Different Script Types

There are two different categories of script type:

Client Scripts:

- Affects how and what type of data is shown on a NetSuite Page. Can execute as the page is loaded and any time the user enters or updates data on the page.

Server Scripts:

- Provides general processing and saving of data, based on its type

Client Scripts

Documentation

Description: These scripts run in the user's browser when interacting with a NetSuite form.

Use Cases: Real-time form validation, auto-populating fields, custom form behavior.

Example Entry Points:

1. fieldChanged()
   a. Runs when a field is changed on a form by the user or client call, example: Disable or enable fields based on input.
2. validateField()
   a. Also runs when a field is changed, examples: validate field lengths, restrict submitted values to specific range, etc..
3. saveRecord()
   a. Runs when a record is saved (after the submit button is pressed but before the form is submitted), examples: provide alerts before committing data, redirect the user to specific URL, etc…

Server Scripts

Documentation

Provides logic that occurs on the server side. Several common types are listed here:

- Map/Reduce: Process large amounts of data across one or more records. Can be set to a schedule or one-off event

- Portlet: used to create custom portlet's for a user's dashboard

- User Event: Perhaps the most common script type. Perform processing on records during various execution events (when a record is created, loaded, deleted, submitted, etc..)

Server Scripts continued

- Mass Update: Perform custom updates to record fields that are not available through general mass updates.

- RESTlet: Used to import or export data to/from NetSuite. A RESTlet is called from an external application or from within NetSuite itself.

- Scheduled: Used to perform actions based on a specific schedule.

Server Scripts continued

- Suitelet: Allow you to build custom NetSuite pages and backend logic. There are two types:
  - **UI Suitelet:** Create custom pages that look like NetSuite pages (example: A custom page where you can specify shipping info en masse on Sales Orders)
  - **Backend Suitelets:** Do not use any UI objects and execute backend logic. (example: providing a service for backend logic for other SuiteScripts)
- Workflow Action: Create custom workflow actions that native NetSuite workflow actions can’t perform.

Locating Object References and Best Practices

The SuiteScript API Documentation includes references to **NetSuite objects**—these are the fundamental building blocks for interacting with NetSuite records, sublists, and searches.

Key Object Categories:

- **Record Object**: Represents a record in NetSuite, such as an invoice, customer, or item. Contains methods for getting and setting field values, handling sublists, etc.
  - Example: `record.load({type: record.Type.INVOICE, id: 12345})` in SuiteScript 2.0 loads an invoice record for manipulation.

- **Search Object**: Allows querying the NetSuite database to retrieve specific records or data sets.
  - Example: `search.create({type: 'customer', filters: [...], columns: [...]})` to create and execute searches.

- **Sublist Object**: Represents sublists within records, like line items on an order. Manipulating sublists requires understanding how to loop through sublist rows and interact with individual fields.

Best Practices for Documentation Usage

Use Specific Key Terms:

- **Method Names**: If you know the method you're looking for, search directly for it (e.g., `setValue`, `getValue`, `submitRecord`). This will lead you to the exact part of the documentation that discusses the method.

- **Object Names**: If you're working with a specific object, like a `Record` or `Search`, use terms like `N/record` or `N/search` to narrow your results to relevant documentation.

- **Script Type**: Include the script type you're working with, such as "User Event", "Client Script", or "Scheduled Script" along with the term (e.g., "User Event setFieldValue").

Tips for Finding Relevant Documentation

- Bookmark commonly used pages (line N/Record)

