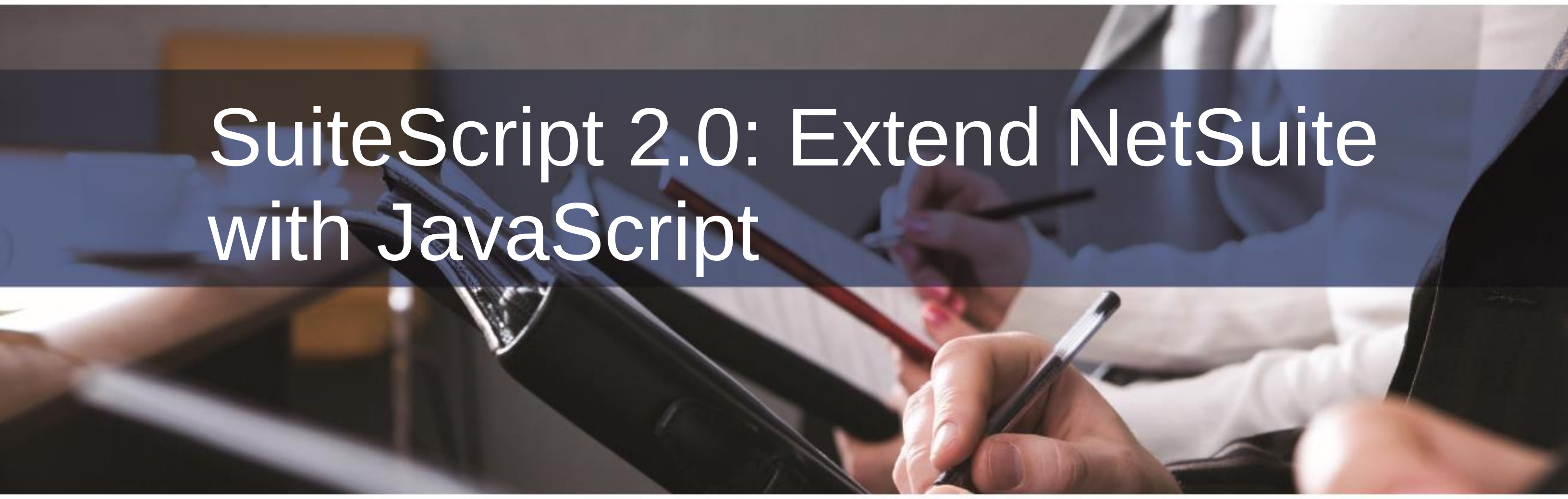
ORACLE
NETSUITE



SuiteScript 2.0: Extend NetSuite with JavaScript

Student Guide

This printing: August 2021.

Oracle NetSuite
500 Oracle Parkway
Redwood Shores, CA 94065

TABLE OF CONTENTS

BEFORE YOU BEGIN

Important Notes

⚠ **IMPORTANT**

This student guide uses functionality not covered in the videos.

If you need assistance with newly introduced syntax, please refer to the Help Center.

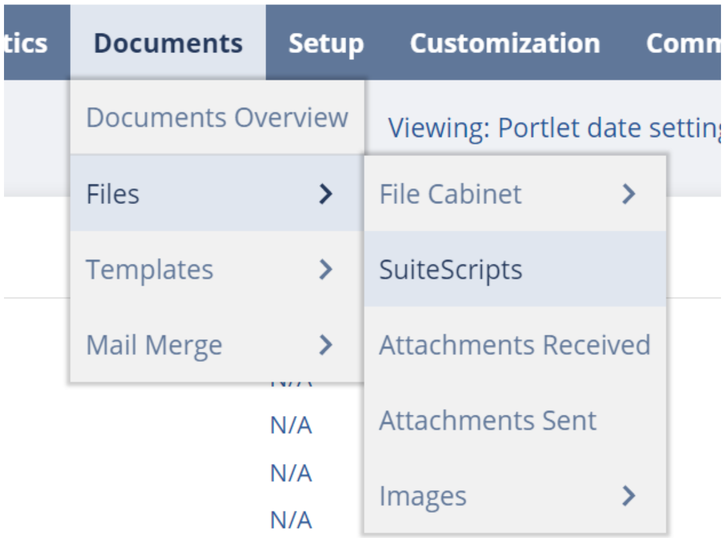
⚠ **IMPORTANT**

Eclipse IDE has been deprecated. Use a different IDE of your choosing.

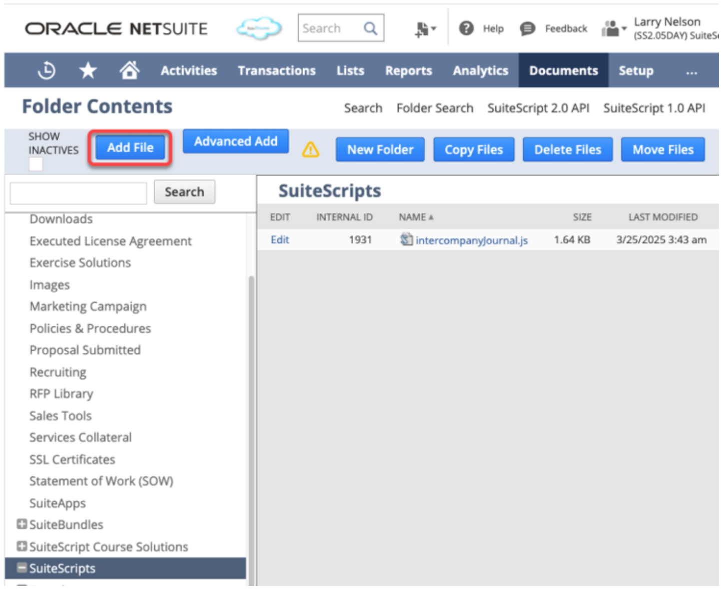
Refer to the below steps on how to upload files manually.

Uploading Files Manually

1 Go to **Documents > Files > SuiteScripts**.



2 Click on the **Add File** button on the top left and select the file to be uploaded.



SuiteScripts

EDIT	INTERNAL ID	NAME ▴	SIZE	LAST MODIFIED
Edit	1931	intercompanyjournal.js	1.64 KB	3/25/2025 3:43 am

Note: if there is a prompt for file replacement, click Yes.

INTRODUCTION TO SUITESCRIPT

MODULE EXERCISES

Required Exercises	
01	Overview of Technical Components
Optional Exercises	
02	Adjusting NetSuite Preferences
03	Reviewing the Basics of NetSuite Navigation

EXERCISE 01: Overview of Technical Components (Required)

Scenario: NetSuite supports several avenues to customizing applications. Gain an understanding of the overall ways that applications can be customized. This will be expanded on throughout the course.

View a SuiteScript in the File Cabinet

1. Go to **Document > Files > SuiteScripts**.
2. Click on the `intercompanyJournal.js` file to view script.

Note: The behavior will be different across browsers. In some, the file will open directly and in others you are prompted to download. Open the file in your favorite text editor to view it.

This script is using the SuiteScript 1.0 syntax. If you're part of a company that created SuiteScripts before 2015.2, you will notice that the scripts look like this. For this course, we'll be concentrating on the SuiteScript 2.0 syntax.

Inspect a Field ID

3. Hover over the home icon then click **Set Preferences**.
4. In the **General** subtab, look for the **Defaults** section and make sure that the SHOW INTERNAL IDS is checked.

BEST PRACTICES

This option exposes the internal ids of records and fields to the user. While usually not relevant to end users, this information is invaluable for SuiteScript developers and must always be enabled.

5 Go to **Lists > Relationships > Customers**.

6 Notice the Internal ID column? This is where you will get the internal ids of records. We'll be using this information to load a specific record.

Note: If you don't see any values, expand the FILTERS at the top of the page and make sure the SALES REP is set to **- All -**.

7 Open the customer record for **ABC Marketing Inc** by clicking on the **View** link.

8 Click on the WEB ADDRESS field label.

9 Look for the Field ID. This ID will be used when referring to a field value from the script.

EXERCISE 02: Adjusting NetSuite Preferences (Optional)

Scenario: You may want to adjust how web pages in NetSuite display dates, time zone, as well as the default language.

> **Note**: The training materials assume the **English (U.S.)** language, but you can switch between **English (U.S.)** and **English (International)**.

Adjust Several Preferences

1 Navigate to **Home > Set Preferences**.

2 On the **General** subtab (this is the default subtab); you may adjust the following in the Localization and Formatting sections:

Language	**English (U.S.)** or **English (International)**
Time Zone	Adjust to your desired time zone
Date Format	Adjust to your desired date format

3 Click **Save**.

EXERCISE 03: Reviewing the Basics of NetSuite Navigation (Optional)

Scenario: Refer to this exercise if you're new to NetSuite or have limited experience with it. You can refer back to this exercise at any time to enhance your productivity.

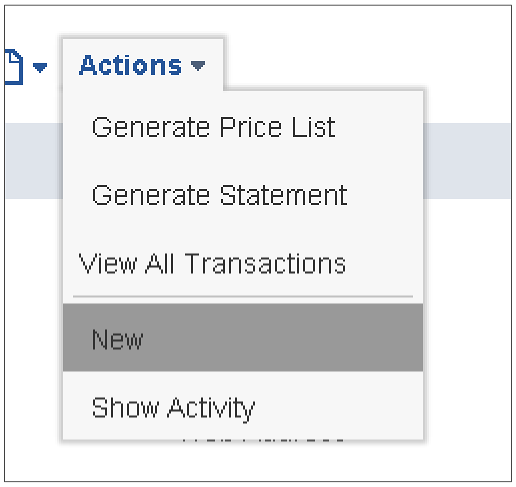
Working with a List of Records

1. On the main NetSuite page, navigate to **Lists > Relationships > Customers**. This displays a list view of customer records.
2. There are various **Filters** at the top of the page such as **Sales Rep**, **Stage**, and **Style**. There is nothing you need to change now, but keep this in mind when viewing other record lists, as you may need to adjust the selectors on them.
3. List views can be sorted by clicking on most column headings. Notice the arrow in the NAME column heading. This indicates records are currently sorted in ascending order by NAME.
4. Click on the Name column heading to refresh the list with the set of customers in descending order of Name.
5. Click the NAME column heading again to change the sort order back to ascending order by NAME.

Viewing and Editing Records

6. Click **View** beside the customer whose name is **ABC Marketing Inc**. Now you are in view-only mode for the customer.
7. Click **Edit** on the customer record to open ABC Marketing Inc**.** in edit mode.
8. Modify the COMMENTS field (right-side under **Primary Information**) by adding something like, "Modified by - <your name>".
9. **Save** the record. You are taken back to view-only mode for the customer.

10 Hover your mouse over the **More Actions** dropdown and choose **New**. This opens a form to create a new customer.



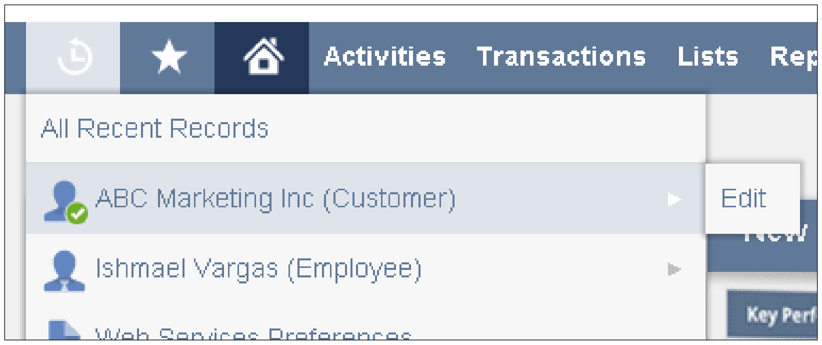
Note: There's no need to create a new customer record at this point.

Note: You can also create a new customer by navigating to **Lists > Relationships > Customers > New**.

Managing Multiple Tabs

11 Open a list of tasks by navigating to **Activities > Scheduling > Tasks**.

12 Re-open the **ABC Marketing Inc**. customer by hovering the mouse over the recent records icon (located on the far left of the tabs, looks like a clock). Choose **Edit** to open the customer in edit mode or choose the customer itself to open in view only mode.



13 Re-open a list of tasks, but in a new tab. This preserves the **ABC Marketing Inc** customer in a separate tab.

14 Navigate to **Activities > Scheduling > Tasks**, but right-click on the **Tasks** menu before selecting it. A context window opens. Select **Open Link in New Tab** to open the list of tasks in a new tab.

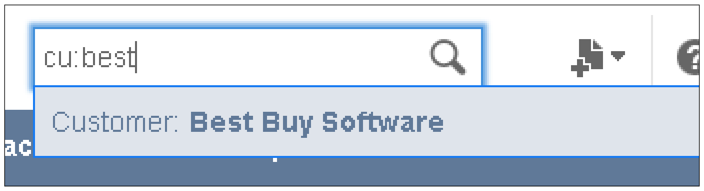
Note: The selection in the context menu may vary slightly across different browsers and/or operating systems.

15 From the list of tasks; right-click to open the task titled Phase 2: Design in a separate tab, in edit mode or view-only mode.

Note: You can open just about any link or menu selection in a new browser tab.

Using Global Search

16 In the **Search** box at the top of the page, type **cu: best**:



17 This immediately performed a search in the system on customers (by using the first two or three letters of the record type as a prefix) whose names contain **best**. From here you can select the record for viewing or editing.

18 You can take the above approach with any type of record in NetSuite. You can also enter without a prefix and the search will be across all records in NetSuite. E.g. **customers** returns a set of customer pages (many of them reports) and searches.

19 Enter **search**(or just “se”)**:customers** in the global **Search** box and the results are filtered to saved searches containing **customers** in the title.

DEVELOPING SUITESCRIPTS

MODULE EXERCISES

Required Exercises	
01	Hello World Script

EXERCISE 01: Hello World Script (Required)

Scenario: SuiteDreams would like to apply customizations beyond what is capable with simple form customizations. Scripting knowledge is necessary to apply advanced customizations, and this exercise is a first step.

To start the script development, a simple “hello world” script will need to be created.

SuiteDreams would like to have their development team be productive in the development of scripting solutions.

Create a SuiteScript File

1 In your preferred IDE, create a file called `sdr_ue_customer.js`.

Build Your Hello World Script

2 Add the following JSDoc annotations at the top of your script:

```
/**
* @NScriptType UserEventScript
* @NApiVersion 2.0
*/
```

CAUTION

Not adding this annotation will prevent the system from recognizing this script. If you attempt to upload the file to the file cabinet, it will be rejected. The file cabinet accepts JavaScript files with proper JSDoc tags.

3 Go to your define function's return statement and add an **afterSubmit** function. Refer to the following syntax:

```
return {
    entryPointName : function (context) {

    }
};
```

4 Create a debug log that will display **hello world**.

5 Login to your NetSuite account on your browser.

6 Go to **Documents > Files > SuiteScripts**

7 Click **Add File** on the upper left corner. Upload the `sdr_ue_customer.js` file.

8 After the file is uploaded, go to **Customization > Scripting > Scripts > New**.

9 Choose the SCRIPT FILE that you've uploaded then click the **Create Script Record** button.

10 Use the following configuration for your script record.

Note: Keep all fields as their defaults unless otherwise specified.

