Getting Started with SuiteScript

Create your very first script!

Saved Searches and Their Significance

1.1 What are Saved Searches?

Definition: Saved Searches in NetSuite are a way to create custom reports that allow you to filter and display specific data.

Why do we need to know?: Saved searches are used extensively in SuiteScript to find necessary information to be used by our code!

Key Features:

- Dynamic filtering and sorting.
- Display specific fields and data from different records.
- Set alerts and email notifications based on search criteria.
- Create complex reports using joins and summary types.

1.2 Types of Saved Searches

- **Transactional**: Focus on transactions like invoices, sales orders, and purchase orders.
- **Customer**: For tracking customer data and behaviors.
- **Item**: Focus on inventory and product details.
- **Employee**: Used to manage and report on employee data.

Example Transaction Saved Search

B2B ORDER STATUS OPEN - key accounts - sb

Save & Run | Cancel | Preview | New Template | Change ID | Actions

SEARCH TITLE *
B2B ORDER STATUS OPEN - key accounts - sb

ID
customsearch67663

OWNER *
<Type then tab>

- [x] PUBLIC
- [x] AVAILABLE AS LIST VIEW

- [x] AVAILABLE AS DASHBOARD VIEW
- [ ] AVAILABLE AS SUBLIST VIEW
- [ ] AVAILABLE FOR REMINDERS
- [ ] SHOW IN MENU

Criteria | Results | Highlighting | Available Filters | Audience | Roles | Email | Audit Trail | Execution Log

Use this tab to specify criteria that narrow down your search.

- [ ] USE EXPRESSIONS

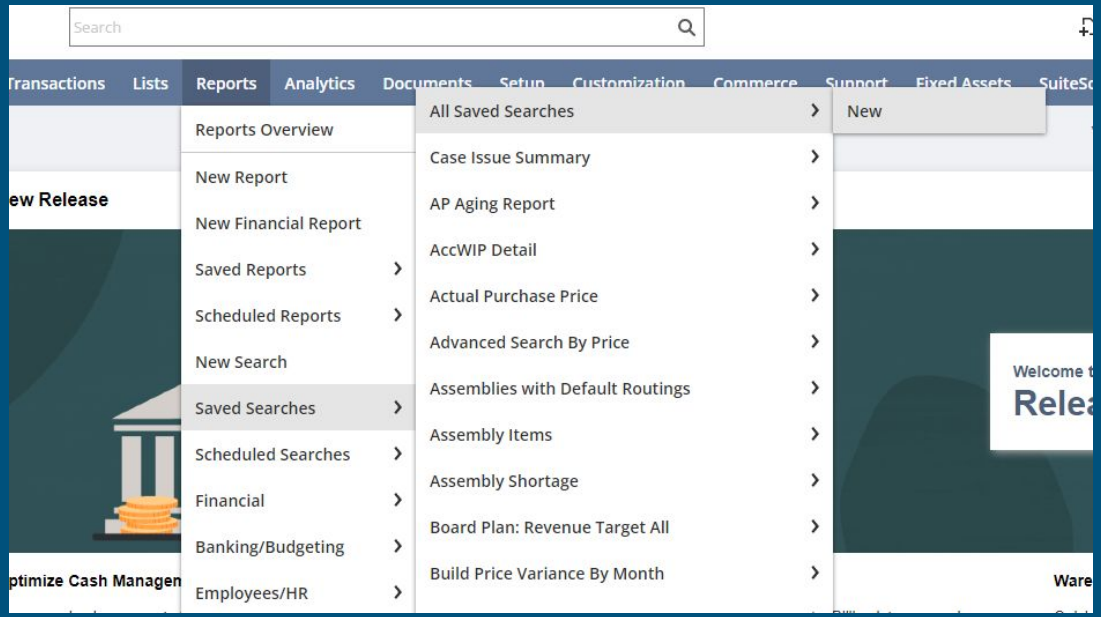
Standard • Summary

FILTER *	DESCRIPTION *	FORMULA
Main Line	is true	
Type	is Sales Order	
Customer : Category	is any of B2B-Open, Distributor, B2B-Closed, Corporate	
Status	is any of Sales Order:Partially Fulfilled, Sales Order:Pending Approval, Sales Order:Pending Billing, Sales Order:Pending Billing/Partially Fulfilled, Sales Order:Pending Fulfillment	