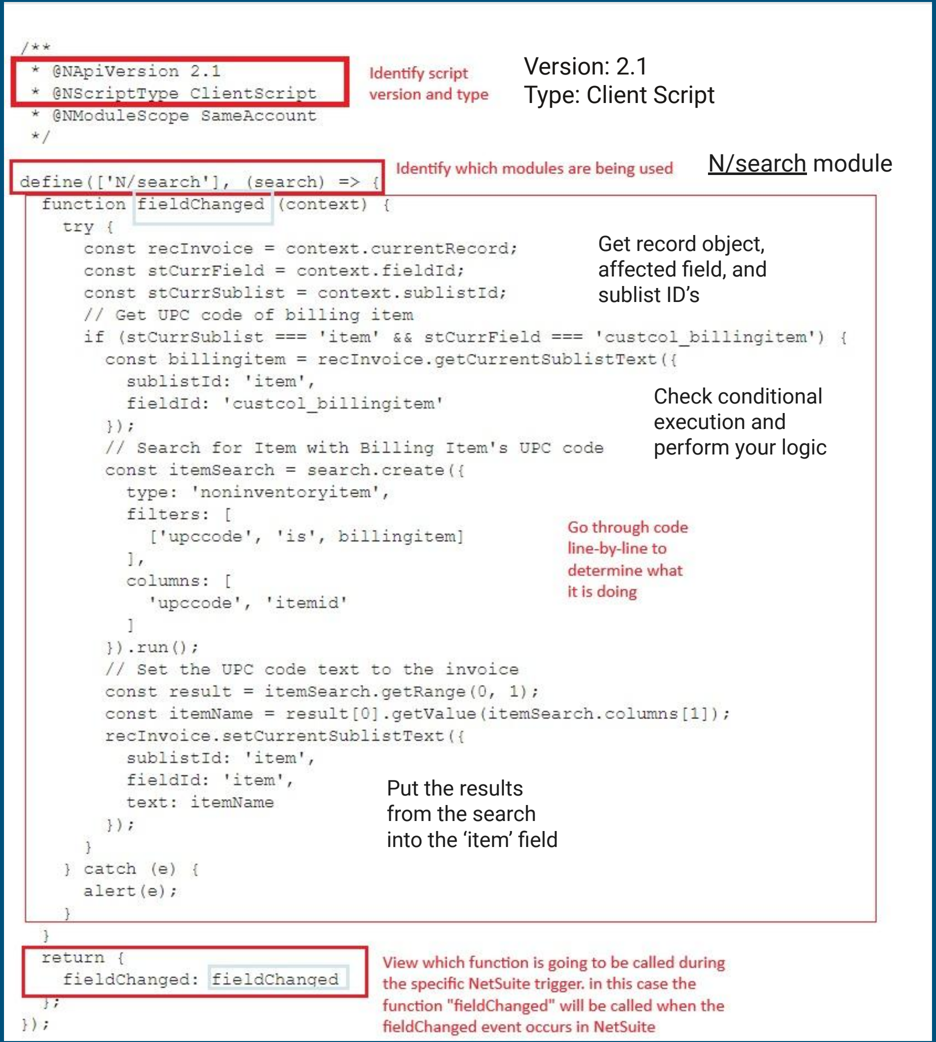
- Google is your friend.
  - “How to get value from record suitescript” adding “NetSuite” or “SuiteScript” will usually bring up the official documentation as the top result.

Tips for Reading SuiteScript Examples

Understanding Code Structure:

- Examples typically follow a **step-by-step structure**, showing how methods are called and in what sequence. Focus on understanding how each method interacts with the `Record`, `Sublist`, or `Search` object.
- Look for **initialization patterns**: In SuiteScript 2.0, scripts use **require** to load modules (`define` function). Understand the basic module import structure to use examples effectively.
- Identify how variables are used to store objects such as records or fields.

Reading SuiteScript Example - Copy a Value to the Item Column



```
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 * @NModuleScope SameAccount
 */

define(['N/search'], (search) => {
  function fieldChanged (context) {
    try {
      const recInvoice = context.currentRecord;
      const stCurrField = context.fieldId;
      const stCurrSublist = context.sublistId;
      // Get UPC code of billing item
      if (stCurrSublist === 'item' && stCurrField === 'custcol_billingitem') {
        const billingitem = recInvoice.getCurrentSublistText({
          sublistId: 'item',
          fieldId: 'custcol_billingitem'
        });
        // Search for Item with Billing Item's UPC code
        const itemSearch = search.create({
          type: 'noninventoryitem',
          filters: [
            ['upccode', 'is', billingitem]
          ],
          columns: [
            'upccode', 'itemid'
          ]
        }).run();
        // Set the UPC code text to the invoice
        const result = itemSearch.getRange(0, 1);
        const itemName = result[0].getValue(itemSearch.columns[1]);
        recInvoice.setCurrentSublistText({
          sublistId: 'item',
          fieldId: 'item',
          text: itemName
        });
      }
    } catch (e) {
      alert(e);
    }
  }
  return {
    fieldChanged: fieldChanged
  };
});
```

Identify script version and type

Version: 2.1
Type: Client Script

Identify which modules are being used

N/search module

Get record object, affected field, and sublist ID's

Check conditional execution and perform your logic

Go through code line-by-line to determine what it is doing

Put the results from the search into the 'item' field

View which function is going to be called during the specific NetSuite trigger. in this case the function "fieldChanged" will be called when the fieldChanged event occurs in NetSuite

Retrieving and Submitting Field Values Using SuiteScript

SuiteScript Field Methods Overview

N/record module is used for working with record objects, providing methods to get and set field values. The key methods are:

- record.load()
- record.getValue()
- record.getText()
- record.setValue()
- record.setSublistValue()
- record.getSublistValue()

load, getValue, getText

```
// Loads and assigns the sales order with the internal
// ID of 1 to the recordObj variable
var recordObj = record.load({
  type: 'salesorder',
  id: 1
});

//fieldValue will equal whatever is inside the memo field
var fieldValue = recordObj.getValue({
  fieldId: 'memo'
});

//fieldText will equal whatever the text is for the status field (ex. Pending Fulfillment)
var fieldText = recordObj.getText({
  fieldId: 'status'
});
```

getLineCount, setSublistValue, getSublistValue

```
//Get the value from the 'custitem_custom_field_1' field from the first line on the item sublist
var sublistValue = recordObj.getSublistValue({
  sublistId: 'item',
  fieldId: 'custitem_custom_field_1',
  line: 0
});

//Get the line count for the 'item' sublist
var itemLineCount = recordObj.getLineCount({sublistId: 'item'});

//Loop through the item sublist and set the value of the custom fields on all lines
// to match the value found on the first line
for (let i = 0; i < itemLineCount; i++){
  recordObj.setSublistValue({
    sublistId: 'item',
    fieldId: 'custbody_custom_field',
    value: sublistValue,
    line: i
  })
}
```

Common Use Cases

```
/*
Retrieving Field Values for Calculations or
Conditional Logic: When developing custom
scripts,
it's common to retrieve a field's value for
calculations or to drive conditional logic.
 For instance, you might fetch a quantity
field to calculate totals or pull a status
field
 to decide whether to update other fields.
*/
var quantity =
recordObj.getValue({ fieldId:
'quantity' });
if (quantity > 10) {
    // perform some logic
}
```

```
/*
Setting Field Values Dynamically During
Data Entry: During data entry or when
records are updated
via scripts, fields often need to be
dynamically set based on the script's
logic. For example,
setting the status or transaction date
field depending on other conditions.
*/
recordObj.setValue({
  fieldId: 'status',
  value: 'APPROVED'
});
```

```
/*
Handling Sublist Fields: When working
with sublists (e.g., line items on a
sales order), you need to
manipulate field values in specific
sublist lines. The N/record module
provides methods like getSublistValue()
and setSublistValue() for retrieving and
setting values in sublists.
*/
var lineItemQty =
recordObj.getSublistValue({
  sublistId: 'item',
  fieldId: 'quantity',
  line: 0
});
recordObj.setSublistValue({
  sublistId: 'item',
  fieldId: 'quantity',
  line: 0,
  value: 15
});
```

Practical Exercises: Fetching and Modifying Field values in NetSuite Records

Hands on Exercise: Retrieve and Modify Field Values

Use Case:

You have various customer types. Management requires all ‘Wholesale’ customers to purchase a minimum quantity of 10 for each item. To decrease user-error you are tasked with creating a Client Script to prevent this from happening.

Step 1. Break down the problem

1. **Understanding the Requirements**:
   - We need to create a client script.
   - The script should target 'Wholesale' customers.
   - It must prevent these customers from purchasing less than a minimum quantity of 10 for any item.

2. **Identifying the Components**:
   - **Customer Type**: We need to check if the customer type is 'Wholesale'.
   - **Item Quantity**: We need to monitor the quantity of each item being purchased.
   - **Validation Logic**: If the customer is 'Wholesale' and the quantity is less than 10, we need to display an error message and prevent the action.

3. Event Trigger:

- The validation needs to happen when the user attempts to save or submit the transaction.

4. Implementation Steps:

1. Load the record
2. Retrieve the current customer's type.
3. Loop through the line items in the transaction.
4. For each item, check if the quantity is less than 10.
5. f the conditions are met, prevent the submission and show an error message.

Step 2. Prepare the Client Script file

Try implementing the solution yourself now! One possible solution is included on the following pages.

```
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 */
define(['N/currentRecord', 'N/ui/message'], function(currentRecord, message) {

    function saveRecord(context) {

    }

    return {
      // This will trigger on the ‘saveRecord’ event
        saveRecord: saveRecord
    };
  });
```

Solution part 1

```
function saveRecord(context) {
        const recObj = currentRecord.get(); // get the current record

        const customerId = recObj.getValue({fieldId: 'entity'}); // Get the customer ID
        let customerType;

        if (customerId) {
            log.debug('CustomerId', customerId);
            let lookupResult = search.lookupFields({
                type: 'customer',
                id: customerId,
                columns: ['custentity_customer_type' ]
            });
            customerType = lookupResult.custentity_customer_type; //get the customer type and assign it to the correct variable
        } else {
            return true; // No customer ID, allow save
        };
…
```

Implementation Steps:

1. Load the record
2. Retrieve the current customer's type.

Solution part 2

```
...

        const itemCount = recObj.getLineCount({ sublistId: 'item' });
        let errorMessage = '';

        if (customerType === 'Wholesale') {
            // Loop through the items if the customer type is Wholesale
            for (let i = 0; i < itemCount; i++) {
                const quantity = recObj.getSublistValue({
                    sublistId: 'item',
                    fieldId: 'quantity',
                    line: i
                });

                if (quantity < 10) {
                    errorMessage = 'Wholesale customers must purchase a minimum quantity of 10 for each item on the sales order';
                    break; // Exit loop on first error
                }
            }
        }
...
```