NAME	SuiteDreams UE Customer
ID	_sdr_ue_customer
DESCRIPTION	*<Enter a meaningful description>*

BEST PRACTICES

Make sure to get into the habit of adding meaningful descriptions to any customization you create in NetSuite. Doing this makes it easier to maintain customizations and helps you keep track of the customization and why it needed to be created.

The instructor might skip the description during the demos but this is done only in the interest of time. This should not happen in any customization that will end up in production accounts.

Deployments *(subtab)*	
APPLIES TO	Customer
ID	_sdr_ue_customer
STATUS	Testing
LOG LEVEL	Debug

IMPORTANT

You should ALWAYS populate the ID field wherever you see it. As a developer, it's critical for us to use the well-formed IDs since we'll be referring to these a lot in our code. Problem is, a lot of non-developer users in NetSuite tend to ignore this field since it's not mandatory. When that happens, fields are given a default ID of <prefix>+<number> (e.g., custscript1, custentity5, etc). While you can create a script with that ID, it's very difficult to do so. Imagine developing an application with only single letter variable names. It would be challenging to determine which variable is for which value.

Also make an effort to educate non-developers who has access to customization options that have these IDs.

The format for the script id is _<companyAbbr>_<scriptType>_<description>.

- All IDs will be given a prefix. Starting with an underscore separates the ID from the system generated prefix.
- Adding a company abbreviation to your ID prevents collisions in case you're installing a script from another company through a bundle, or if you're a partner and you're selling your script to your own customers.

11 **Save** your script record.

Test Your Script

12 Go to the list of custom records (**List > Relationships > Customers**).

13 Click the **Edit** link on any record.

14 **Save** the record to trigger your function.

Note: You need not make any changes to the record. If a confirmation prompt appears, click OK.

15 Go back to your script record (**Customization > Scripting > Scripts**; click **View** on the script record).

16 Click the **Execution Log** subtab and check if your "Hello World" message was logged.

IMPORTANT

Visit the Help Center and SuiteAnswers for a full overview of SuiteScript functionality.

USING SUITESCRIPT OBJECTS

MODULE EXERCISES

Required Exercises	
01	Add a Free-Form Text Entity Field
02	Log Data from Customer
03	Debugging Server-side scripts

EXERCISE 01: Add a Free-Form Text Entity Field (Required)

Scenario: SuiteDreams would like to have special discount code processing (called coupons) applied to customer. Create a field that will hold the coupon code value.

Add Coupon Code Field

1 Go to **Customization > Lists, Records, & Fields > Entity Fields > New**.

2 Create a **Coupon Code** entity field with the following configuration:

LABEL	Coupon Code
ID	_sdr_coupon_code
DESCRIPTION	*< Enter a meaningful description>*
TYPE	Free-Form Text

Applies To (subtab)	
CUSTOMER	*<checked>*
Display (subtab)	
SUBTAB	Main

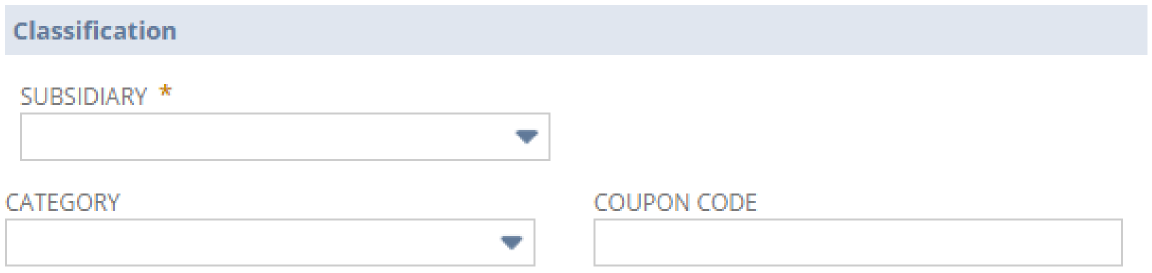
Note: Keep all other fields to their default values.

3 **Save** your changes.

Confirm Field Addition

4 Go to **Lists > Relationships > Customers > New**.

5 Verify that the field was added to the form.



EXERCISE 02: Log Data from Customer (Required)

Scenario: Create an audit trail of customer data for troubleshooting purposes. The data will be used to track the sales related information from the customer, like who sold to the customer and what coupon code value was used, if applicable.

The audit trail should be taken when the record is saved.

Log the Customer Information

1 Go back to your user event script and comment out our hello world log.

2 In your **afterSubmit** function, get the record object from your function's context object. Store this in a **customer** variable.

Note: You can get the record object using the **newRecord** property of your context object.

3 From your customer object, get the values off of the following fields:

- CUSTOMER ID
- Customer EMAIL
- SALES REP Name
- COUPON CODE

Note: Remember to use the right method to extract the values: getValue() or getText().

TIPS AND TRICKS

You can get the field's script id by opening the field level help. This can be done by clicking the field name.

4 Create an audit log, to log the information that you've gathered from the previous step.

5 Upload the script to the File Cabinet.

Test

6 Go to the list of customer records (**List > Relationships > Customers**).

7 **Edit** an existing customer record. Make sure that all the fields you're logging are populated before saving.

Note: You can use any value for the fields.

8 Go back to the script record you've just created.

9 Go to the **Execution Log** subtab and verify if the log was generated correctly.

Note: You can view formatted logs by clicking on the **View** button.

EXERCISE 03: Debugging Server-side scripts (Required)

Scenario: As part of the development process, we'll be looking at how to debug server-side scripts.

Launch the Script Debugger

1 In the NetSuite page o to **Customization > Scripting > Script Debugger**. Look for the, "Click here to log on to the SuiteScript Debugger domain" and use that to open the debugger page.

2 Login to the debugger domain.

Note: You can only login to one web application server at a time. If you're logged in to the debugger domain, you'll automatically get logged out of the production domain and vice versa.

TIPS AND TRICKS

If you already have several pages on the production domain open, just go to the url and replace "system" with "debugger" to reload the page in the debugger domain. Make sure to do this **after** you've logged in to the Script Debugger page.

Debug Your Script

3 Click the **Debug Existing** button to get the list of scripts that you can debug.

IMPORTANT

For a script to appear on the list, the STATUS must be set to **Testing** and the OWNER of the script must be the currently logged in user.

4 Choose your **SuiteDreams UE Customer** script then click the **Select and Close** button. This will cause your debugger to wait for you to trigger your script.

DID YOU KNOW?

The debugging session automatically expires after two minutes of inactivity.

5 In another browser tab, **Edit** any existing customer record and save it to trigger your script.

IMPORTANT

Don't close your debugger window. Make sure to open a new record in another tab/window.

6 Immediately go back to the debugger page and wait for the script to get loaded.

7 Click on the **Step Over** button to execute your code line by line. Stop at about the third line into the function.

8 Click the **Local Variables** subtab. Notice that the all variables that are declared in the instance is listed here.

9 Add a breakpoint at your log.audit() call by clicking on the space to the right of the line number.

10 Click the **Continue** (play icon) button to continue with the execution.

11 Go to the **Break Points** subtab and remove the breakpoint but clicking the x link.

Note: You can also remove the breakpoint by clicking on the breakpoint icon beside the line number.

12 Click on the **Continue** button again to complete the execution.

13 Go to the **Execution Log** subtab. The exercise is complete if you see the logs that you've generated.

UNDERSTANDING ENTRY POINTS

MODULE EXERCISES

Required Exercises	
01	Add a Checkbox Entity Field
02	Enable and Disable Coupon Code
03	Debugging Client-side SuiteScript 2.0 Scripts
03	Validate Coupon Code When Submitting Form
Optional Exercises	
04	Validate Coupon Code When Changing It

EXERCISE 01: Add a Checkbox Entity Field (Required)

Scenario: To automate the customer's discounts, SuiteDreams would like to add a checkbox that controls the coupon code field.

Add Checkbox Field

1 Go to **Customization > Lists, Records, & Fields > Entity Fields > New**.

2 Create a checkbox using the following configuration:

Note: Keep all fields to their defaults unless otherwise specified.

LABEL	Apply Coupon
ID	_sdr_apply_coupon
DESCRIPTION	*< Enter a meaningful description>*
TYPE	Check Box

Applies To (subtab)	
CUSTOMER	*<checked>*
Display (subtab)	
SUBTAB	Main

3 **Save** your entity field.

Modify Coupon Code Configuration

4 Go to the **Custom Entity Fields** page (**Customization > Lists, Records, & Fields > Entity Fields**).

5 Click on the **Coupon Code** link to change the configuration.

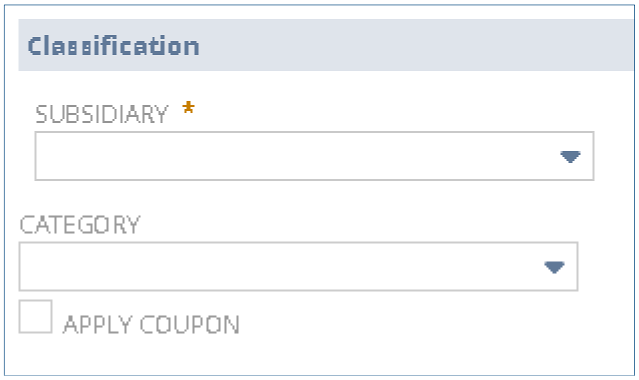
6 In the Display subtab, change the DISPLAY TYPE to **Disabled**.

Note: This change prepares the field for the next exercise.

Verify Field Changes

7 Go to **Lists > Relationships > Customers > New**.

The exercise is complete if you see the APPLY COUPON checkbox on the body section (above the subtabs) of your customer form.



EXERCISE 02: Enable and Disable Coupon Code (Required)

Scenario: SuiteDreams wants to the customer discount process automated. Users expect the following behavior for the discount fields:

Upon changing Apply Coupon:

- If APPLY COUPON is checked, enable COUPON CODE.
- If APPLY COUPON is unchecked, disable COUPON CODE and erase its contents.

Create a Client Side Script

1 What entry point should contain this script?

Answer:

2 What is the script id of the field you'll be testing against?

Note: Remember that you need to check for the field that the user will be changing, not the field that you wish to set.

Answer:

3 Go to your IDE and create a new SuiteScript File.

4 Create a **Client Script** and name it **sdr_cs_customer.js**.

5 Add the annotation for the correct SuiteScript type.

```
/**
 * @NScriptType ClientScript
 * @NApiVersion 2.0
 */
```

Add the Discount Automation

6 Create a **fieldChanged** function.

7 Add an **if** statement to check filter the execution of the script to a specific field.

IMPORTANT

The **fieldChanged** entry point will trigger regardless of the field that the user modifies. It's very important to filter the execution so that it only executes when the user modifies the target field.

ORACLE
NETSUITE

8 Get a copy of the customer record from the context object.

Note: You can get the record object using the **currentRecord** property of the context object.

This is the same object as the one from the newRecord property. The difference is that the currentRecord is used in client side script and newRecord for user event (server side) scripts.

9 From the customer object, get the field reference that refers to the coupon code.

Note: The field object reference can be extracted using the **getField()** method of the record object.

IMPORTANT

The field reference object is different from the field value. The reference is used to manipulate the properties of the field such as enabling/disabling it or making it optional/mandatory.

10 Add the statements to support the following pseudocode:

- If the Apply Coupon is checked, enabled the Coupon Code for data entry.
- If the Apply Coupon is unchecked, disable the Coupon Code field and erase the contents.

Note: Fields can be disabled using the field object.

11 Go to the return statement and comment out all the functions that you have not used.

Create a Script Record

12 Upload the script to the File Cabinet.

13 Create a script record with the following configuration:

NAME	SuiteDreams CS Customer
ID	_sdr_cs_customer
DESCRIPTION	*<Enter a meaningful description>*

Deployments *(subtab)*	
APPLIES TO	Customer
ID	_sdr_cs_customer
STATUS	Testing
LOG LEVEL	Debug

14 **Save** the script record.

Test

15 Create a new customer record (**List > Relationships > Customer > New**).

Note: You need not save the record to test.

16 Test by checking and unchecking the Apply Coupon field. Make sure that the field behaves as expected.

BEST PRACTICES

Be cognizant that the script is triggered only when the user clicks on the APPLY COUPON field. Loading the record will still keep the coupon code field disabled since it was set to always be disabled. To make sure that it's enabled when there's a coupon code value, a similar automation must be added in the pageInit function. This is a common practice with these kinds of automation.

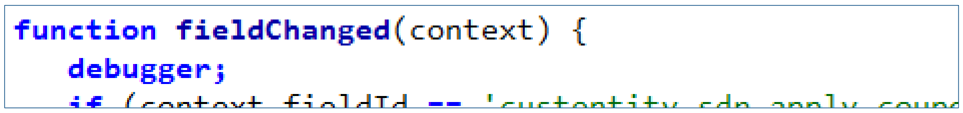
EXERCISE 03: Debugging Client-side SuiteScript 2.0 Scripts (Required)

Scenario: Debugging client-side scripts is an important skill to learn for building SuiteScripts.

Using the Debugger Statement

1 Open the script that you've previously created.

2 Inside your fieldChanged function, add “debugger;” at the start of the function.



```
function fieldChanged(context) {
   debugger;
   if (context.fieldId == 'custentity_sdr_apply_coupo
```

> DID YOU KNOW?
>
> Adding a debugger statement to your client-side script is similar to adding a breakpoint. This stops the execution of the script at the point where the debugger statement is at.

3 Upload your changes to the File Cabinet.

4 Edit an existing customer and do a force refresh.

> DID YOU KNOW?
>
> Client side scripts sometimes don’t get downloaded because the browser prefers getting the script from cache. To force the browser to fetch the information from the server, press <CTRL>+<F5> (Windows) or <Command>+<Shift>+<R> (Mac).

5 Open your browser's debugger/developer tool.

> IMPORTANT
>
> Client side scripts are debugged locally using your browser's debugger/developer tool. The common keyboard shortcut for opening the debugger is <F12> (on Windows) but it would still depend on your browser.

6 Trigger your function by checking or unchecking the APPLY COUPON field

Note: The debugger should stop the execution of your script at the point where you've added the debugger statement.

7 Notice the filename displayed for the script. Depending on your browser, you might see that the script's filename is labeled as VM### (eg. VM1064).

DID YOU KNOW?

SuiteScript 2.0 entry point functions are rendered during execution. This is why the debugger is not linking the function to a particular JavaScript file.

8 Continue the execution of your script by stepping through your code until it finishes.

EXERCISE 04: Validate Coupon Code When Submitting Form (Required)

Scenario: Before submitting coupon code values to the server, it must be validated. Here is the validation criteria:

- You must enter a COUPON CODE of 5 characters in length when APPLY COUPON is checked. The validation is performed at the time of submitting the form.