Add | Cancel | Insert | Remove

1.3 Building a Saved Search

1. Navigate to Reports > Saved Searches > all Saved Searches > New



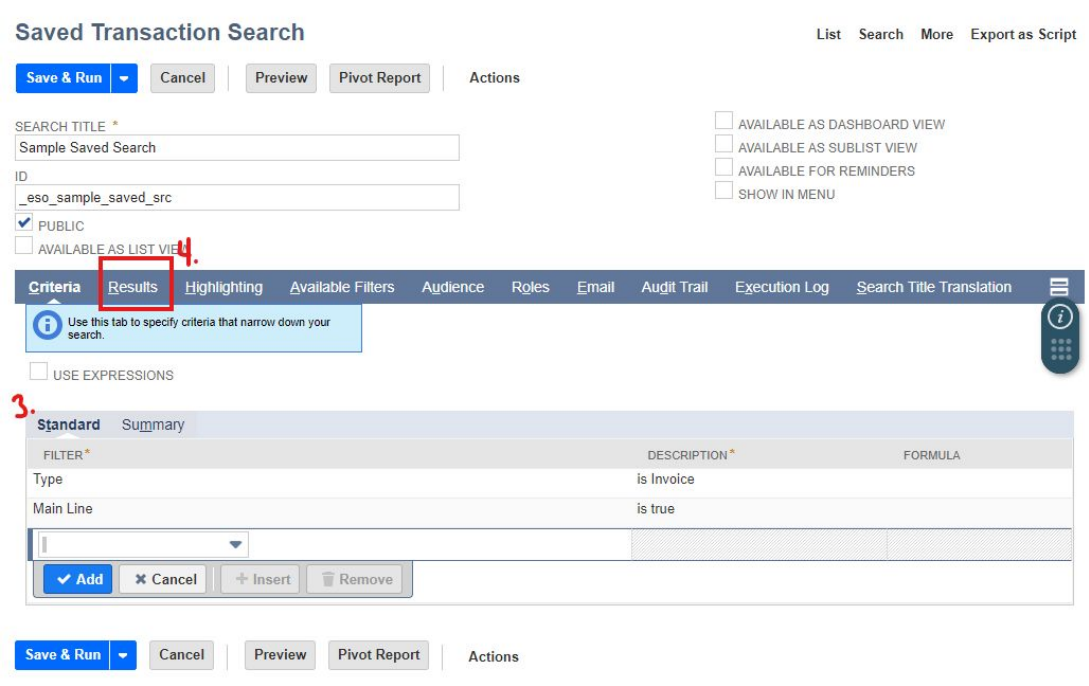
1.3 Building a Saved Search

2. Select type of search (e.g., Transaction, Customer).

3. Define basic Criteria (filters)

4. Define results columns (the data you want to see)

5. Save and Run the Search



Introduction to SuiteScript

Types Of SuiteScripts

Most Common script types

1. User Event Scripts
2. Map / Reduce Scripts
3. Client Scripts

Additional Resources:

- All SuiteScript Types
- Official SuiteScript Documentation
- Official “Getting Started” page

User Event Scripts

User Event Scripts are triggered by record-level events such as Create, Edit, View, Delete, or Submit. They are used to perform actions before or after a record is processed.

Example: Automatically set a default sales rep on a Sales Order when it is created

```
function beforeSubmit(context) {
    var salesOrder = context.newRecord;
    salesOrder.setValue('salesrep', 12345); // Set default sales rep ID
}
```

Map / Reduce Scripts

Map/Reduce Scripts are used for processing large data sets in parallel to enhance performance. They are ideal for complex data transformations and bulk record processing.

Example: Update the price of all items in the inventory based on a percentage increase.

```
function map(context) {
    let item = JSON.parse(context.value);
    let newPrice = item.price * 1.10 // Increase price by 10%

    // Sets the value of rate on the Item sublist of a record
    record.setSublistValue({
        sublistId: "item",
        fieldId: "rate",
        value: newPrice
    })
}
```

Client Scripts

Client Scripts run on the client-side (browser) and are used to validate data, automate user interactions, and enhance the user experience on forms and records.

Example: Validate that a required field is filled before allowing a user to save a form.

```
function saveRecord(context){
    let approvedCheckbox = context.currentRecord.getValue('custbody_approved_checkbox');
    if (!approvedCheckbox){ // if the checbox is not checked the alert will fire
        alert('Please approve')
        return false; // Prevents Saving
    }
    return true;
}
```

SuiteScript Modules

NetSuite provides pre-built modules that you can use to lower development time and complexity.

Common SuiteScript Modules

All Modules

- **N/record**
  - **Purpose:** Used for creating, loading, copying, deleting, and manipulating NetSuite records.
  - **Common Uses:** Creating new records (e.g., sales orders, customers), updating fields, and deleting records.
- **N/search**
  - **Purpose:** Allows you to perform saved searches programmatically to find and retrieve records.
  - **Common Uses:** Querying data, retrieving lists of records, and working with search results.
- **N/log**
  - **Purpose:** Used for logging information, errors, or debug data during script execution.
  - **Common Uses:** Logging messages to help debug scripts or track script execution.

Formatting your SuiteScript

Before you begin writing your SuiteScript you must format the file so NetSuite knows what to do with it.