Implementation Steps:

3. Loop through the line items in the transaction.

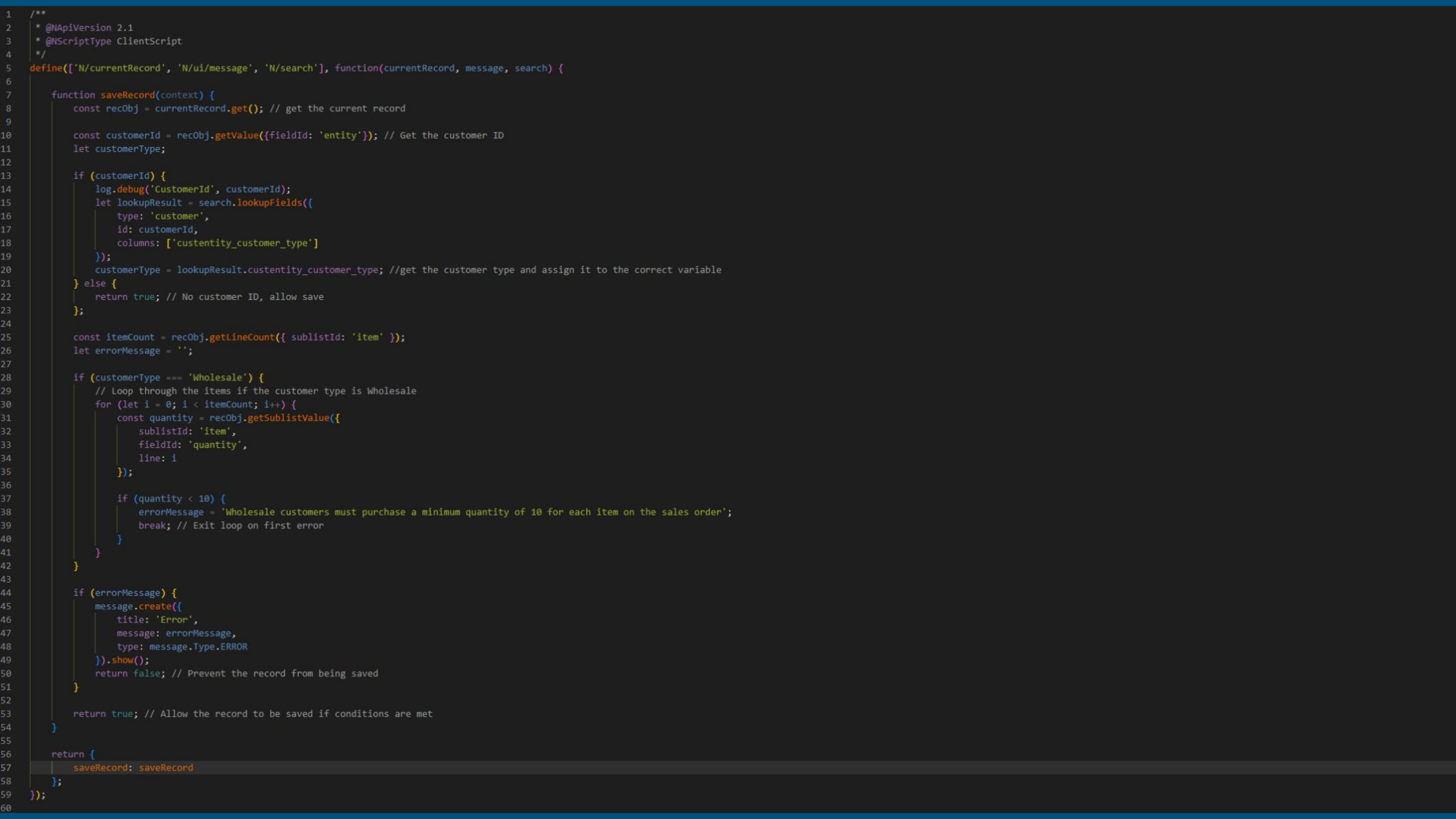
4. For each item, check if the quantity is less than 10.