Note: You can use the `length` property to get the number of characters in a string.

IMPORTANT

Remember to use the return statement to give a boolean value that would either allow or prevent a user from saving this record.

Validate Coupon Code Upon Save

1 What entry point should contain this script?

Answer:

2 Create script that is modeled after the following pseudocode:

If APPLY COUPON is checked and length of COUPON CODE is not 5, display an alert message to the end user regarding this length restriction. Do not submit the form.

3 Submit the form if the above test passes.

Note: You can edit an existing customer record to test instead of creating a new customer record.

BEST PRACTICES

On saveRecord, validateField, and any entry points that return boolean values; it's recommended that you put a `return true` statement at the end of your function. This makes sure that your function will always end properly,

EXERCISE 05: Validate Coupon Code When Changing It (Optional)

Scenario: End users decide they would rather have the coupon code validation occur immediately upon changing the field. The validation is the same as before:

- You must enter a COUPON CODE of 5 characters in length when APPLY COUPON is checked.

Note: This is practically the same as the previous exercise but triggered from a different entry point.

BEST PRACTICES

The validateField entry point is used if for validating individual fields. If used on multiple fields, it can potentially be annoying for your users. A solution for this is to validate multiple fields at the same time using the saveRrecord entry point.

Create a Field Level Validation for Coupon Code

1. Create script that is modeled after the following pseudocode:

   If APPLY COUPON is checked and length of COUPON CODE is not 5, display an alert message to the end user regarding this length restriction. Keep the user from clicking buttons or editing other fields on the form.

2. Let the user continue editing if the above test passes.

   Note: Skip this validation when Apply Coupon is unchecked, otherwise you will get this validation message as you are disabling Coupon Code and erasing its contents.

3. Test.

   Note: This exercise is intended for you to try using the validateField event. You would not normally create the same validation for validateField and saveRecord events.

ORACLE NETSUITE

EXERCISE SOLUTIONS

EXERCISE 02: Enable and Disable Coupon Code

What entry point should contain this script?

Answer: Field Changed, similar to the previous exercise.

What is the fieldId of the field you'll be testing against?

Answer: The APPLY COUPON field, custentity_sdr_apply_coupon.

EXERCISE 03: Validate Coupon Code When Submitting Form

What entry point should contain this script?

Answer: Save Record.

SUITESCRIPT MODULES

MODULE EXERCISES

Required Exercises	
01	Create Sales Rep Task
02	Create Custom Task Form
Optional Exercises	
03	Send Email to Customer
04	Investigating the Promise API

EXERCISE 01: Create Sales Rep Task (Required)

Scenario: SuiteDreams likes to provide the best customer service in the industry. SuiteDreams would like to find an automated way to remind sales reps to follow up with new customers. This can be done through the automatic creation of task records.

The following are the business requirements for the task record:

- Set the TITLE to "New Customer Follow-up".
- Add a **Message** to say "Please take care of this customer and follow-up with them soon."
- Set PRIORITY to **High**.
- Set the COMPANY field to the Customer the user is editing.
- If SALES REP on the **Customer** record is not empty, set the ASSIGNED TO field to the SALES REP from the Customer record.

Note: Tasks are created in the user interface at **Activities > Scheduling > Tasks > New**.

Modify the Script

1 Go to your customer user event script and load the record (N/record) module.

Note: Please refer to Appendix A for the syntax for loading modules.

2 In the afterSubmit function, add a condition so that the task record is created only when new customer records are created. Editing existing customers should not trigger this section of the script.

Note: Test this initially to see if it works. If you're confident that it does, feel free to comment it out so you can test the rest of the script with existing customer. Doing so may speed up testing for this exercise.

DID YOU KNOW?

Similar to client side scripts, you can also get the access type from the context object. The type property value holds this parameter.

The type property holds an enum value. This value is also in the context object in the UserEventType property. So if you want to check if the user is editing a record, you can add:

```
if (context.type == context.UserEventType.EDIT) {
   // Do something here
}
```

3 Create your task record using the record module's create method.

Note: Task records can be created in the UI by going to **Activities > Scheduling > Tasks > New**.

DID YOU KNOW?

The record type for your create method is also an enum value. This time the enum value is coming from the record module's Type property.

Note: Make sure to use a capital T for the Type enum.

4 Set the task's TITLE to "New Customer Follow-up".

5 Add a MESSAGE to say, "Please take care of this customer and follow-up with them soon."

Note: The script id for the MESSAGE field is **message**.

6 Set the PRIORITY to **High**.

ORACLE
NETSUITE

DID YOU KNOW?

List/Record fields are usually set by using the internal id of the value you need to set. For the PRIORITY field, you can use the actual value you want to set. The only requirement is that you use all capital letters for the value. For example to set the PRIORITY to low, use customer.setValue('priority', 'LOW');

Note: An error message will be returned if you use the wrong value or case for the PRIORITY.

7 Assign the Customer to the COMPANY field of the task.

Note: For any List/Record fields, use the internal id to set the value. You can get this from the record's id property.

8 Assign the task to the SALES REP on the customer record but only if the Sales Rep field is not empty.

9 Finalize the record by saving it to the database.

Test Your Script

10 Test your script by creating a new customer record.

Note: You can use an existing record if you've already tested this part previously.

11 Go to the list of task record and verify if a new one is created.

Note: Make sure to check the FILTERS and make sure that all filters are set to **All**.

12 Check the field values. The exercise is complete if all the fields are properly set.

EXERCISE 02: Create Custom Task Form (Required)

Scenario: An Administrator at SuiteDreams decides to create a new task form to be used for creating tasks generated from sales orders. The new form is to be used as the preferred form and it requires that a Contact be selected. Contacts that are selectable in the user interface are those tied to the selected Company:

Note: The script that creates a task record doesn't require a Contact. We need to make sure the script continues to run without setting the Contact field.

Customize the Task Form

1. Go to **Customization > Forms > Entry Forms**, and **Customize** the **Standard Task Form**.
2. NAME the new form **SuiteDreams Task Form** and give it the id **_sdr_task**.

BEST PRACTICES

Whenever you're creating a form, make sure that you name it based on the group who will be using the form. In this case, it's named Sales Customer Form because it will be used by the Sales team. If this will be used by the whole company, for example, you can name it SuiteDreams customer form.

3. Enable the FORM IS PREFERRED option.

   Note: Enabling this forces task records to use the form when editing an existing or creating a new record.

4. Click the **Fields** subtab, then the **Related Records** subtab.
5. Look for the **Contact** field and make it MANDATORY.
6. **Save** the custom form.

Add a Task Through the User Interface

7. Open a new task form. Confirm that the **SuiteDreams Task Form** is the selected form in the CUSTOM FORM field.
8. Give the task a TITLE then immediately Save. What happens and why?

   Answer:

Add a Task Through Script

9 Re-execute the script from the previous exercise.

Note: Do not make any changes to the script.

What happens and why?

Answer:

Use the Standard Task Form in the Script

10 Find the internal id of the **Standard Task Form**.

Internal ID:

11 In your `create` method call, add a **defaultValues** property.

12 Create a payload object for the defaultValues property with the property customForm and the internal id of the **Standard Task Form**.

13 In your defaultValues object, add the **customform** property and set it to the internal id of your Standard Task Form.

IMPORTANT

Take note that the property name is all lowercase. Using the wrong case would cause the system to not recognize the property.

14 Upload your changes to the File Cabinet.

15 Test your script by creating a new customer record. Your script is complete if the script ran without errors and if the task record was created.

DID YOU KNOW?

Standard forms in NetSuite cannot be modified. Customizing a standard form create a new copy under a different name, keeping the original standard form intact. Because of this, using standard forms is perfect for making sure that your script will work properly without worrying about form modifications.

EXERCISE 03: Send Email to Customer (Optional)

Scenario: New customers of SuiteDreams should get welcome emails sent to them. SuiteDreams would also like to have a copy of the email attached to the customer record. Someone viewing the customer record should be able to see that the email has been attached.

Email the New Customer

1 Add the email and runtime modules in your customer user event script.

2 Get a reference to the currently logged in user using the runtime module's getCurrentUser method.

Note: You will use this user as the sender of the email.

3 Using the email module, send an email with the following configuration:

author	*<Internal ID of the currently logged in user>*
recipients	*<Internal ID of the customer record being created>*
subject	Welcome to SuiteDreams
body	Welcome! We are glad for you to be a customer of SuiteDreams.

Test

4 Test your script by creating a new customer record.

IMPORTANT

Make sure that you populate the EMAIL field on the customer record you've created is populated. You will get the error SSS_INVALID_TO_EMAIL or an unexpected error if the field is empty. This is because the system would not know where to forward the email that you're sending.

Also, use a long fictitious email when creating your record. This is to prevent accidentally sending email messages to real email addresses. For example, do NOT send to test@test.com as that is a real email address and the owners of the domain are not happy to receive hundreds of spam daily. What you can do is to use your account number (**Setup > Integration > Web Service Preferences**) as your domain name. This should look something like "test@TSTDRV1234567.com"

5 View the customer record that you've just created, if it isn't already open.

6 Go to the **Communications** subtab. Your exercise if complete if you see the email that your script has sent.

Note: Several of the messages in the system are future dated. Make sure to go through the different pages to look for the email the script sent.

EXERCISE 04: Investigating the Promise API (Optional)

Scenario: Before using the Promise API, we'll be investigating how it performs when processing multiple server calls from the client-side. The exercise will have you compare the performance difference between promise and non-promise calls. Multiple sales order records will be loaded on pageInit and you will be inspecting the performance using your browser's developer tools.

Create the Function Calls

1 Create a client side script with the record module loaded.

Note: This script will be deployed to a sales order record. Also, take note that records can be renamed in NetSuite. In your training account, the Sales Order record has been renamed to Order.

2 Create two functions, one for loading records using the promise API and the other for non-promise calls. Have both functions accept a parameter for the internal id of a record.

Note: Make sure to put the functions outside of the returned entry point functions. You will be calling these inside a pageInit Script

3 Go to your non-promise function and load a sales order record using the internal id passed to the function.

4 After loading the sales order, get the TOTAL amount value and log the information to the browser console.

TIPS AND TRICKS

Aside from the alert statement, another way of displaying information on the client-side is by using the **console.log()** function.

For this to work, you need to open the browser's developer tools and have the console open.

ORACLE NETSUITE

5 Go to your promise function and do the same thing as your non-promise function except use the promise version of **record.load()**.

```
function promiseCall(id) {
   record.load.promise({
       // Set the required properties
   }).then(
       // The salesOrder value is returned by record.load.promise
       function(salesOrder){
           // Get the total field and log that in the console
       }
   );
}
```

Prepare the Script for Testing

6 Create a **pageInit** function and call your non-promise function about 15-20 times, loading different records on each call.

7 Add the same amount of calls to your promise functions loading the same records. Comment out the promise API calls for now.

8 Create a script record and deploy it to the sales order record.

Test

9 Go to a new sales order form (**Transaction > Sales > Enter Orders**).

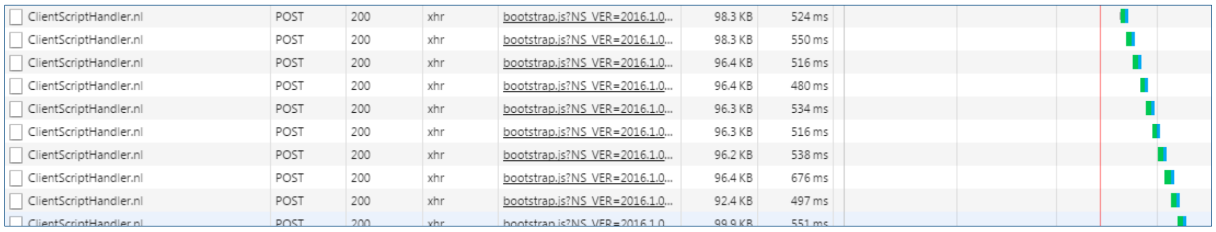
10 Open your browser's developer tools and go to the **Network** tab.

Note: Monitoring network traffic in Chrome is done on the Network tab. If you're using a different browser, the tab might be labeled differently.

11 Click the **XHR** button so the Network will only log XMHttpRequests. This allows you to monitor the performance of calls to the server.

12 Refresh the sales order form and monitor the performance of the requests sent by your non-promise function.

Your network call should look something like this:



ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	98.3 KB	524 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	98.3 KB	550 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.4 KB	516 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.4 KB	480 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.3 KB	534 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.3 KB	516 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.2 KB	538 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.4 KB	676 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	92.4 KB	497 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0	99.9 KB	551 ms

Note: Looking at this kind of request, you can see the requests are called one after the other. Multiple requests like this will take a long time to do.

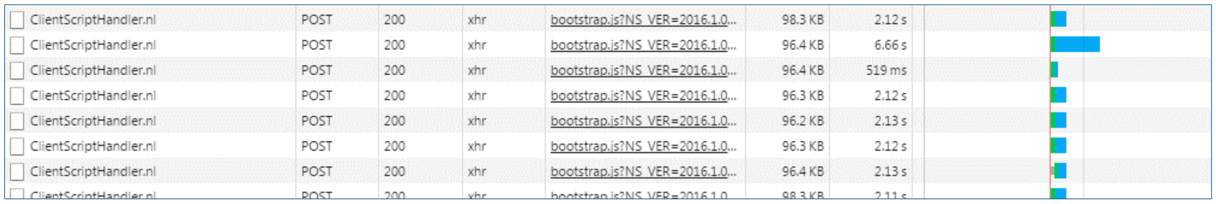
13 Take note of the total amount of time it took for the requests to complete.

Time spent on non-promise calls:

14 Comment out the non-promise calls and uncomment the promise calls.

15 Refresh the page again and notice how the network logs have changed.

The network calls should have changed to this:



ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	98.3 KB	2.12 s
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.4 KB	6.66 s
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.4 KB	519 ms
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.3 KB	2.12 s
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.2 KB	2.13 s
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.3 KB	2.12 s
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0...	96.4 KB	2.13 s
ClientScriptHandler.nl	POST	200	xhr	bootstrap.js?NS_VER=2016.1.0	98.3 KB	2.11 s

Note: Comparing this to the previous request, each promise call runs independently from the main thread. This allows the script to make multiple server calls without waiting from the previous server call to finish; making this a more efficient approach.

16 Take note of the total amount of time it took for the promise requests to complete:

Time spent on promise calls:

BEST PRACTICES

Promise calls are extremely effective when used properly. Make sure to think about processing multiple threads at the same to get the most out of promises.

EXERCISE SOLUTIONS

EXERCISE 02: Create Custom Task Form

8 Give the task a TITLE then immediately Save. What happens and why?

Answer: The record was not saved because the CONTACT field needs to be populated. This is because the preferred form has the field set as mandatory.

9 Re-execute the script from the previous exercise. What happens and why?

Answer: The script is not working anymore. It's asking for the contact field to be populated similar to the NetSuite UI. This is because SuiteScript uses the preferred form when creating and modifying record objects.