Let's take a look

Here is a very simple User Event Script that will set the memo of every transaction record that this script is applied to

```
/**
 * @NApiVersion 2.1
 * @NScriptType UserEventScript
 */

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

    function beforeLoad(context) {
        let recordObject = record.load({
            type: 'salesorder',
            id: 12345 // loads the record with the internal id of 12345
        });

        recordObject.setValue({
            fieldId: 'memo',
            value: 'SuiteScript is cool!'
        })
    }

    return {
        beforeLoad: beforeLoad
    };
});
```

1. Header

2. Module Dependencies

3. Custom Logic

4. Specify entry point function

Header

```
/**
 * @NApiVersion 2.1
 * @NScriptType UserEventScript
 */
```

- This tells NetSuite what script type and version you are using.
- Must be inside a comment block
  - /* */

@NApiVersion
- 2.0
- 2.1
- 2.x

@NScriptType
- UserEventScript
- ClientScript
- MapReduceScript
- WorkflowActionScript
- etc..

Module Dependencies

```
define(['N/record', 'N/log'], function(record, log) {
    // the rest of the code lives here
    return {
        beforeLoad: customFunction
    };
}
```

The define() function

a. Takes an array [ ] of dependencies (modules)
b. The modules are given as strings ‘N/record’, ‘N/log’, etc…

Callback function

a. Executed once the dependencies are loaded.
b. The parameters are the modules given in your define function, in the same order. This lets you use them in your code

Custom Logic

"This is where the magic happens"

- Write your custom functions and code to execute here.

```
define(['N/record', 'N/log'], function(record, log) {

    function beforeLoad(context) {
        let recordObject = record.load({
            type: 'salesorder',
            id: 12345 // loads the record with the internal id of 12345
        });

        recordObject.setValue({
            fieldId: 'memo',
            value: 'SuiteScript is cool!'
        })
    }

    return {
        beforeLoad: beforeLoad
    };
});
```

Specify point of entry

This is a vital step in any SuiteScript.

This is the return of the callback function

You designate the entry point and tell it which function to use.

```
define(['N/record', 'N/log'], function(record, log) {

    function customFunction(context) {
        let recordObject = record.load({
            type: 'salesorder',
            id: 12345 // loads the record with the internal id of 12345
        });

        recordObject.setValue({
            fieldId: 'memo',
            value: 'SuiteScript is cool!'
        })
    }

    return {
        beforeLoad: customFunction
    };
});
```

Try it out yourself!

Objective: Whenever a new invoice is created for a customer, we want to update a custom field called ‘Total Number of Invoices’ on the customer record.

Copy + Paste this template into your text editor

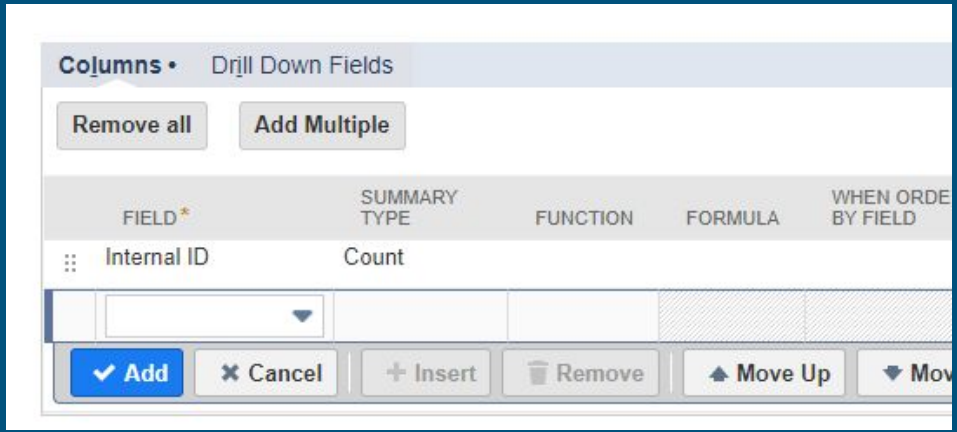
```
/**
 * @NApiVersion 2.1
 * @NScriptType UserEventScript
 */

define([], function() {

    function updateTotalNumberOfInvoices (context){
    }

    return {
        beforeSubmit : updateTotalNumberOfInvoices
    };
});
```