Solution part 3

```
...
        if (errorMessage) {
            message.create({
                title: 'Error',
                message: errorMessage,
                type: message.Type.ERROR
            }).show();
            return false; // Prevent the record from being saved
        }

        return true; // Allow the record to be saved if conditions are met
    }

    return {
        saveRecord: saveRecord
    };
});
```

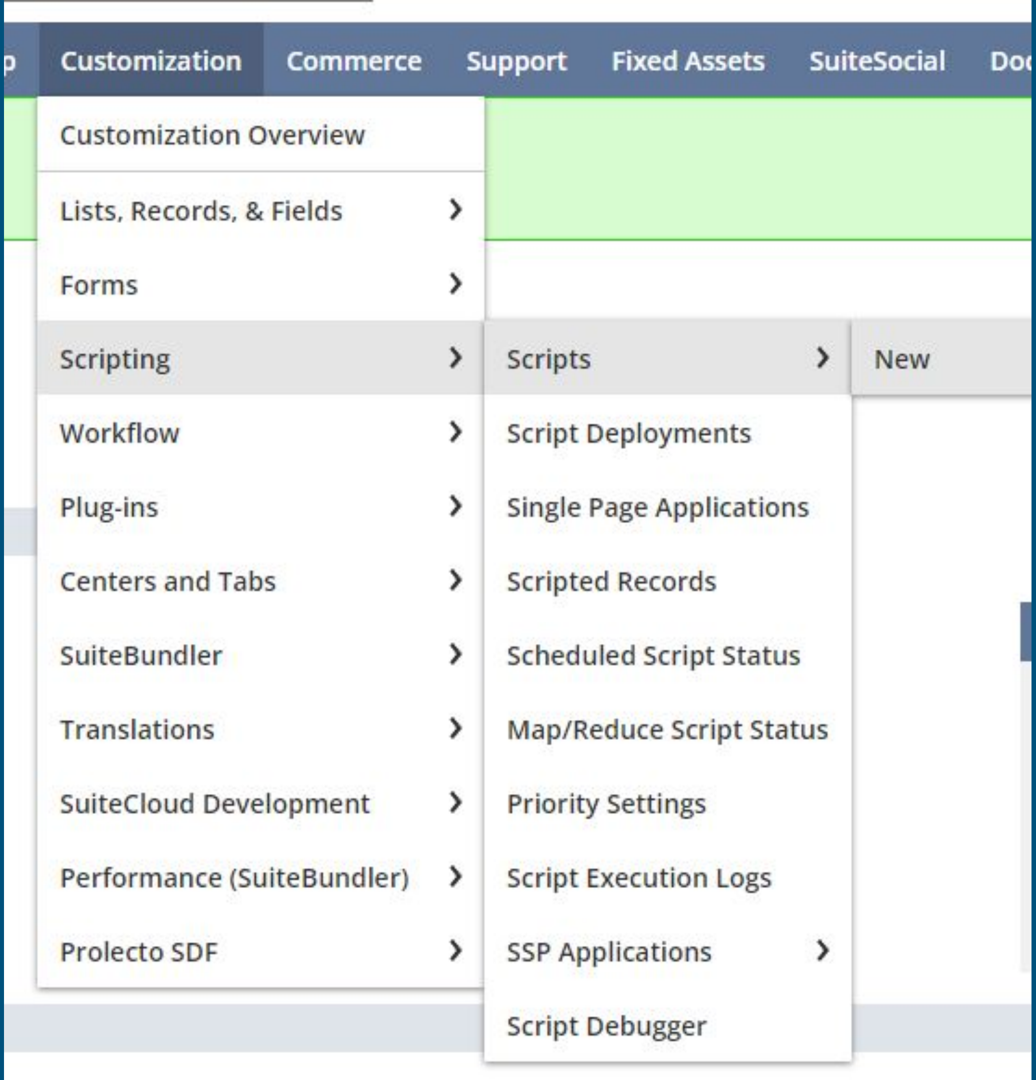
Implementation Steps:

5. If the conditions are met, prevent the submission and show an error message.

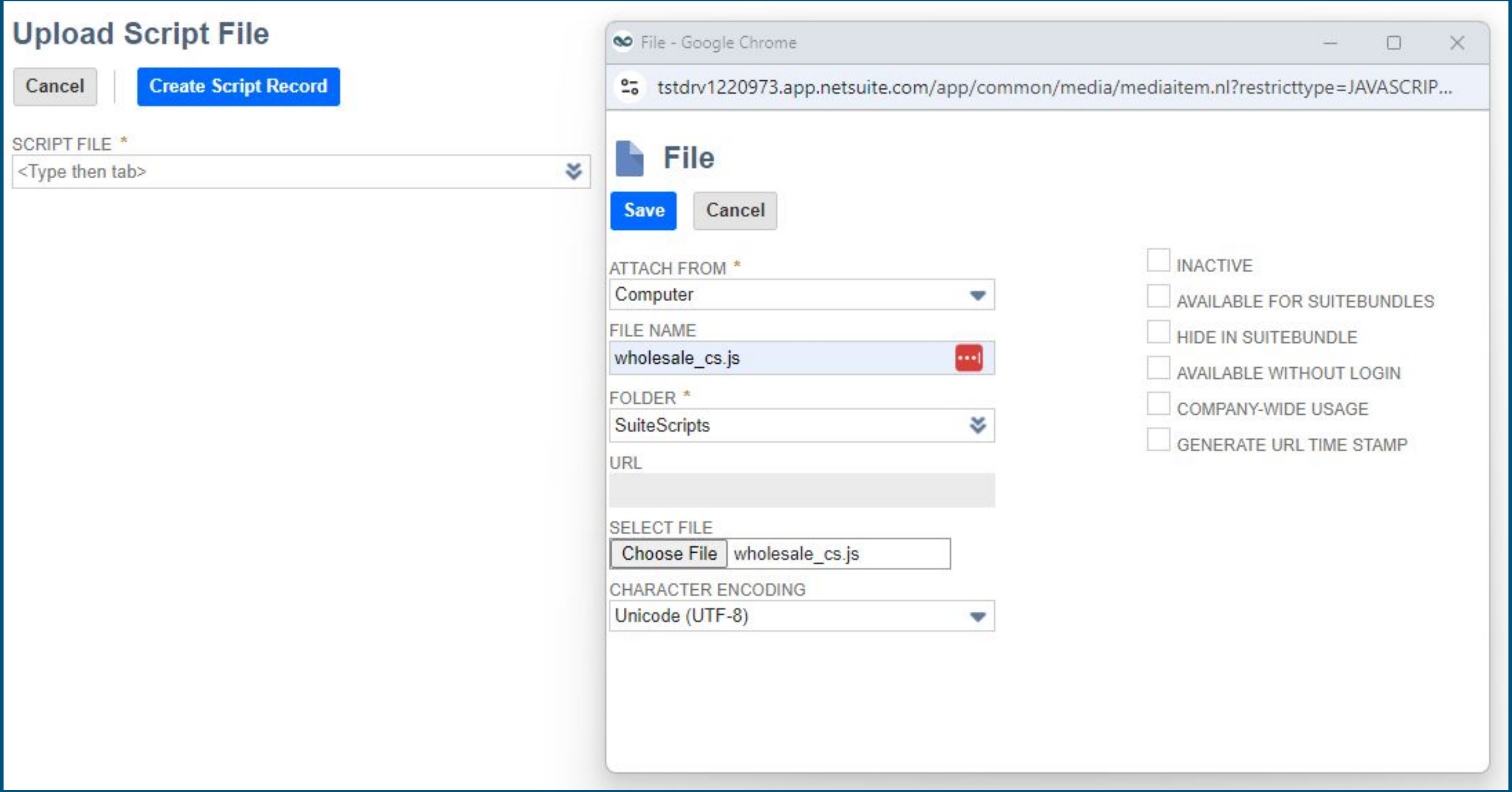


```javascript
/**
 * @NApiVersion 2.1
 * @NScriptType ClientScript
 */
define(['N/currentRecord', 'N/ui/message', 'N/search'], function(currentRecord, message, search) {

    function saveRecord(context) {
        const recObj = currentRecord.get(); // get the current record

        const customerId = recObj.getValue({fieldId: 'entity'}); // Get the customer ID
        let customerType;

        if (customerId) {
            log.debug('CustomerId', customerId);
            let lookupResult = search.lookupFields({
                type: 'customer',
                id: customerId,
                columns: ['custentity_customer_type']
            });
            customerType = lookupResult.custentity_customer_type; //get the customer type and assign it to the correct variable
        } else {
            return true; // No customer ID, allow save
        };

        const itemCount = recObj.getLineCount({ sublistId: 'item' });
        let errorMessage = '';

        if (customerType === 'Wholesale') {
            // Loop through the items if the customer type is Wholesale
            for (let i = 0; i < itemCount; i++) {
                const quantity = recObj.getSublistValue({
                    sublistId: 'item',
                    fieldId: 'quantity',
                    line: i
                });

                if (quantity < 10) {
                    errorMessage = 'Wholesale customers must purchase a minimum quantity of 10 for each item on the sales order';
                    break; // Exit loop on first error
                }
            }
        }

        if (errorMessage) {
            message.create({
                title: 'Error',
                message: errorMessage,
                type: message.Type.ERROR
            }).show();
            return false; // Prevent the record from being saved
        }

        return true; // Allow the record to be saved if conditions are met
    }

    return {
        saveRecord: saveRecord
    };
});
```

Deploying the Script



Upload the script file



Upload Script File

SCRIPT FILE *
<Type then tab>

File - Google Chrome

tstdrv1220973.app.netsuite.com/app/common/media/mediaitem.nl?restricttype=JAVASCRIP...