SCRIPTING SUBLISTS

MODULE EXERCISES

Required Exercises	
01	Create Product Preferences Record Type
02	Script Product Preferences Record Type
03	Schedule Welcome Conversation with Sales Rep

EXERCISE 01: Create Product Preferences Record Type (Required)

Scenario: SuiteDreams is creating a custom record type to store information about products that customers often order. This is used by SuiteDreams to automatically generate sales orders, as well as determine what items might need to be replenished from inventory.

End users do not have access to custom record types directly, but should be able to access product preferences as a sublist off of the customer record.

A product preferences record type contains these fields:

NAME	Associates a name with each product preference, such as Preference 1, Preference 2, Preference 3, etc
CUSTOMER	Identifies the related customer
ITEM	Identifies the item that is being preferred
PREFERRED QUANTITY	Identifies the quantity of the preferred item; this is the quantity of the item the customer usually places on a sales order

Create Record Type

1 Go to **Customization > Lists, Records, & Fields > Record Types > New**.

2 Create your new custom record with the following configuration. Unless otherwise specified, keep all fields as their defaults.

NAME	Product Preferences
ID	_sdr_prod_pref
DESCRIPTION	*<Enter a meaningful description>*
SHOW ID	*<checked>*
ALLOW CHILD RECORD EDITING	*<checked>*
ALLOW DELETE	*<checked>*

3 **Save** the custom record type.

Note: You need to initially save the custom record before you can add custom fields.

4 Click **New Field** button from the **Fields** subtab to create fields for the Product Preferences record type.

TIPS AND TRICKS

You can hover over **Save** then click **Save & New** to quickly create another field.

LABEL	Customer
ID	_sdr_prod_pref_customer
TYPE	List/Record
LIST/RECORD	Customer
SHOW IN LIST	*<checked>*

LABEL	Item
ID	_sdr_prod_pref_item
TYPE	List/Record
LIST/RECORD	Item
SHOW IN LIST	*<checked>*

LABEL	Preferred Quantity
ID	_sdr_prod_pref_qty
TYPE	Integer Number
SHOW IN LIST	*<checked>*

Attach the Custom Record as a Sublist to the Customer Record

5 Edit the **Customer** field again by clicking on the link.

6 Enable to RECORD IS PARENT field.

Note: This attaches the Product Preferences (child) custom record as a sublist to the Customer (parent) record.

7 Go to the **Display** subtab and choose Sales in the PARENT SUBTAB field.

8 **Save** your changes.

Add Product Preferences

9 In the **Custom Record Type** page, hover over the **More** link (located at the upper right-hand corner of the page) then click on the **View Records** link to view, edit, and create new product preferences.

10 Create a minimum of two product preferences for each of customers. When selecting items, select inventory items.

11 To find inventory items, go to **Lists > Accounting > Items**, making sure the TYPE filter at the top of the page is set to **Inventory Item** and the VIEW filter is set to **All**.

12 Use items that have both a PURCHASE PRICE and an amount listed for AVAILABLE.

Note: You are going to need to scroll to the right to see these field columns. Future exercises are going to evaluate the PURCHASE PRICE and AVAILABLE quantity.

CAUTION

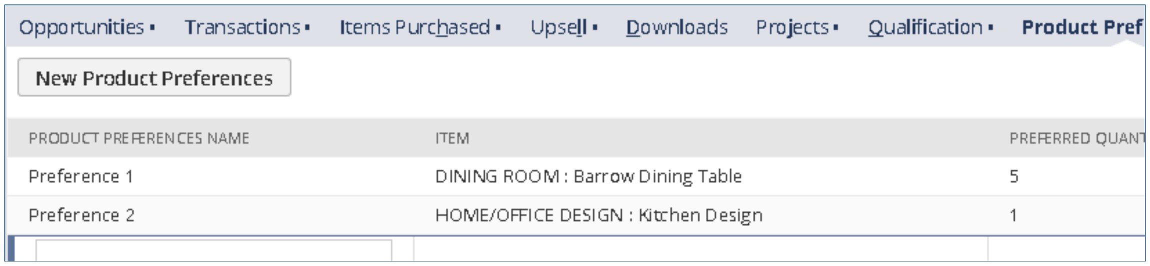
When selecting items, make sure that you're selecting item records instead of the parent items.

Items in NetSuite can be grouped together through a parent-child relationship. For example you might see entries like, BEDROOM : Yao Bed. The value before the colon (BEDROOM) is the parent item and the actual item is after the colon (Yao Bed).

Parent items are not real item and are there only to represent the grouping. Using parent items for the exercise can cause problems as they don't have the values that you'd find on an actual item record.

13 Edit one of the customers for which you created a product preference (**Lists > Relationships > Customers**). You should see a **Product Preferences** subtab display as a child subtab to the **Sales** subtab.

14 Here is a sample of what you might see when you click on the Product Preferences subtab of the Sales subtab on a customer record. You should be able to click into each field in the sublist and edit the values in place.



PRODUCT PREFERENCES NAME	ITEM	PREFERRED QUANT
Preference 1	DINING ROOM : Barrow Dining Table	5
Preference 2	HOME/OFFICE DESIGN : Kitchen Design	1

EXERCISE 02: Script Product Preferences Record Type (Required)

Scenario: Implement the following business requirements on your customer record:

- Alert the end user to the number of product preferences upon opening a customer record for editing.
- Default the QUANTITY to 1 when entering a new product preference.
- Apply the following validation to the addition and modification of product preferences on the customer record:
  - PREFERRED QUANTITY of a product preference cannot be more than 10.
- Apply the following validation at the time of form submittal:
  - The total PREFERRED QUANTITY across all product preferences for an individual customer cannot exceed 25.

Alert the End User to the Number of Product Preferences

1. Alert the end user to the number of product preferences upon opening a customer record for editing.

   What is the correct client entry point for implementing this script?

   Answer:

2. Determine the sublist id. For custom record types, it is recmach + the id of the list/record field you used to link this record as a sublist. What is the sublist id?

   Answer:

3. Go to your client-side script deployed on the customer record and add the statement to get the number of sublist lines.

4. Display the following alert message: **This customer has <n> product preferences**, where n = the number of line items

5. Test this alert message before moving onto the next section.

DID YOU KNOW?

If you get a -1 value on your getLineCount, that means that the ID you used is incorrect. If a sublist is empty, you'll get the value 0.

Default Preferred Quantity to 1

6 Default the PREFERRED QUANTITY to 1 when entering a new product preference (i.e. a new line item, one where the preferred quantity is empty), otherwise do nothing.

CAUTION

Make sure to wrap your **getCurrentSublistValue()** statement with a **parseInt()** or **parseFloat()**. When the value is empty, the functions will return a NaN value. You could use the isNaN() function to do value check.

Here's an example on how it would look like:

```
        var qty = parseInt(value);
        if (isNaN(value)) {
           // do something
  }
```

7 Use the lineInit entry point for this section of the exercise.

8 Test this line initialization script before moving onto the next section.

Note: The lineInit entry point gets trigged when the user clicks any of the sublist buttons. When you initially load the record, the sublist value will not default to 1 unless the user clicks on a sublist button. Make sure to remember this behavior and test by clicking on the sublist button instead of just refreshing the page.

Validate the Preferred Quantity

9 Preferred quantity of a product preference cannot be more than the 10. Validation should occur as a line is being inserted or edited.

What is the correct client event function for implementing this script?

Answer:

10 Get the PREFERRED QUANTITY off of the current line item.

11 Alert the end user with the following message when the PREFERRED QUANTITY is greater than 10: **You have selected a preferred quantity that exceeds the limit of 10.**

Note: Like the validateField and saveRecord entry points, you must return a boolean value on this function.

12 Test this validation script before moving onto the next section.

TIPS AND TRICKS

If you want to validate a sublist field without moving saving the sublist line, you can use the validateField entry point instead.

Validate the Sum of Preferred Quantity Across All Preferences

13 The sum of PREFERRED QUANTITY across all product preferences for a customer cannot be more than 25. Validation should occur at the time of form submittal.

What is the correct client event function for implementing this script?

Answer:

14 Loop through all line items, adding up the PREFERRED QUANTITY. Use a for loop or any other looping constructs you're comfortable with.

Note: Please see Appendix A for the for loop syntax.

15 Return the following message when the total preferred quantity exceeds the limit of 25: **The total preferred quantity across all product preferences has exceeded the limit of 25.**

IMPORTANT

Remember that sublist line numbers start with 0.

16 Test this final sublist script.

EXERCISE 03: Schedule Welcome Conversation with Sales Rep (Required)

Scenario: Aside from the task record for the sales rep, the script will also be creating a meeting with new customers. If no sales rep is assigned to a customer, the user should be prevented from saving the record.

Validate Sales Rep Value

1. Go back to the user event script for the customer record.
2. Add a beforeSubmit entry point function to the script and get the customer object from the context.

   Note: Validations don't work on afterSubmit and must be executed on a beforeSubmit.

3. Filter the execution so that it only runs when a new customer record is created.

   Note: Comment the code for now to make testing faster but test it later once the script has been completed.

DID YOU KNOW?

The context object has a **type** property that indicates the user's action. It also has the **UserEventType** enum that you can compare to.

4. Get the SALES REP value from the customer record.
5. Add a condition so that if the SALES REP field is empty, it would throw an error message.

TIPS AND TRICKS

Passing the actual variable without adding a condition can effectively be used in JavaScript to check if a value is empty. These are called truthy/falsey checks.

For more information about this, please refer to Appendix A.

6. To display an error, use the throw statement to display a string error message. This could be as simple as, "`throw 'Save failed. Please make sure that the Sales Rep field is not empty.';`"

Note: SuiteScript supports a more comprehensive error handling including a native error object.

7 Perform an initial test to make sure that the validation works.

Setup Meeting with Customer

8 Go to the afterSubmit function of your customer user event script.

9 Create an event record with the following configuration

Title	Welcome conversation with <customer name>
Notify Attendees By Email	<yes>
Company	<customer>
Required Attendees	<customer> <assigned sales rep>

Note: Use the SuiteScript Records Browser to get the ids for the event record.

The COMPANY field is under the **Related Records** subtab of the event form.

IMPORTANT

When creating the record object, make sure to set the **isDynamic** property to true. This is necessary since we're dynamically accessing sublist values.

Test

10 Test the script by creating a new customer record with the sales rep assigned.

11 Go to the list of events, Activities > Scheduling > Events, and verify that the event was created.

Note: Make sure that all events are listed by setting the FILTERS values are set to **All**.

DID YOU KNOW?

The currently logged in user is automatically set as the ORGANIZER of the event and will be added to the event.

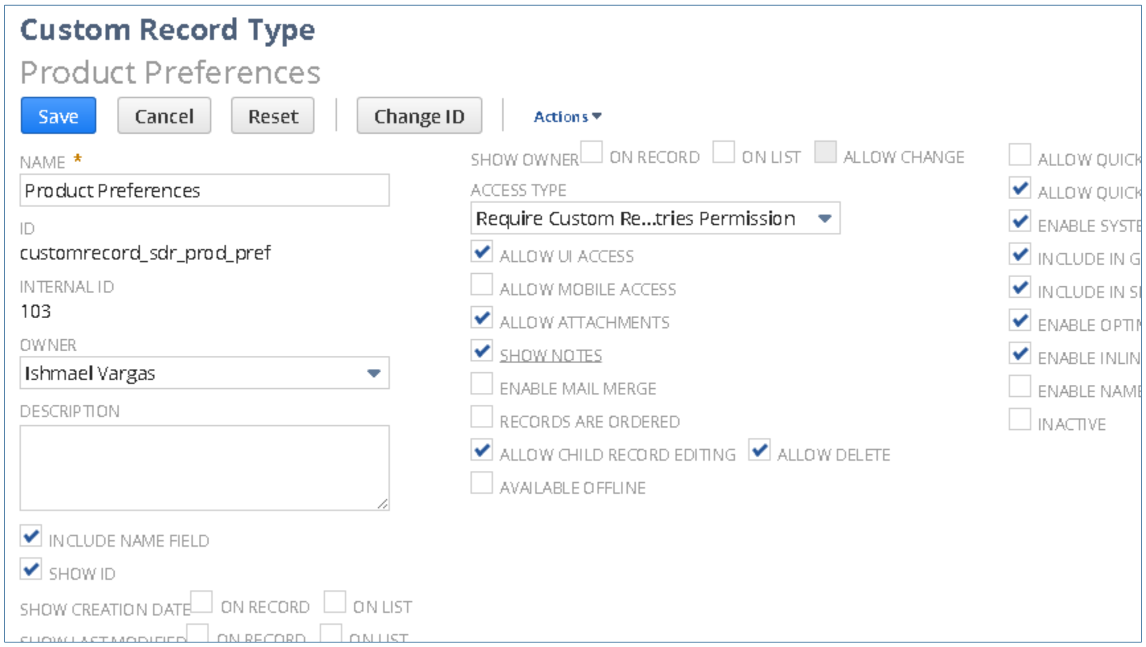
CAUTION

When selecting a customer for testing, make sure to use a customer's SALES REP is not the currently logged in user. Doing so will cause the script to throw an error since the user is already in the event as an organizer.

EXERCISE SOLUTIONS

EXERCISE 01: Create Product Preferences Record Type

Definition of Product Preferences record type



Custom Record Type

Product Preferences

NAME * Product Preferences

ID customrecord_sdr_prod_pref

INTERNAL ID 103

OWNER Ishmael Vargas

DESCRIPTION

☑ INCLUDE NAME FIELD

☑ SHOW ID

SHOW CREATION DATE ☐ ON RECORD ☐ ON LIST

SHOW LAST MODIFIED ☐ ON RECORD ☐ ON LIST

SHOW OWNER ☐ ON RECORD ☐ ON LIST ☐ ALLOW CHANGE

ACCESS TYPE Require Custom Re...tries Permission

☑ ALLOW UI ACCESS

☐ ALLOW MOBILE ACCESS

☑ ALLOW ATTACHMENTS

☑ SHOW NOTES

☐ ENABLE MAIL MERGE

☐ RECORDS ARE ORDERED

☑ ALLOW CHILD RECORD EDITING ☑ ALLOW DELETE

☐ AVAILABLE OFFLINE

☐ ALLOW QUICK

☑ ALLOW QUICK

☑ ENABLE SYSTE

☑ INCLUDE IN G

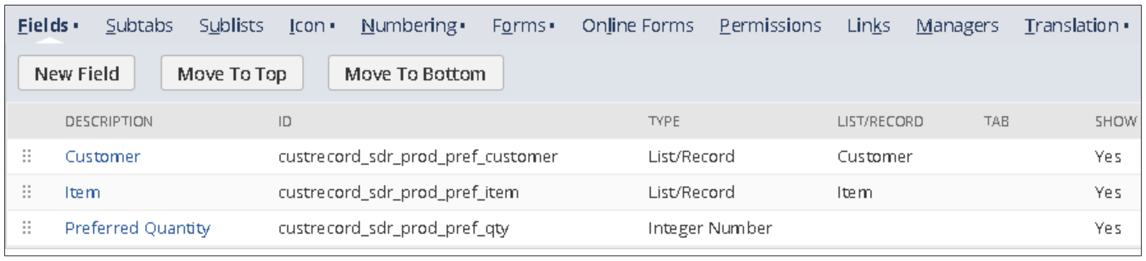
☑ INCLUDE IN S

☑ ENABLE OPTI

☑ ENABLE INLIN

☐ ENABLE NAME

☐ INACTIVE



DESCRIPTION	ID	TYPE	LIST/RECORD	TAB	SHOW
Customer	custrecord_sdr_prod_pref_customer	List/Record	Customer		Yes
Item	custrecord_sdr_prod_pref_item	List/Record	Item		Yes
Preferred Quantity	custrecord_sdr_prod_pref_qty	Integer Number			Yes

EXERCISE 02: Script Product Preferences Record Type

1 What is the correct client entry point for implementing this script?

Answer: Page Init since you want to alert the user upon loading the page.