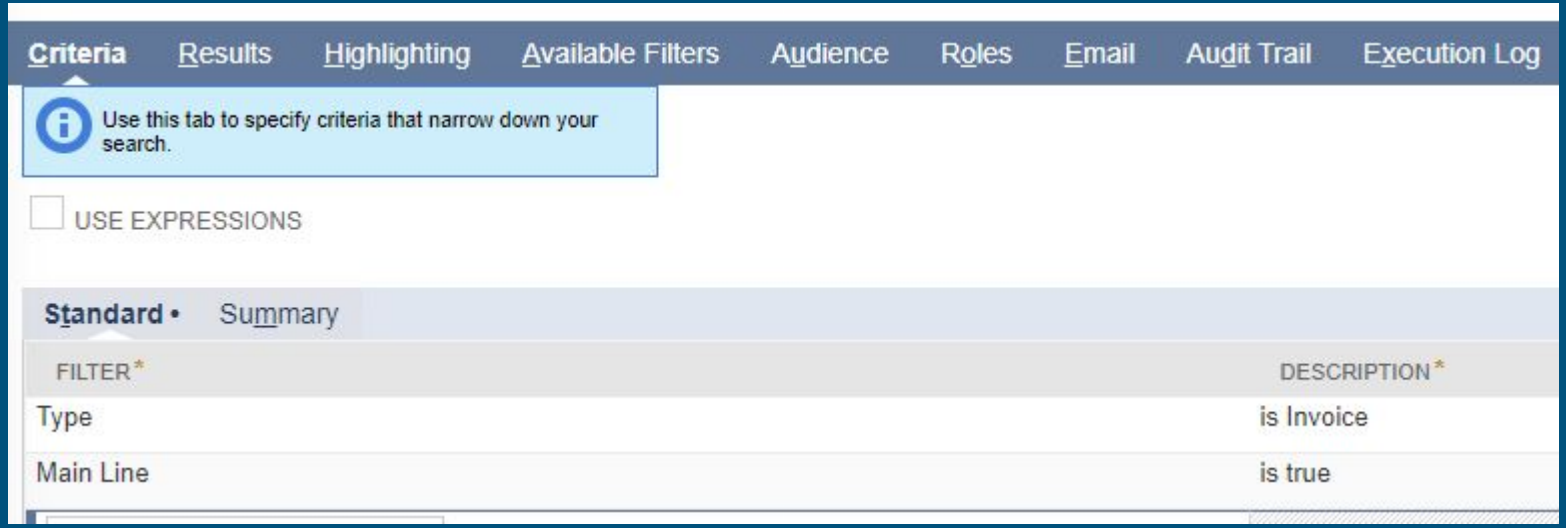
First we will create the saved search we will be using inside of NetSuite

1. Create a Saved Search in NetSuite that has the desired information

2. Take note of the ID

These are the criteria / results for the saved search I will be using for the script.





Next, we Identify which modules you will need to use. Add them to the dependencies.

1. N/record
   a. For loading + manipulating the customer record
2. N/search
   a. For obtaining required information
3. N/log
   a. For logging information to NetSuite

```
/**
 * @NApiVersion 2.1
 * @NScriptType UserEventScript
 */

define(['N/record', 'N/search', 'N/log', function(record, search,
log){

    function updateTotalNumberOfInvoices (context){
    }

    return {
        beforeSubmit : updateTotalNumberOfInvoices
    };
});
```

Next, we specify when we want the script to run.

1. Identify your desired entry point
   - We will be using **beforeSubmit** which runs right before the record is submitted to the server.

2. Identify your desired event context type
   - We want to update the customer record every time an invoice is **created** or **deleted**

```
/**
 * @NApiVersion 2.1
 * @NScriptType UserEventScript
 */

define(['N/record', 'N/search', 'N/log'], function(record, search, log) {

    function updateTotalNumberOfInvoices (context){
        if (context.type !== context.UserEventType.CREATE && context.type !== context.UserEventType.DELETE) {
            return; // This will return (end) the addCustomerInformation function if the context type isn't one of the listed types.
        }
    }

    return {
        beforeSubmit : updateTotalNumberOfInvoices

    };
});
```

Here we check the context type, if it isn't create or delete we end the script early to save resources. The return keyword will end whatever function it is inside of. If there is a value to return you can specify after the return keyword (return true, return internalId, etc..)

Here we specify that on the beforeSubmit event, NetSuite will try to run the updateTotalNumberOfInvoices function

Start entering in our custom business logic

```
define(['N/record', 'N/search', 'N/log'], function(record, search, log) {

    function updateTotalNumberOfInvoices(context){
        if (context.type !== context.UserEventType.CREATE && context.type !== cor
            return;
        };

        let invoiceRecord = context.newRecord; //context.newRecord is the record being submitted to the server

        let customerId = invoiceRecord.getValue({
            fieldId: 'entity' //this gives us the internal ID of the customer on the invoice record
        });

        let searchObj = search.load({
            id: 'customsearch_eso_total_invc'
        });

        // Create a new search Filter using the N/search module
        let customerFilter = search.createFilter({
            name: 'entity',
            operator: search.Operator.IS,
            values: [customerId]
        })

        // Add a filter to the search to only include invoices for the specific customer
        searchObj.filters.push(customerFilter);

    };

    return {
        beforeSubmit: updateTotalNumberOfInvoices

    };
});
```

Create a reference (variable) to the record being submitted (in this case it is the invoice record)

getValue() can be used on any record to get the value for specific fields. This is using the N/record module