File

ATTACH FROM *
Computer

FILE NAME
wholesale_cs.js

FOLDER *
SuiteScripts

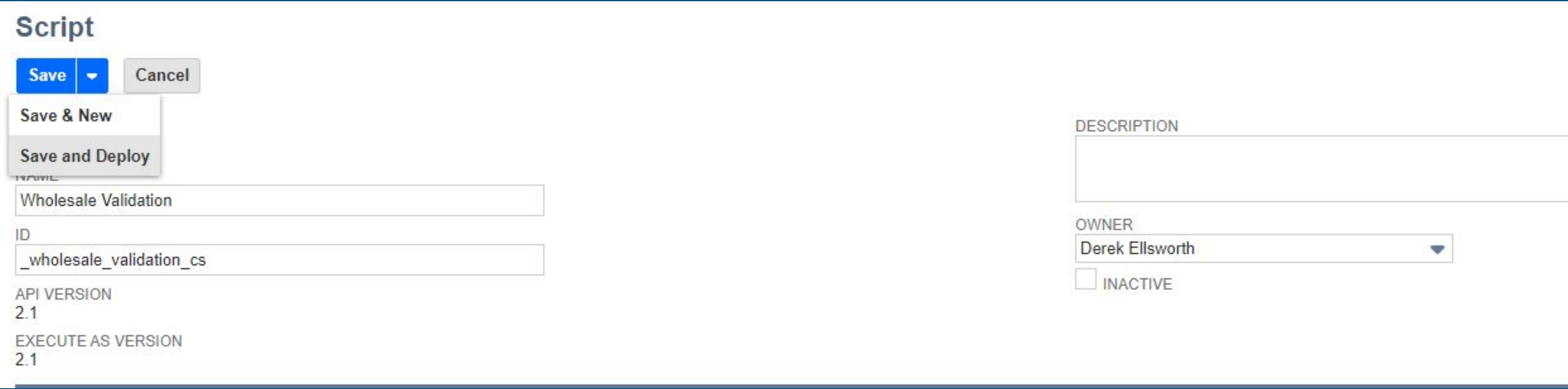
URL

SELECT FILE
Choose File wholesale_cs.js

CHARACTER ENCODING
Unicode (UTF-8)

☐ INACTIVE
☐ AVAILABLE FOR SUITEBUNDLES
☐ HIDE IN SUITEBUNDLE
☐ AVAILABLE WITHOUT LOGIN
☐ COMPANY-WIDE USAGE
☐ GENERATE URL TIME STAMP