2 Determine the sublist id. For custom record types, it is recmach + the id of the list/record field you used to link this record as a sublist. What is the sublist id?

Answer: recmachcustrecord_sdr_prod_pref_customer.

9 What is the correct client event function for implementing this script?

Answer: Validate Line.

13 What is the correct client event function for implementing this script?

Answer: Save Record since you need to validate upon saving the record.

SEARCHING IN NETSUITE

MODULE EXERCISES

Required Exercises	
01	Create a Saved Search
02	Execute Saved Search through Scripting
03	Execute Custom Search through Scripting
04	Log Script Search Results

EXERCISE 01: Create a Saved Search (Required)

Scenario: SuiteDreams would like to determine when there may be shortages for products preferred by their customers. Any product preference where the Preferred Quantity is greater than or equal to 2 is indicative of a potential shortage if these product preferences become sales orders.

SuiteDreams is only interested right now in determining product shortages where Customer Subsidiary is in HEADQUARTERS : AMERICAS : US – West.

Prepare Product Preference Data

1 Add or modify product preference records so that a search will return results when based on the filters described in the Scenario section.

Note: You can find customers in the required subsidiary by going to **Lists > Relationships > Customers**.

Select Search Type for Saved Search

2 Go to **Lists > Search > Saved Searches > New**.

3 Select the appropriate Search Type or record on the **New Saved Search** page.

For which record type are we selecting a list of records?

Answer:

Fill Out Main Area of Search Form

4 Use the following configuration for your search:

SEARCH TITLE	Product Shortages
ID	_sdr_prod_shortages

Note: Leave other fields in the main area of the form as their defaults.

Configure the Criteria

5 Click on the **Criteria** subtab.

6 In the Criteria's **Standard** subtab, configure your filters based on the following:

FILTER	DESCRIPTION
Preferred Quantity	is greater than 2
Customer's Subsidiary	is HEADQUARTERS : AMERICAS : US - West

DID YOU KNOW?

Fields of your target record (Product Preferences in this case) will be displayed on top of the fields dropdown. Related records, or joins, will be displayed at the bottom of the list and will have an ellipsis or three dots, after the field entry (ie. Customer Fields…, Item Fields…, etc.)

Configure the Results

7 Click on the **Results** subtab.

8 In the Result's **Columns** subtab, **Remove All** the listed fields.

9 In the **Columns** subtab, return the following fields in the search results.

FIELD
Customer
Customer's Email
Customer's Subsidiary
Item

FIELD
Preferred Quantity
Available (from Item record)

Test Your Search

10 Hover over **Save** and click **Save & Run**.

BEST PRACTICES

You can also view the results using the Preview button but remember that the saved wouldn't be saved. It would be safer to keep saving your results specially for searches with multiple criteria.

IMPORTANT

If you're not getting any results, check your product preference record list. Make sure that you have at least one record that matches the criteria.

EXERCISE 02: Execute Saved Search through Scripting (Required)

Scenario: SuiteDreams plans to execute the Product Shortages search as part of some periodic processing through scheduled scripts, and then set up support cases based on the search results.

This exercise is a very first step by developers to test out the execution of the saved search from within server-side JavaScript.

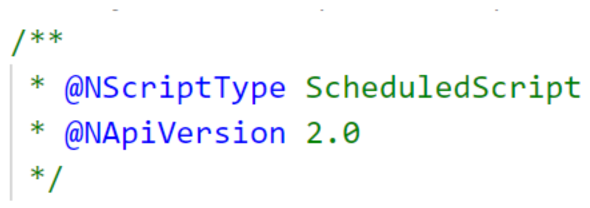
Build Script to Execute a Saved Search

1 Create a scheduled script file with the search module as a dependency.

2 Name the script file **sdr_ss_product_shortage.js**

Note: This script will be used in the next module’s exercise. For now, we’ll be running all scripts using the Script Debugger.

3 Add the following JSDoc annotations at the top of your script:



```
/**
 * @NScriptType ScheduledScript
 * @NApiVersion 2.0
 */
```

4 Using the load method, load the saved search using the search module. This will return a **Search** (search.Search) object.

BEST PRACTICES

The load method accepts a saved search's internal id or the script id as a parameter. Since the internal id will change when the search is moved from one account to another, sandbox to production for example, it's best to always use the script id.

5 Execute the search using the `run()` method of the Search object then use the `getRange()` method on the resulting object to get the first 1,000 results.

Note: The `getRange()` method needs a payload object with two properties, **start** and **end**. These properties define the index numbers for the results. To get only 10 results use, 0 as a start value and 9 as an end value.

TIPS AND TRICKS

You don't have to put each method call result in a separate variable; chain them together instead.

```
var searchResults = mySearch.run().getRange(...);
```

Login to Debugger for Testing

6 Go to Customization > Scripting > Script Debugger and login to the debugger domain.

7 Copy the codes from your script file to the debugger's editor window.

8 Replace the **define** statement with a **require** statement for debugging.

Note: The require statement syntax is very similar to the define statement. The only difference is that the define statement returns an object and the require doesn't.

DID YOU KNOW?

Debugging scripts in both the client and server side uses the **require** statement instead of **define**. To convert your require statement into define, just remove the return statement and replace the define keyword with require.

If the function is implemented inside your return statement, move it the return before deleting it.

Test the Script

9 Set the API VERSION to **2.0** and click the Debug Script button.

10 Put a breakpoint at the statement that returns the search result and click the **Continue** (play icon) button.

11 Go to the Local Variables subtab and check the search result values. It should still be empty at this point.

12 Click the **Step Over** button to execute the search and check the results again. Notice that the Local Variables subtab is empty.

Note: Executing the last line of the code ends the debugging session which prevents you from inspecting the contents of the variable from the last call.

13 Add a **var x = 0;** line after your getRange() call.

TIPS AND TRICKS

This dummy statement is called a stopper line and is used to pause the execution so that you can inspect the value of the last executed statement without ending the debugging session.

Be mindful of stopper line as you might accidentally copy it back to your script. To prevent this, you can use statements like `var stop = 'this is a stopper line';` to indicate that it's not part of your code.

14 Debug the session again and pause the execution at your stopper line.

15 Inspect the search result and verify that the result matches the execution in the NetSuite UI.

EXERCISE 03: Execute Custom Search through Scripting (Required)

Scenario: SuiteDreams has determined that searching for a product shortage may need to be a little more dynamic in terms of the filtering that is required. To support this going forward; SuiteDreams is having its developers convert the current saved search into a manually created search from within the JavaScript.

Plan Your Script Search

1 Prepare your search by knowing what you're going to use in your script. Use the following as a guide

Search Filter	PREFERRED QUANTITY is greater than 2 and the CUSTOMER's SUBSIDIARY is HEADQUARTERS : AMERICAS : US - West
Search Column	Customer CUSTOMER'S EMAIL CUSTOMER'S SUBSIDIARY Item Preferred Quantity AVAILABLE (FROM THE ITEM RECORD)

2 Plan your search filter configuration. Determine the **ID**s you're going to use for the field, join, operator and value.

Field	**Join**	**Operator**	**Value**

Note: Remember to use the **SuiteScript Records Browser** to get the appropriate IDs. For IDs belonging to your custom record, use the IDs in the custom record definition.

Also, the internal IDs for the subsidiary fields is listed in **Setup > Company > Subsidiaries**.

3 Similar to the search filters, plan your search filters.

Field	Join

Build Your Search

4 Go to your IDE to start building your search. First comment out search.load statement from the previous exercise.

5 Just before the run method call, create your search.Search object. This is similar to what you've used in the previous exercise but this time use the create method of the search module.

Note: In this step we need to pass an object with three properties: the search's record `type`, an array of search `filters`, and an array of search `columns`.

6 For your search filters, you need to build an array of `search.Filter` objects. You can create this object using the search module's `createFilter()` method.

Note: Use the IDs that you've gathered in the earlier steps.

TIPS AND TRICKS

An easier way to build searches is by using Search Expressions. Please refer to **Appending A: Search Expressions** for more information.

7 Building a search column array is similar to the search filters. Use the `createColumn()` method of the search module.

Note: You can get the internal ids of subsidiaries at **Setup > Company > Subsidiaries**.

Test the Search

8 Go back to the Script Debugger page and copy your script to the Editor window.

Note: Remember that you should be using the require function to use the debugger.

9 Debug the script and go to the **Local Variables** subtab to inspect your search result.

10 Your script is complete if it's returning the same values as the saved search you've created in the previous exercise.

EXERCISE 04: Log Script Search Results (Required)

Scenario: Complete the product shortage search by logging the results of the search.

Parse the Results

1. Go to your IDE and modify your product shortage script search.
2. After your call to run the search, add a for loop to iterate through the results.
3. In each element of the array, extract the field values by using the either the `getValue()` or the `getText()` method.
4. Once you've gotten your field values, log them using the `log.debug()` method.

Test the Completed Search

5. Go back to the Script Debugger and copy the script to the editor.
6. Run the search completely. Your script is complete if the **Execution Log** subtab displays all the information that you've logged in your script.

EXERCISE SOLUTIONS

EXERCISE 01: Create a Saved Search

4 Select the appropriate Search Type or record on the New Saved Search page.

For which record type are we selecting a list of records?

Answer: Product Preferences.

EXERCISE 03: Execute Custom Search through Scripting

2 Plan your search filter configuration. Determine the **ID**s you're going to use for the field, join, operator and value.

Field	**Join**	**Operator**	**Value**
custrecord_sdr_prod_pref_qty	*<none>*	greater than or equal to	2
subsidiary	custrecord_sdr_prod_pref_customer	any of	1 (for US West)

3 Similar to the search filters, plan your search filters.

Field	**Join**
custrecord_sdr_prod_pref_customer	*<none>*
email	custrecord_sdr_prod_pref_customer
subsidiary	custrecord_sdr_prod_pref_customer
custrecord_sdr_prod_pref_item	*<none>*
custrecord_sdr_prod_pref_qty	*<none>*
quantityavailable	custrecord_sdr_prod_pref_item

BULK PROCESSING (PART 1)

MODULE EXERCISES

Required Exercises	
01	Regularly Log Product Shortages
Optional Exercises	
02	Create a Support Case

EXERCISE 01: Regularly Log Product Shortages (Required)

Scenario: SuiteDreams wants the existing product shortage search to run on a scheduled basis.

Modify the Product Shortage Search

1 Go back to the IDE and edit the script from the previous exercise by converting the require statement back to a define statement.

Note: Remember to return an object with the function named **execute**.

2 Add the two required SuiteScript 2.0 annotations for the script.

Note: The **NScriptType** value for scheduled scripts is **scheduledscript**.

3 Move your existing search implementation inside your execute function.

Note: This will allow your script to be triggered as a scheduled script entry point.

Create the Script Record

4 Upload the script to the File Cabinet.

5 Create a script record for your script with the following configuration:

NAME	Product Shortages Log
ID	_sdr_ss_product_shortages
DESCRIPTION	*<Enter a meaningful description>*

6 Hover over the **Save** button and click on **Save and Deploy**.

Note: You can also use the **Deployment** subtab to configure your deployment record. For this exercise though, we'll be using the deployment record page to get more details.

7 Configure your deployment record with the following:

TITLE	Product Shortages Log
ID	_sdr_ss_product_shortages
STATUS	Not Scheduled
Schedule *(subtab)*	SINGLE EVENT

BEST PRACTICES

When creating scheduled scripts, execute the script immediately to check if everything is working. Once you're sure that the script is working fine, then go ahead and set the schedule that you need.

8 **Save** the script deployment.

Note: The script deployment needs to be initially saved before it can be executed.

Test the Script

9 **Edit** the deployment record again then hover over the **Save** button then click on **Save and Execute**. This will redirect you to the **Scheduled Script Status** page.

10 Click **Refresh** after a few seconds to check if the execution is completed.

Note: If you accidentally clicked away from the page, you can go back by going to **Customization > Scripting > Scheduled Script Status**.

11 Click your script's deployment id to go back to the deployment record.

12 Go to the **Execution Log** subtab and verify if the logs where triggered successfully.

The exercise is complete if all the search results were logged.

EXERCISE 02: Create a Support Case (Optional)

Scenario: Right now SuiteDreams has to look through every execution log entry from the Product Shortages search to determine whether a product shortage warrants additional action. What they want is to have the system automatically generate a support case for each product shortage search result where the available quantity of an item is less than the preferred quantity on a product preference. The search parameters should not be changed. This should be additional processing performed on the search results.

Create Support Case

1. Create a **support case** (**Lists > Support > Cases > New**) to familiarize yourself to the record, paying attention to the mandatory fields.
2. Go to the scheduled script from the previous exercise and after the logging statement (inside the for loop), add a condition to support the exercise scenario.

   Note: If the available number of items go below the customer's preferred quantity, the system will be creating a support record.

   Also, make sure to surround your quantity values with `parseInt()` to make sure that you're getting native number values.
3. Load the record module to the script.
4. Create a support case object with the following information:

Subject	Item low for customer
Company	<Company returned by search result>
Message	This company prefers to purchase <preferred quantity> <item name> each time they create a sales order, but only <available quantity> are left in stock.

   Note: Use the SuiteScript Records Browser to get the field IDs.

5. Submit the support case record to the database.

Test

6 Update the preferred quantity on some of your product preferences so they will exceed the available quantity. If you enter 999 for preferred quantity, then this should exceed the available quantity for most items.

Note: If you configure the product preferences directly from the customer record, you will need to update the validation from one of the previous exercises to allow for a higher level of preferred quantity, or you can remove the validation entirely.

7 Verify that the case record was created.

BULK PROCESSING (PART 2)

MODULE EXERCISES

Required Exercises	
01	Determine payment amounts per customer

EXERCISE 01: Determine payment amounts per customer (Required)

Scenario: As part of the reporting process, the company wants to get a report on all deposited and undeposited accounts per customer.