Using N/search we can load the saved search we created earlier

The N/search module has a .createFilter() function we can use to create a new filter when running the search in our script

We use .push() to add new elements to arrays!
(searchObj.filters is an array)

Now that the search is ready lets run it and get the results

```
    let searchObj = search.load({
        id: 'customsearch_eso_total_invc'
    });

    // Create a new search Filter using the N/search module
    let customerFilter = search.createFilter({
        name: 'entity',
        operator: search.Operator.IS,
        values: [customerId]
    })

    // Add a filter to the search to only include invoices for the specific customer
    searchObj.filters.push(customerFilter);


    // This will run the updated searchObj and get us the results from just the first line (the search is a count of internal IDs so there will only be 1 line)
    let searchResult = searchObj.run().getRange({ start: 0, end: 1 });

    if (searchResult.length > 0) {
        let invoiceCount = searchResult[0].getValue(searchObj.columns[0])
        log.debug({
            title: 'Invoice Count',
            details: invoiceCount
        });
    };
};
```

.run() and getRange() are both part of the N/search module

Here we check if we have any results. If we do we assign the results to a variable and log the variable to NetSuite!

We've added a lot! Let's get it into NetSuite and test it out.

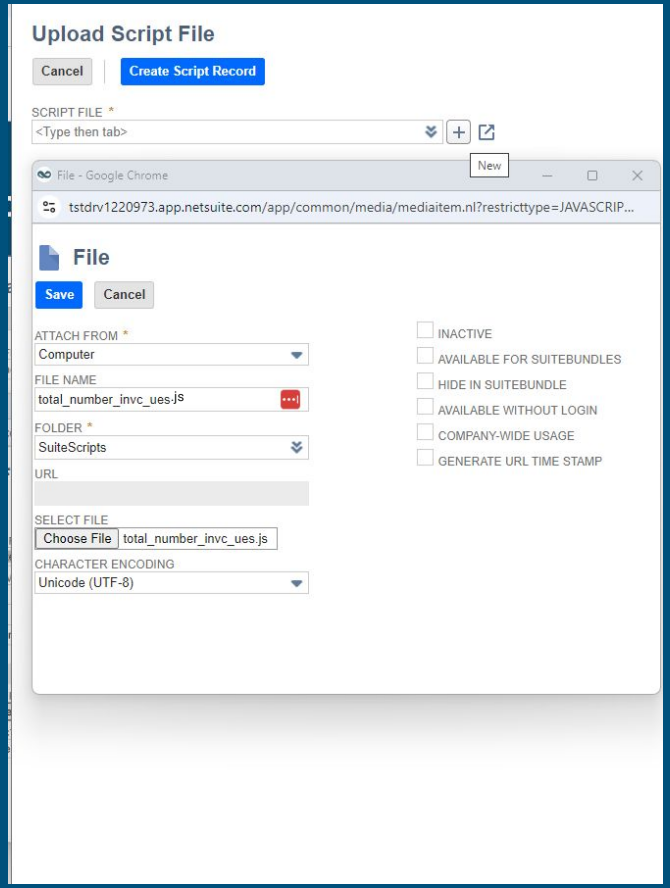
1. Save the file to your computer and give it a descriptive title.