Save the Script Record



NAME
Wholesale Validation

ID
_wholesale_validation_cs

API VERSION
2.1

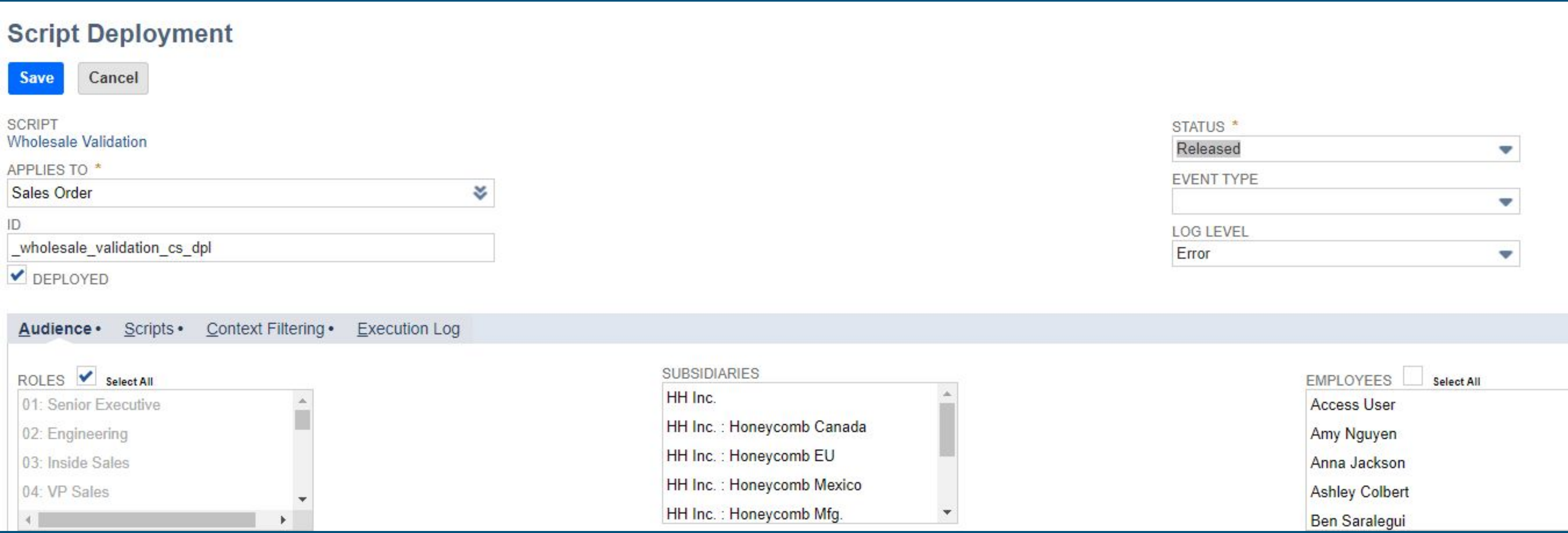
EXECUTE AS VERSION
2.1

DESCRIPTION

OWNER
Derek Ellsworth

INACTIVE

Deploy Script to Sales Order



Script Deployment

SCRIPT
Wholesale Validation

APPLIES TO *
Sales Order

ID
_wholesale_validation_cs_dpl

✓ DEPLOYED

STATUS *
Released

EVENT TYPE

LOG LEVEL
Error

ROLES ✓ Select All

01: Senior Executive
02: Engineering
03: Inside Sales
04: VP Sales

SUBSIDIARIES

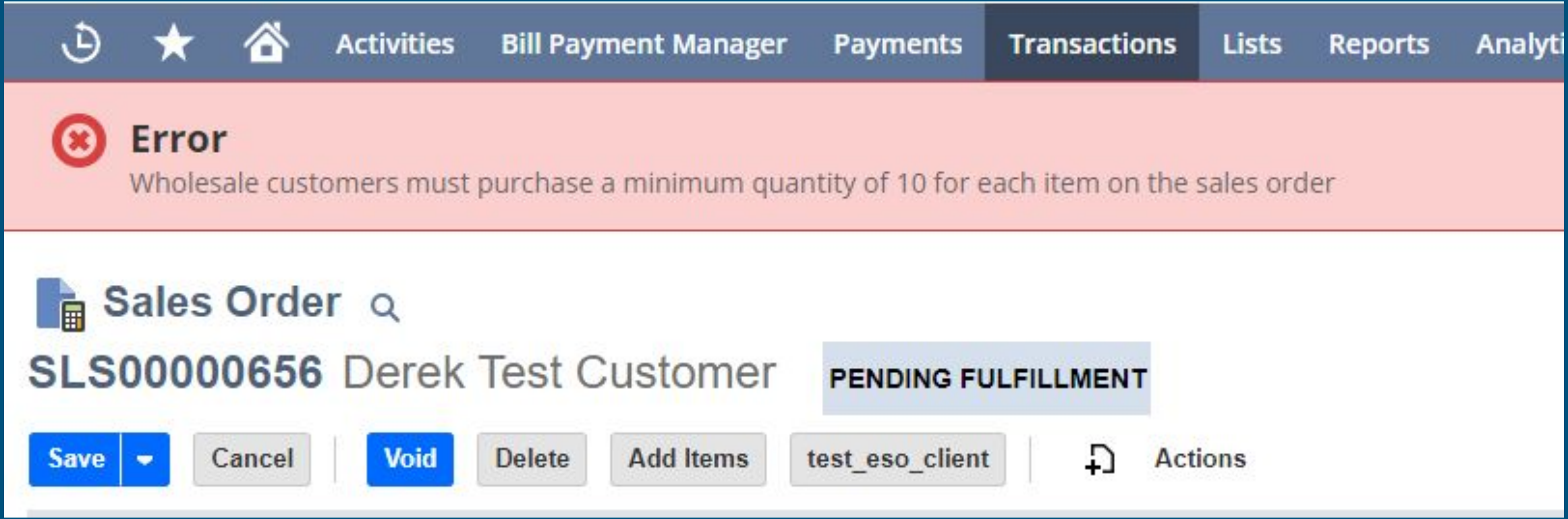
HH Inc.
HH Inc. : Honeycomb Canada
HH Inc. : Honeycomb EU
HH Inc. : Honeycomb Mexico
HH Inc. : Honeycomb Mfg.

EMPLOYEES Select All

Access User
Amy Nguyen
Anna Jackson
Ashley Colbert
Ben Saralegui

Validate the test script!

- Create a test order with a quantity less than 10 on an item and validate that your script is working correctly.



Error

Wholesale customers must purchase a minimum quantity of 10 for each item on the sales order

Sales Order

SLS00000656 Derek Test Customer PENDING FULFILLMENT

Recap and Review

Great job!

In today’s lesson you learned:

1. Navigating the NetSuite SuiteScript documentation
2. Script types
3. Getting and setting field values using built-in SuiteScript methods! (record.getValue(), record.setValue())
4. How to build, deploy, and test your very own client script!