Note: To keep the exercise simple, we'll be logging the values instead of creating an actual report.

Create a Payment Search

1 Go to the NetSuite UI and create a **Transaction** search (**List > Search > Saved Searches > New**).

2 Use the following settings for the search:

SEARCH TITLE	Customer Payments
ID	_sdr_payments

FILTER	DESCRIPTION
Type	is Payment
Main Line	is true

Columns : FIELD
Name
Status
Amount Paid

Note: We're creating a saved search instead of a script search to use the reference object format for the map/reduce.

3 **Save & Run** the search to check if you're getting the right results.

Create the Script

4 Create a map/reduce script and name it **sdr_mr_payment_report.js**. Include the **N/search** module to the script.

5 Add the following JSDoc annotations at the top of your script:

```
/**
 * @NScriptType MapReduceScript
 * @NApiVersion 2.0
 */
```

6 In the **getInputData**() return an object reference pointing to the saved search you've created.

Note: An object reference is simply a payload object that has type and id. For example:

```
return {
    type : 'search',
    id   : 1234
}
```

7 Create a **map**() function in your script.

8 Extract the search result value from the map's context object. This is stored in the **value** property.

Note: Remember that the map stage processes a single search result value per invocation. You need not loop through the results as you would in other script types.

9 Log resulting **value** property of the context object to inspect the JSON string.

Initial Test

10 Upload the script to the File Cabinet.

11 Create a script record and click **Save and Deploy** to create a deployment record.

Note: Take note of the ID that you use for the deployment record.

12 On the Script Deployment page, specify and ID then **Save** the record.

Note: Feel free to initially modify the saved search to get fewer results. This would make the execution faster.

13 Edit the deployment record again and execute the map/reduce script.

14 Once the execution is complete, go to the **Execution Log** and take note of the JSON structure.

TIPS AND TRICKS

There are several JSON editors/formatters you can use online to make it easy to look at the structures of a JSON string.

Total Customer Payments

15 Go back to the script and convert the JSON string to a JavaScript object using **JSON.parse()**.

Note: Remember that the system will be returning a JSON string value. Using the JSON.parse() function makes the value easier to handle.

16 Write a key/value pair back into the **context** using the **write()** method of the context object. Use the customer name as a **key** and the status & amount as the **value**.

17 Create a **reduce**() function in your script.

18 Extract the array of values that was associated to the key in the previous stage. You can get this from the **values** parameter of the reduce's context object.

Note: Similar to map function, the reduce function processes each individual key/value pair so there's no need to iterate through the results to get one pair.

19 Create a variable that will hold the deposited and undeposited values. Initialize those values to 0.

20 Loops through the values array, extract one element of the values array and put that in a variable.

Note: Since you're using an object as a value for your key/value pair, you need to convert that individual element from a JSON string to JavaScript using JSON.parse().

21 Add a condition to check the values to total all deposited and undeposited amounts.

22 Log both the name and the combined totals for each customer.

Display Summary Statistics

23 Extract the following values from the summarize's summary object.

- Usage Consumed (usage)
- Number of Queues used (concurrency)
- Number of Yields done (yields)

Test

24 Go back to the script deployment and execute the script again.

25 Click the **Details** link to view the status of execution of all stages.

26 Go back to the script or deployment record's **Execution Log** subtab. The exercise is complete if the invoice data is properly logged.

SCRIPT PARAMETERS

MODULE EXERCISES

Required Exercises	
01	Deployment Specific Script Parameters
02	User Specific Script Parameters
Optional Exercises	
03	Offload User Event Script Processing

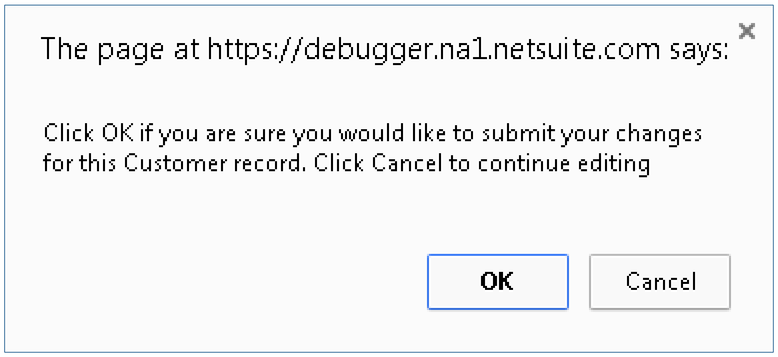
EXERCISE 01: Deployment Specific Script Parameters (Required)

Scenario: Display a confirmation message when saving a customer and partner record. The message should mention which record the user is trying to save.

IMPORTANT

This exercise is an example of how to pass values from deployment level script parameters to the script. Normally if you need to get the type of a particular record, you would be getting that from the type parameter of your record object.

Sample confirmation message:



The page at https://debugger.na1.netsuite.com says:

Click OK if you are sure you would like to submit your changes for this Customer record. Click Cancel to continue editing

Create a New Script File

1 Create a new client script file and add the **runtime** module as a dependency.

2 Create a `saveRecord` function, add an alert statement that would display the message mentioned in the scenario.

3 Upload the file to the File Cabinet and create a script record.

4 Deploy the script record to a customer, vendor, and partner record.

Create Script Parameter

5 Click the **Parameters** subtab then the **New Parameter** button to create the script parameter.

6 Create a script field with the following configuration:

LABEL	Save Confirmation Message – Record Type
ID	_sdr_save_record_type
DESCRIPTION	*<Enter a meaningful description>*
TYPE	Free-Form Text
PREFERENCE	*<Blank>*

7 **Save** the script parameter.

Configure Script Parameter for Each Deployment

8 Go to the **Deployments** subtab and open the script deployment records.

9 Open both the **Customer** and **Partner** deployment records in separate subtabs.

10 Go to the Customer deployment record and **Edit** it.

11 In the **Paremeters** subtab, set the SAVE CONFIRMATION MESSAGE – RECORD TYPE field to **Customer** then **Save**.

12 Next go to the Partner deployment record and do the same thing, this time setting the field to **Partner**.

Access the Script Parameter from the Script

13 Go to back to your client side script.

14 In your saveRecord function just above your confirm statement, get the script object using the `getCurrentScript()` method of the runtime module.

15 Once you have your script object, get the script parameter value using the script object's `getParameter()` method.

Note: The payload object for the `getParameter()` method has one property, **name**, which accepts the id of the script parameter.

16 Edit your **confirm** statement so that the message incorporates the value from the script parameter.

Test

17 Edit an existing customer record then save it. Check that the proper message is displayed.

18 Also edit a partner record and check the message. The exercise is complete if the confirm message adapts to the record being saved.

Note: Exercise is complete when the confirmation message properly displays in both record types.

EXERCISE 02: User Specific Script Parameters (Required)

Scenario: The prompting of an "are you sure" message upon saving a customer, partner, or vender should be configurable on a per user basis.

Create the Script Record

1. Go back to your client side script record.
2. Add a new script parameter with the following configuration:

LABEL	Display Save Confirmation
ID	_sdr_save_confirmation
DESCRIPTION	*<Enter a meaningful description>*
TYPE	Check Box
PREFERENCE	User
DEFAULT CHECKED (UNDER THE **Validation & Defaulting** SUBTAB)	*<checked>*

Modify Script

3. Go to back to your script and navigate to the lines that you've just edited.
4. Get the value of the save confirmation script parameter and use the value to support this requirement:

   If the DISPLAY SAVE CONFIRMATION field is checked, display the confirmation message; otherwise, continue saving without displaying the message.

   Note: A checkbox field will return a boolean value.

Test

5. Edit a customer record and save. Confirm that the message still displays.
6. Go to **Home (icon) > Set Preferences**.
7. In the **Custom Preferences** subtab, uncheck the DISPLAY SAVE CONFIRMATION field then **Save**.
8. Edit and save a customer record again. The exercise is complete if the confirmation does not appear when a customer or partner record is saved.

EXERCISE 03: Calling Map/Reduce scripts (Optional)

Scenario: SuiteDreams requests that the payment summary be specific to a customer. And to automate the process, it would be automatically triggered when a customer record is saved.

Note: This exercise continues the exercise from the map/reduce module and covers how to pass values from one script to another using script parameters. This process can also be applied to other script types like scheduled scripts, and csv import.

Edit the User Event Script

1. Go to your IDE and edit the **sdr_ue_customer.js** script.
2. Add the **N/task** module to your script.
3. Create a task object for your map/reduce script.
4. Copy the map/reduce's scriptId and deploymentId. Set these values as property values to your task object.
5. Go to the map/reduce's script record and create a free-form text parameter to hold the internal id of the customer. Take note of the script parameter id.

   Script Parameter ID:
6. In your user event script, pass the customer id as a parameter in the task object. The parameter payload is a key/value pair with the script parameter ID as the key.

IMPORTANT

Make sure to use the script parameter id exactly or it will not work.

7. Use the task object's submit() method to execute the map/reduce script.
8. Upload your changes to the File Cabinet.

Modify the Map/Reduce Script

9. Go to your map/reduce script and load the N/runtime module.
10. Using the runtime method, load the map/reduce's script object and get the value of the script parameter.

11 Add the customerId from the script parameter to the search so only invoices from that particular customer will be processed.

IMPORTANT

Since the search is loaded from the UI, you need to recreate the search in the script to incorporate the customer id passed from the user event script.

12 Upload your changes to the File Cabinet.

Test

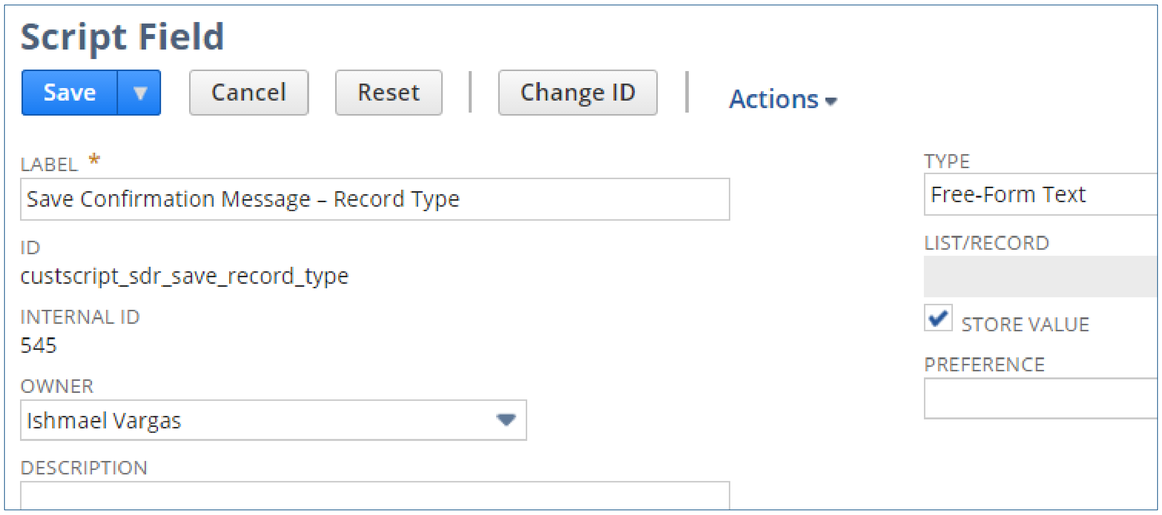
13 Test by saving a customer record to trigger the script.

The exercise is complete if the invoice was logged for the saved customer.

EXERCISE SOLUTIONS

EXERCISE 01: Deployment Specific Script Parameters

Script parameter configuration for the SAVE CONFIRMATION MESSAGE – RECORD TYPE field:



LABEL *
Save Confirmation Message – Record Type

ID
custscript_sdr_save_record_type

INTERNAL ID
545

OWNER
Ishmael Vargas

DESCRIPTION

TYPE
Free-Form Text

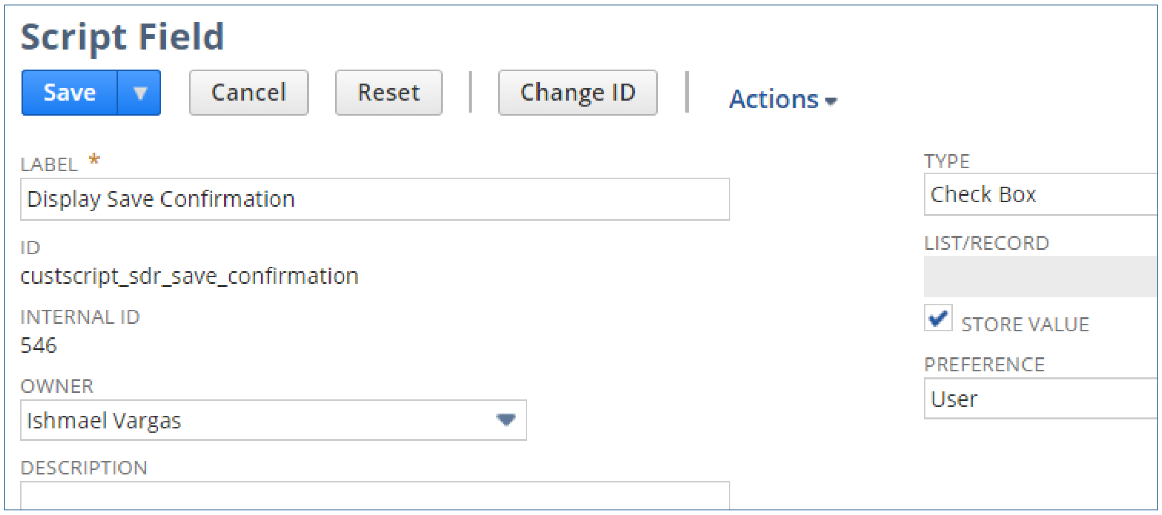
LIST/RECORD

✓ STORE VALUE

PREFERENCE

EXERCISE 02: User Specific Script Parameters

Script parameter configuration for the DISPLAY SAVE CONFIRMATION field:



LABEL *
Display Save Confirmation

ID
custscript_sdr_save_confirmation

INTERNAL ID
546

OWNER
Ishmael Vargas

DESCRIPTION

TYPE
Check Box

LIST/RECORD

✓ STORE VALUE

PREFERENCE
User

Workflow Action Scripts

MODULE EXERCISES

Required Exercises	
01	Create Sales Order Workflow
02	Create Script to Update Sales Order
Optional Exercises	
03	Alter Workflow Based on Result of Custom Action

EXERCISE 01: Create Sales Order Workflow (Required)

Scenario: Most people that enter sales orders for SuiteDreams end up needing to place some notations onto the related customer record. To speed this up; a workflow is created to automatically move the end user to the customer record after a sales order is submitted.