2. Open NetSuite and navigate to Customization -> Scripting -> Scripts -> New

3. Select the '+' and select the file you just saved to your computer.

4. Under FILE NAME make sure you name it **EXACTLY** how the file is saved on your computer

5. Click Save and Create Script Record

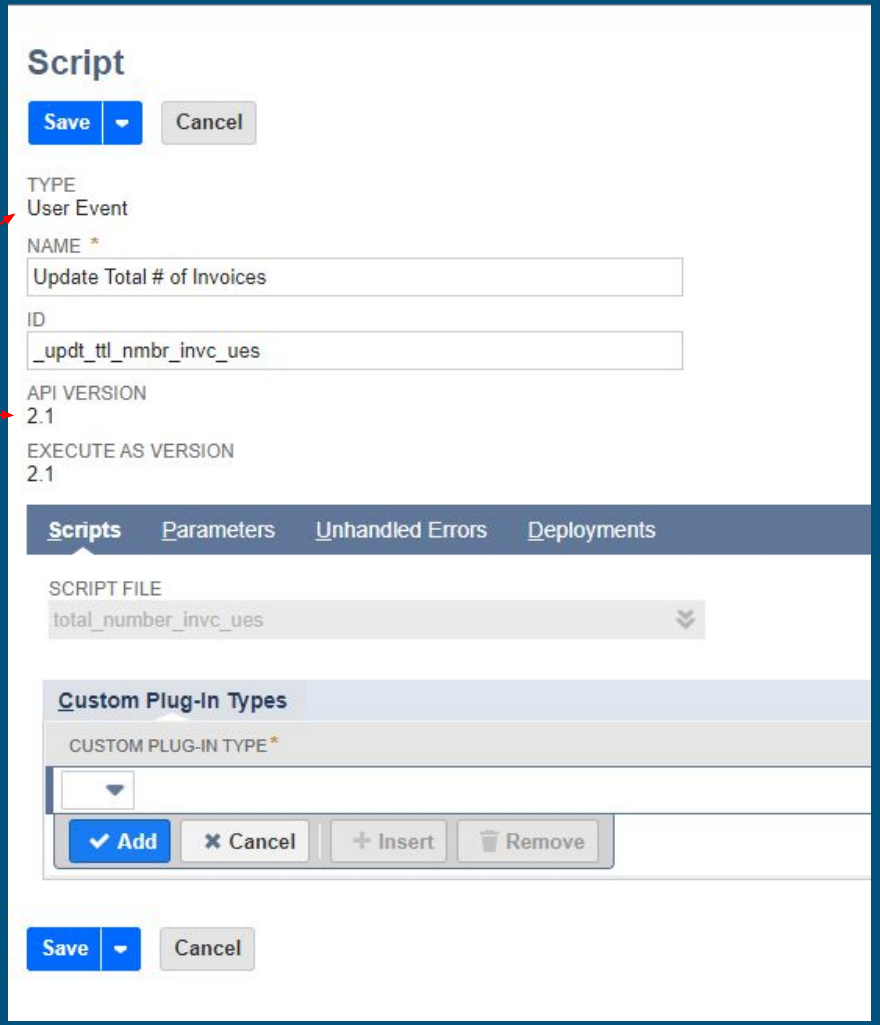


You will be taken to this screen

Name your script and give it an ID.

NetSuite will automatically detect your script type and version using the headers on your script!

Click the drop down arrow next to save and choose ‘Save and Deploy’



Script Deployment

Save | Cancel

SCRIPT
Update Total # of Invoices

Choose what record type this deployment applies to

APPLIES TO *
Invoice

ID
_updt_nmbr_invc_ues_dply

☑ DEPLOYED

Give it an ID and make sure Deployed is checked

Change the status to released

STATUS *
Released

EVENT TYPE

Since we are using log.debug in our code change the log level to Debug

LOG LEVEL
Debug

EXECUTE AS ROLE
Administrator

Select what role you want this script to execute as. Make sure it has all necessary permissions.

Audience • Scripts • Context Filtering • Execution Log

ROLES ☑ Select All
01: Senior Executive
02: Engineering
03: Inside Sales
04: VP Sales

The audience determines which accounts can run a deployed script.

DEPARTMENTS
Admin
Consumer Electronics
Engineering
Executive Team
Marketing

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

GROUPS
3D Printer
Assembly Cell
Corporate East
Corporate West
Finishing

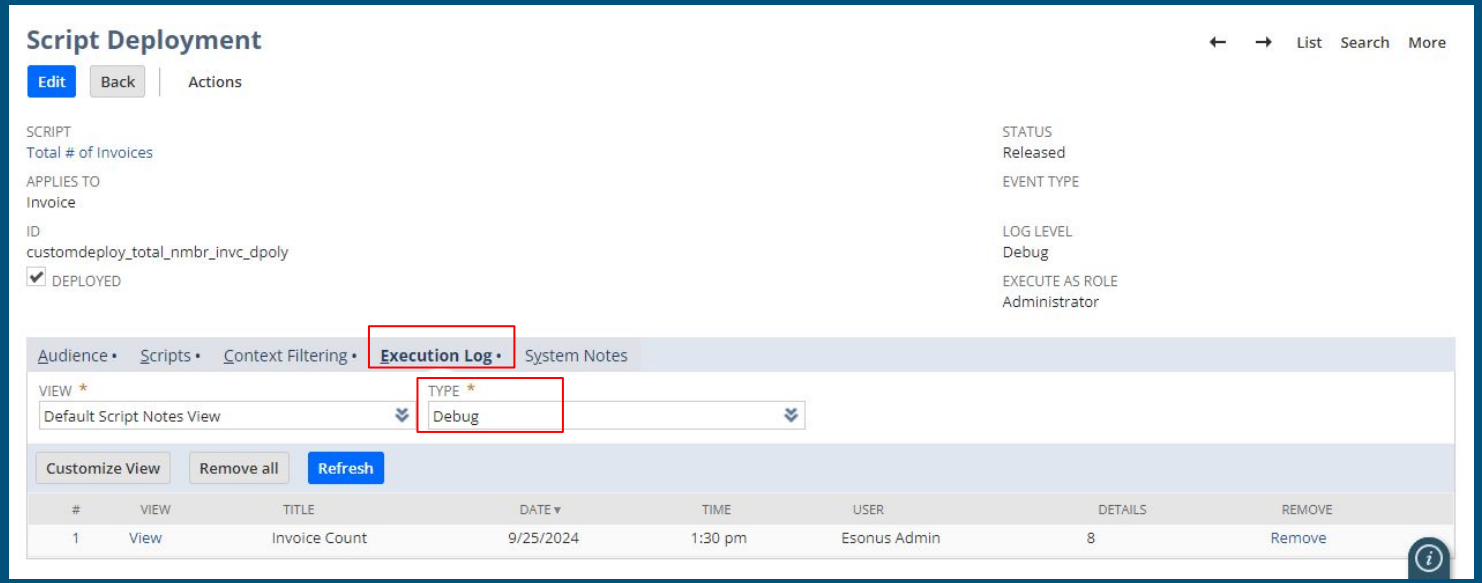
EMPLOYEES ☐ Select All
Access User
Amy Nguyen
Anna Jackson
Ben Saralegui
Brad M Sparling