Define Workflow

1. Create a workflow at **Customization > Workflow > Workflows > New**.
2. Create your workflow using the following configuration:

Basic Information	
NAME	Process Sales Order
ID	_sdr_process_sales_order
RECORD TYPE	Transaction
SUB TYPES	Order
DESCRIPTION	*<Enter a meaningful description>*
RELEASE STATUS	Testing
Event Definition	
ON CREATE	*<checked>*
ON UPDATE	*<checked>*
TRIGGER TYPE	After Record Submit

ORACLE NETSUITE

3 **Save** your workflow.

Set Up the Start State

4 In your **Workspace** pane, double-click **State 1**.

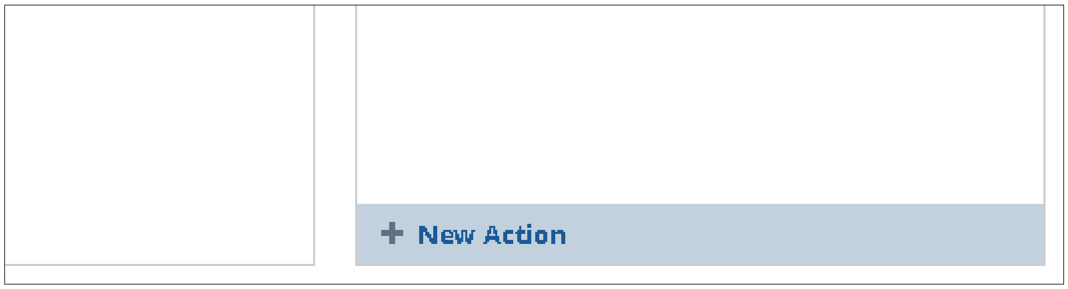
Note: Alternatively, you can also click on the pencil icon in the **State** subtab.

5 Use the following configuration then **Save**:

NAME	State 1: Entry
DESCRIPTION	*<Enter a meaningful description>*

Configure Go To Record Action

6 While State 1: Entry is still selected, click on the **+ New Action** button at the lower right hand corner of the screen.



7 Add a **Go To Record** action with the following configuration:

RECORD TYPE	Customer
FIELD	Entity
OPEN IN EDIT MODE	<checked>

8 **Save** the action.

Test

9 **Edit** your Sales Order script and undeploy it by unchecking the DEPLOY option under the **Deployments** subtab.

Note: The previous script can slow down your testing and needs to be disabled.

10 Open an existing sales order or create a new one (**Transactions > Sales > Enter Orders**).

11 Save the sales order and the end user should be automatically taken to the customer record defined in the Customer field of the sales order.

12 **Save** the record.

The exercise is complete if you were automatically redirected to the customer record that is on the sales order.

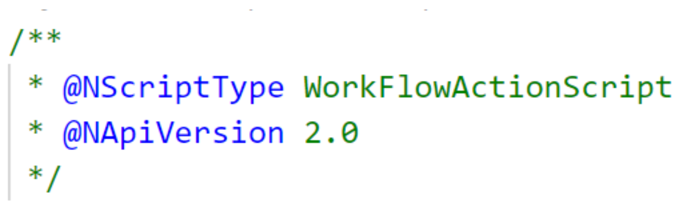
EXERCISE 02: Create Script to Update Sales Order (Required)

Scenario: Update the related customer record with a notation about the sales order, and then navigate the end user to that record (as is being done currently).

Create Workflow Action Script

1 Create script file and name it **sdr_wf_update_customer.js**. Load the **record** and **runtime** modules as a dependency.

2 Add the following JSDoc annotations at the top of your script:



```
/**
 * @NScriptType WorkFlowActionScript
 * @NApiVersion 2.0
 */
```

3 To start, add the lines that will get a script parameter. Use the ID, **custscript_sdr_order_date**.

Note: We'll be creating this later after we create the script record.

4 Get the sales order record object from your entry point's context object.

Note: This is stored in the same property as your user event scripts.

5 Get the number of line items in the Items sublist.

Note: Don't forget to use the SuiteScript Records Browser to get the proper ids.

6 Create a notes variable that contains the following as a string:

- Last Order Date: <order date>
- Unique items ordered: <count of items sublist>

Note: You can append **\n** to place a carriage return into your string.

7 Get the internal id of the customer on the sales order record. Use this to load the customer record object.

8 Once you have the customer object, update the COMMENTS field with the value from your notes variable.

9 Create script and deployment record for workflow action script. Use the following values:

Script Record

NAME	Update Customer
ID	_sdr_wf_update_customer
DESCRIPTION	*<Enter a meaningful description>*

BEST PRACTICES

The name that you use in your workflow action's script record will be displayed in the list of action on the workflow. Using your normal naming convention for scripts can potentially confuse your workflow users.

To help with the use the following format for your names: <verb> + <description>. For example, "Send email", "Reset form values", or "Hide related record".

Script Parameter (under the **Parameters** subtab)

LABEL	Order Date
ID	_sdr_order_date
TYPE	Date
PREFERENCE	*<blank>*

Note: Make sure that the ID you use here and your script are the same, otherwise you will get an error message.

Deployment Record (under the **Deployments** subtab)

APPLIES TO	Order*
ID	_sdr_wf_update_customer

Note: *Remember that records in NetSuite can be renamed. In the training account, the Sales Order record has been renamed to Order but it is referring to the same record type.

10 **Save** the script record.

Use Script on Workflow

11 Go back to your **Process Sales Order** workflow and click on **State 1: Entry** in the diagram.

12 Click the **+ New Action** button and look for the script that you created.

Note: Remember that it will use the name that's in your script record.

13 Change the TRIGGER ON value to **After Record Submit**.

14 Scroll down to the **Parameters** section and choose Date in the VALUE FIELD column for the **Order Date** field. This copies the date from the sales order and passes it to the script parameter so the script would be able to use it.

15 **Save** your action.

Test

16 Test the workflow by opening an existing sales order or creating a new one.

17 **Save** the sales order. The exercise is complete if you see the notes from your script displayed in the COMMENTS field of your customer.

Note: After you're done, disable the workflow by setting it as inactive. This prepares the account for the next module.

CAUTION

If you don't see the Comments get updated, it may just be because the browser did not load the latest copy of the page. Open the customer in another browser tab to check if the record was updated.

EXERCISE 03: Alter Workflow Based on Result of Custom Action (Optional)

Scenario: The update to the related customer record could potentially fail, though unlikely. Workflows can be designed to properly handle failures such as this, or from any other custom action. To see how this works, the workflow is going to be modified to branch out to one of two different end states depending upon the success of the custom action.

Note: Enable the workflow if you disabled it in the previous exercise.

Return Status from Script

1 Go back to your IDE and edit your script. Have it return “SUCCESS” if the customer record was properly updated and “FAILED” if it wasn’t.

Note: The save() method returns the internal id of the successfully saved record. You can use this as a condition.

2 Move on to the browser and **Edit** the script record.

3 In the **Parameters** subtab, set the RETURN TYPE to Free-Form Text. This tells the system to expect the script to return a value.

4 **Save** your changes.

Add Workflow State Field

5 Edit the Process Sales Order workflow.

6 Click on **State 1: Entry** then select the **Fields** button. At the bottom of the page, click **+ New State Field** button.

Note: This field will hold the value that’s returned by the script.

7 Configure the state field using these settings:

LABEL	Customer Update Status
ID	_sdr_customer_update_status
DESCRIPTION	*<Enter a meaningful description>*
TYPE	Free-Form Text

8 **Save** the state field.

9 Click on the **Actions** button and edit the **Update Customer** action.

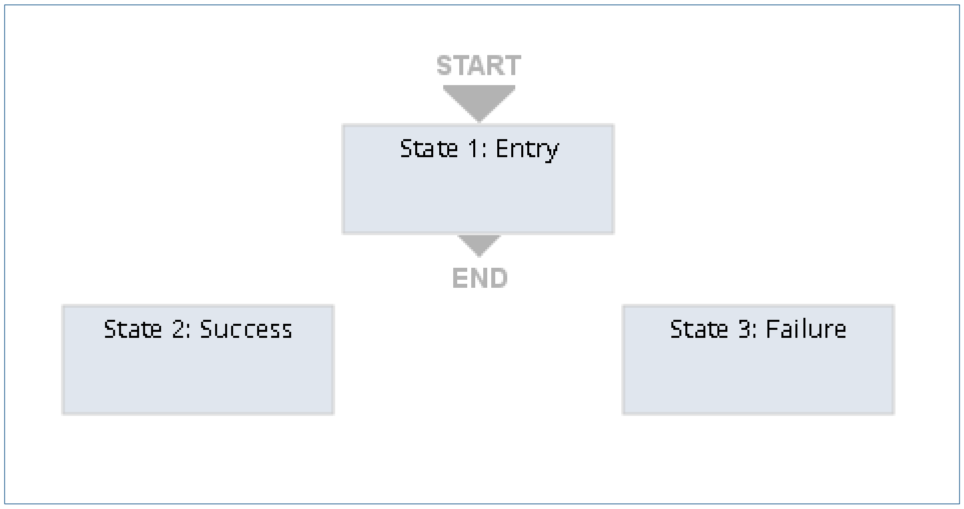
10 Scroll down and look for the STORE RESULTS IN field. Set this to the state field you've just created.

11 **Save** your changes to the action.

Add New States

12 Click on the **New State** button to add two new states.

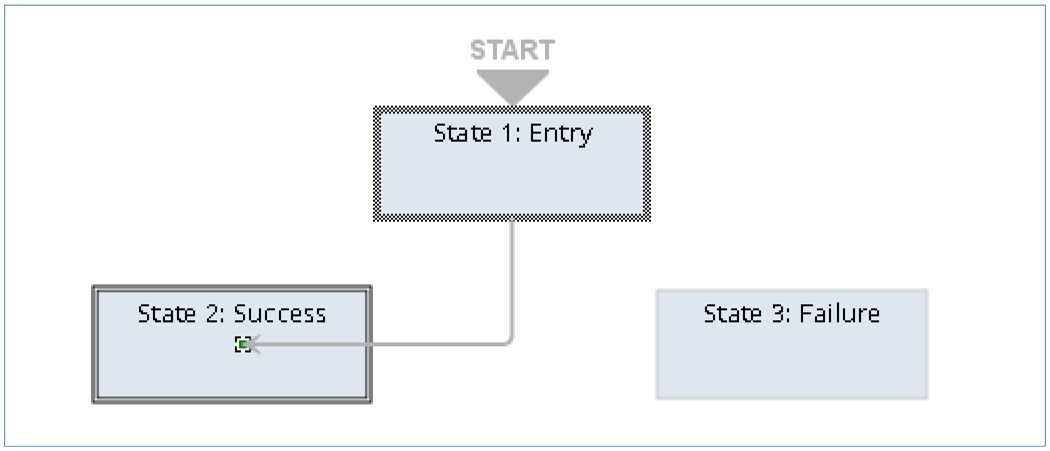
13 Call one **State 2: Success** and **State 3: Failure** for the other.



Note: You can re-arrange the workflow states in any direction you choose.

Connect the States

14 Add a transition from State 1 to State 2 by dragging the transition handles (half circle) from one state to the other.



15 Double-click on the transition line to edit it.

Note: Similar to the state, you can also click on the pencil icon in the Transition pane while it's highlighted to edit it.

16 Hover over the CONDITION field and click the open button that appears beside it. Use this for your condition:

FIELD	Customer Update Status (State)
COMPARE TYPE	Equal
VALUE	SUCCESS

17 **Save** the condition and the transition.

18 Repeat the process with State 3, from creating the transition and the condition, but this time use the value FAILED for your condition.

Test

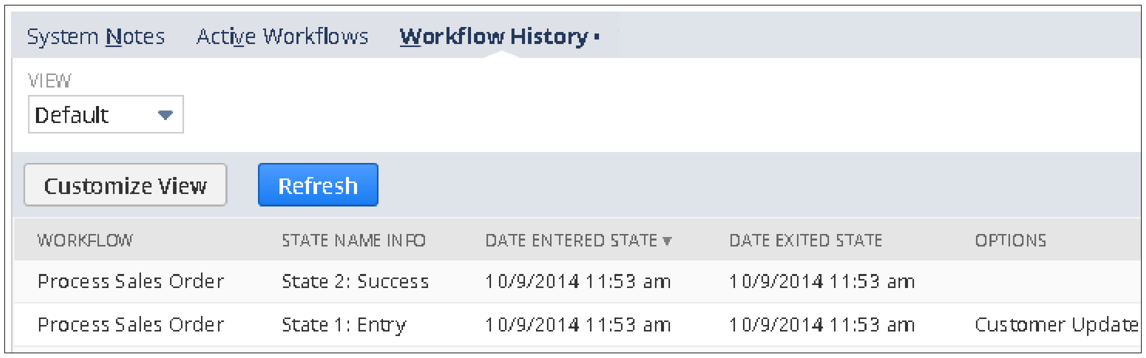
19 Open an existing sales order or create a new one then **Save** it.

20 Now re-open the sales order that you've just saved.

Note: You can re-open the record using the Recent Records options.

21 Go to the **System Information** subtab then the **Workflow History** subtab.

22 Check the log entries. The exercise if complete if the workflow transitioned from State 1 to State 2 (or State 3 depending on the status).



WORKFLOW	STATE NAME INFO	DATE ENTERED STATE ▾	DATE EXITED STATE	OPTIONS
Process Sales Order	State 2: Success	10/9/2014 11:53 am	10/9/2014 11:53 am	
Process Sales Order	State 1: Entry	10/9/2014 11:53 am	10/9/2014 11:53 am	Customer Update

Note: After you're done, disable the workflow by setting it as inactive. You can do this by clicking on the pencil icon in the **Workflow** subtab.

This prepares the account for the next module.