PARTNERS ☐ Select All
Jasper Supply

Now we are ready for an initial test!

Stay on the deployment record you just saved and create a new invoice in NetSuite.

You should now see a log of the invoice count for your customer! We are almost done!



```
//Additional code above

        // Create a new search Filter using the N/search module
        var customerFilter = search.createFilter({
            name: 'entity',
            operator: search.Operator.IS,
            values: [customerId]
        })

        searchObj.filters.push(customerFilter);

        var searchResult = searchObj.run().getRange({ start: 0, end: 1 });

        if (searchResult.length > 0) {
            var invoiceCount = searchResult[0].getValue(searchObj.columns[0])

            // Load the Customer record to update the field
            let customerRecord = record.load({
                type: record.Type.CUSTOMER,
                id: customerId,
                isDynamic: true
            });

            customerRecord.setValue({
                fieldId: 'custbody_nmbr_invcs',
                value: invoiceCount
            });
            customerRecord.save();
        };
    };
    return {
        beforeSubmit: updateTotalNumberOfInvoices

    };
});
```

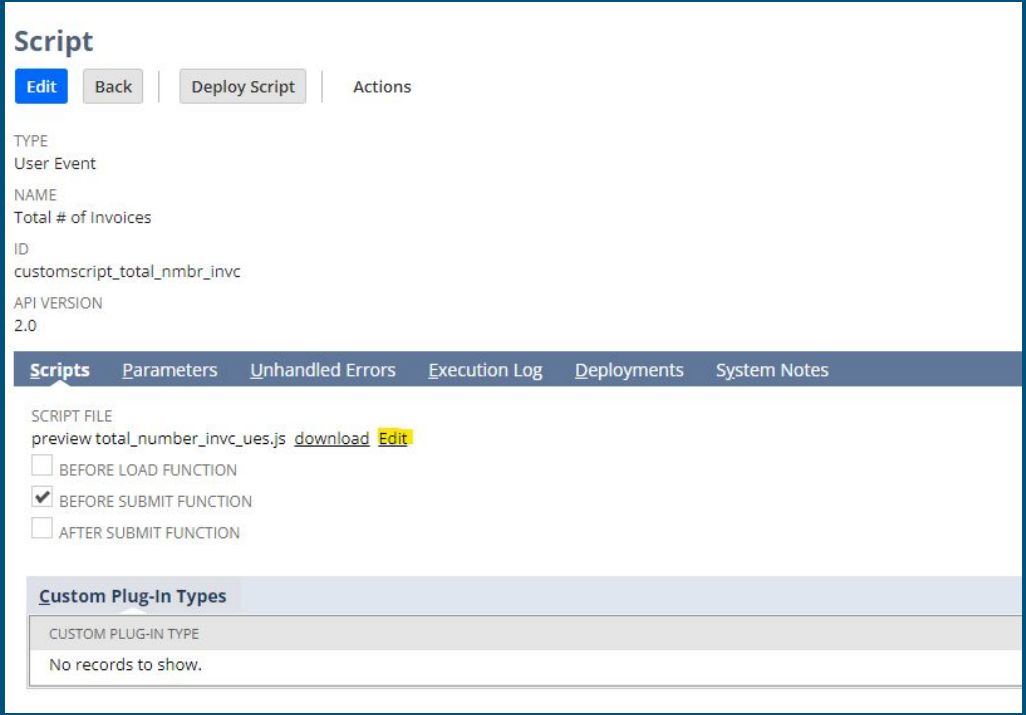
Now we know our search logic works! Let's remove the log.debug and replace it with the rest of our code

Let's load the customer record using the customer ID we received earlier

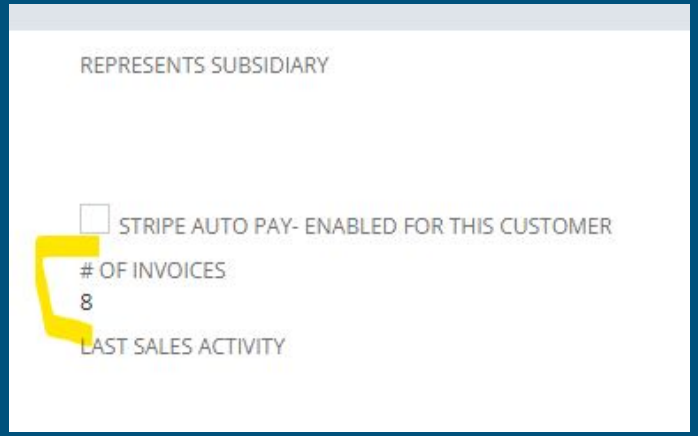
Now we set the value of the custom field on our customer record

Then we save the customer record and we are all set!

We are almost done! Go back to the script record, update the script file with your updated code and run it again (by either creating a new invoice or deleting one)!



Congratulations you successfully created your first user event script!



But what if Kevin from accounting keeps messing with your saved search?

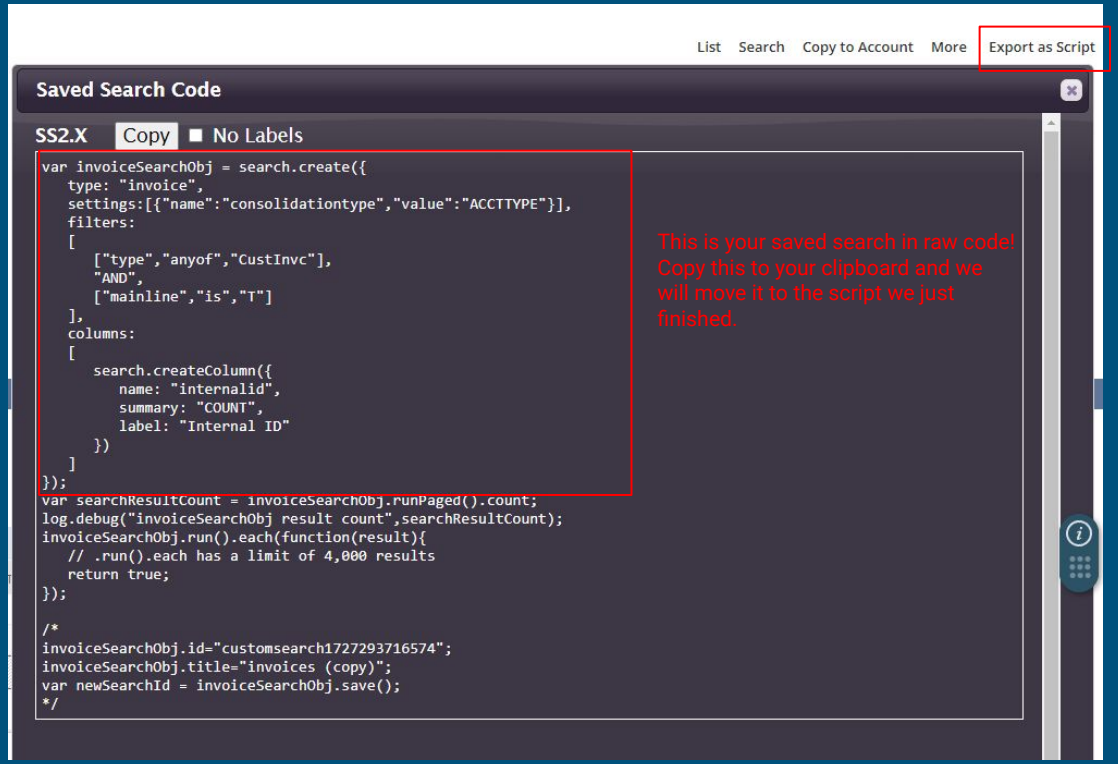
You can create entire searches using pure code!

The easiest way to do this is to create the search you want in the UI.

[Download this Chrome extension]

And export your saved search as code!

Once you save your saved search and download the chrome extension. This button will be available on the 'edit' view of your saved search.



This is your saved search in raw code! Copy this to your clipboard and we will move it to the script we just finished.

```
// Additional code Above

// We can remove or comment out our old search.load() code.
 var searchObj = search.load({
    id: 'customsearch_eso_total_invc'
 });

let searchObj = search.create({
    type: "invoice",
    settings:[{"name":"consolidationtype","value":"ACCTTYPE"}],
    filters:
    [
       ["type","anyof","CustInvc"],
       "AND",
       ["mainline","is","T"]
    ],
    columns:
    [
       search.createColumn({
          name: "internalid",
          summary: "COUNT",
          label: "Internal ID"
       })
    ]
 });

// Additional code below
```

Remove this part of the code and replace it with the hard-coded version of the search

Update the script file in NetSuite, run it again and confirm that the results are the same!

Congratulations

You successfully created your very first User Event Script!