NETSUITE PAGES

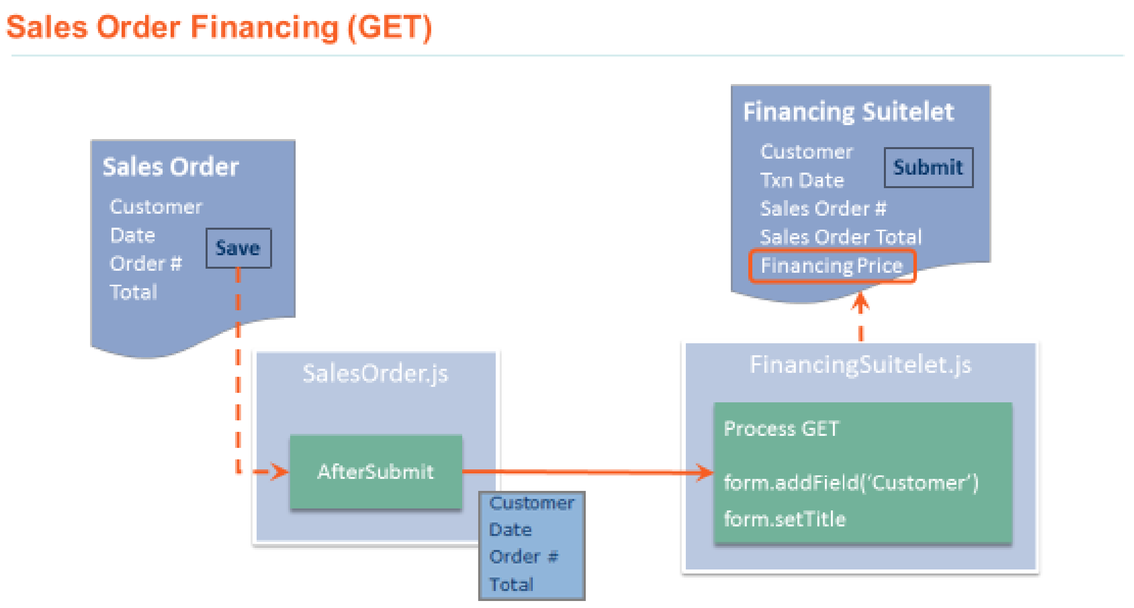
MODULE EXERCISES

Required Exercises	
01	Create Custom UI Page
02	Process Data from Sales Order
02	Return to the Sales Order

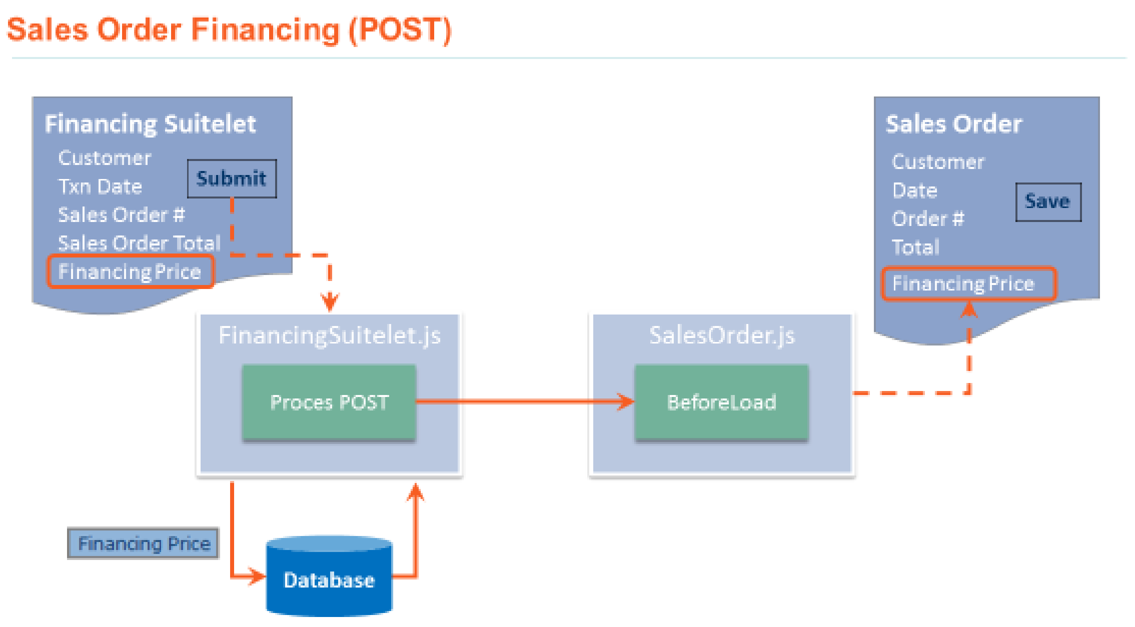
EXERCISE DIAGRAM

IMPORTANT

Here's an illustration on what you will be doing in the following exercises.



For the first part, the user is automatically directed to the suitelet as soon as the sales order record is saved. The user will then update the FINANCING PRICE field through the suitelet.



The use would then be saving the information by submitting the suitelet. The suitelet would then redirect the user back to the sales order record to verify if the financing price was updated.

EXERCISE 01: Create Custom UI Page (Required)

Scenario: A sales order financing program is added to sales orders through creation of a form Suitelet.

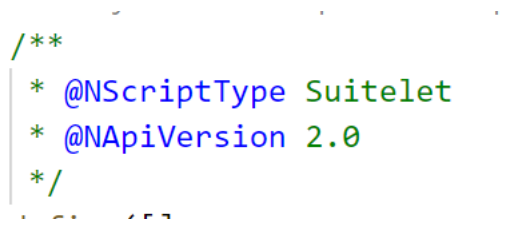
When an order is submitted, users are redirected to a custom page where they will enter the financing information. They will then submit that information to the system and get redirected back to the order. The system will be updating the finance information in the background.

Note: This is the first of four exercises to configure the Suitelet. This exercise creates a form, adds some help text, adds a button, and configures the Suitelet to be selectable from the menu.

Create Suitelet Script

1 Create a script. Name it **sdr_sl_salesorder_finance.js** and include the **record**, **redirect**, **ui/serverWidget** modules as dependencies.

2 Add the following JSDoc annotations at the top of your script:



```
/**
 * @NScriptType Suitelet
 * @NApiVersion 2.0
 */
```

3 Create your base form object by using the ui/serverWidget module's **createForm()** method. This will return a **serverWidget.Form** object.

4 Set the **title** to "Sales Order Financing".

5 Before adding the fields, it's important to remember the id naming convention for suitelets.

What system prefix should be used for field IDs when creating fields on form objects?

Answer:

6 Add fields to the form by using the `addField()` method on your form object. Use the following configuration:

id	<System prefix> + '_sdr_financing_help'
label	Please assign a price to the financing of this sales order, then click Submit Financing
type	FieldType.HELP **Note**: FieldType is an enum value stored in the ui/serverWidget module.

Note: This will return a serverWidget.Field object. Store that in a variable so we can modify its properties.

TIPS AND TRICKS

If you're not using the Hungarian Notation on your variables, consider adding "fld" at the end of your variable names. This helps differentiate the field object from field values.

Note: Please see the Appendix for more information about the Hungarian Notation.

7 Add a submit button to your form to allow the user to send information back to the server. Set the title to **Save Finance Info**.

DID YOU KNOW?

There are two ways of adding buttons to the form, the addButton() and the addSubmitButton(). Custom pages where users enter data should have the submit button otherwise the data will be lost. Regular buttons are used for a more custom processing. This is done by attaching a function to the button which will be called when it's pressed.

8 Render the page by writing it to response. This is done by passing the form object to the `writePage()` method of the response object as a parameter.

Note: The response object can be obtained from the context object through a property by the same name.

Create the Script Deployment

9 Create a script deployment with the following configuration:

NAME	SuiteDreams SL Sales Order Finance
ID	_sdr_sl_salesorder_finance
DESCRIPTION	*<Enter a meaningful description>*

10 Click **Save and Deploy**.

11 Configure the deployment using the following:

TITLE	Sales Order Financing
ID	_sdr_sl_salesorder_finance

Note: Don't save the script deployment yet.

12 Go to the **Links** subtab and add the following link configuration:

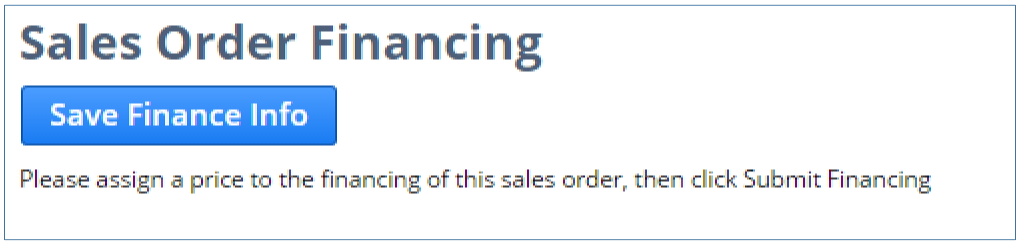
CENTER	Classic Center
SECTION	Setup
CATEGORY	Custom
LABEL	Sales Order Financing

Note: Initially, the suitelet will be accessed through this link for faster testing. Later we'll be redirecting to the suitelet directly from the sales order form.

13 **Save** your script deployment.

Test

14 Select the Suitelet from the **Setup > Custom** menu. Alternatively, you can also click on the link in the URL field of your script deployment. It should look similar to the following:



Please assign a price to the financing of this sales order, then click Submit Financing

Note: If you do not see the link in the menu, try clearing cache using CTRL + F5 or going into your browser options. You may also need to log off and back on.

EXERCISE 02: Process Data from Sales Order (Required)

Scenario: This exercise causes the end user to be redirected to this Suitelet upon submitting the sales order. The following sales order information is displayed on the Suitelet:

- ORDER #
- CUSTOMER
- TOTAL

Note: Turn off the workflow from the previous exercise if you haven't already done so as it can interrupt the execution of the script.

You can turn off the workflow in one of two ways:
- Edit the workflow, setting its RELEASE STATUS to **Not Running**
- Edit the workflow, checking INACTIVE.

Create Sales Order User Event Script

1 Create a user event script for the sales order record (sdr_ue_order.js) and add the **redirect** module as a dependency.

2 Get a copy of your sales order object and extract the values off the following fields:

- ORDER #
- CUSTOMER
- TOTAL

3 Using the redirect module's `toSuitelet()` method, send the user to your custom page.

Note: The `toSuitelet()` method needs an object with three properties: the scriptId, deploymentId, and parameter list. The parameter list is an object with key/value pairs that you'll be sending to your suitelet.

BEST PRACTICES

When creating your parameter object, it's recommended that you use custparam to prefix your parameter names. For example, your parameter might look like:

```
{  custparam_sdr_name : 'Mel',
   custparam_sdr_id   : 23
}
```

While using custparam to prefix your parameters is not required but it's highly recommended. You script will most likely break if it collides with system generated URL parameters or with parameters from other scripts.

4 Create a script record for your user event script. Use the following configuration:

NAME	SuiteDreams UE Sales Order
ID	_sdr_ue_salesorder

5 Deploy the script to the **Order** record and give it the ID **_sdr_ue_salesorder**.

6 Save the script record.

Test Redirection

7 Open an existing sales order or create a new one.

8 Save the sales order. You should be redirected to your suitelet.

Handle Values from Sales Order

9 Go back to your suitelet script.

10 Extract the values that you've passed from your user event script by accessing the parameter object from your request object. For example if you passed a parameter named custparam_sdr_name, you can access it like this: `context.request.parameters.custparam_sdr_name.`

Note: Like the response object, you can also access the request object from your function's context object.

11 Add the following four fields to your form:

Note: These fields will store the parameter values from your user event script.

Label	**Type**
Order #	`FieldType.TEXT`
Customer	`FieldType.TEXT`
Total	`FieldType.CURRENCY`

Note: Remember to store each field object in a variable.

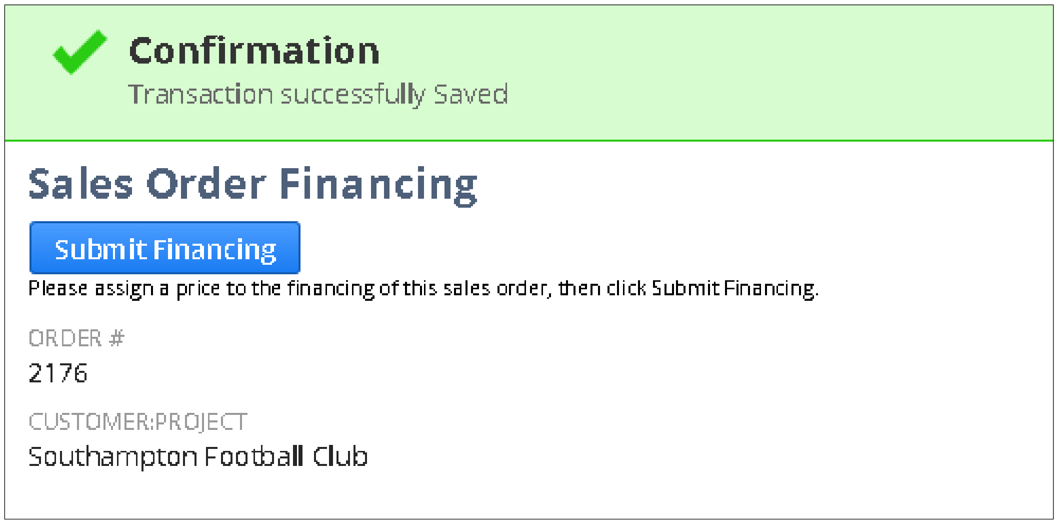
12 Use the field object's `defaultValue` property to assign the values you've extracted from the request.

13 Change the fields to inline so the user won't think that the fields can be edited. This can be done using the field object's `updateDisplayType()` method. Set it to `serverWidget.FieldDisplayType.INLINE`.

Note: The `FieldDisplayType` enum is stored in the ui/serverWidget module.

Retest Redirection

14 Repeat the test from earlier in this module. This suitelet should contain sales order data, similar to the following:


Confirmation
Transaction successfully Saved
Sales Order Financing

Please assign a price to the financing of this sales order, then click Submit Financing.
ORDER #
2176
CUSTOMER:PROJECT
Southampton Football Club

Note: Order # for new records is not available during after submit of the sales order user event script. It displays on the suitelet as **To Be Generated**. A workaround is to load the sales order from your suitelet and get the order number from there instead.

EXERCISE 03: Return to the Sales Order (Required)

Scenario: This exercise causes the end user to be redirected back to the sales order upon submittal of the suitelet. Summary of the additional processing:

- Allow entry of the FINANCING PRICE from the suitelet
- FINANCING PRICE is updated on the sales order record upon submittal of the suitelet
- FINANCING PRICE displays on the sales order

The suitelet is modified to differentiate between GET and POST request processing. Existing suitelet processing plus entry of the financing price is to occur during the GET request. Update of the sales order and redirection back to it is to occur within the POST request.

Add Financing Price

1 Create a Financing Price transaction body field (**Customization > Lists, Records, and Fields > Transaction Body Field > New**). Use the following configuration:

LABEL	Financing Price
ID	_sdr_financing_price
DESCRIPTION	*<Enter a meaningful description>*
TYPE	Currency
Applies To *(subtab)*	
SALE	*<checked>*
Display *(subtab)*	
SUBTAB	Main
DISPLAY TYPE	Inline Text

2 Open a sales order record and verify that the FINANCING PRICE field was added to the form.

Process Financing Price in Script

3 Go back to your sales order user event script.

4 Get the FINANCING PRICE field value and pass it to the request like what you've with the other fields.

5 Get the sales order's internal ID and pass that over to the request as well.

Note: We'll be using this id later in the exercise to update the sales order from the suitelet.

6 Edit your suitelet and add the FINANCING PRICE field to the form.

7 Get the FINANCING PRICE value from the request and set that as a default value.

Note: Don't set the field as inline because the users will be editing that value. Empty fields are returned as undefined so make sure to check for that when setting the value.

8 Add the sales order id to the form and process it similar to the FINANCING PRICE field. Set the display type to HIDDEN.

TIPS AND TRICKS

Passing internal ids to through the request is a faster way to process records in your suitelet. In this case if we didn't pass the internal id, we would have to search for the record using the record number. While that works, it would be more complicated and a lot slower.

If you are going to do this, always hide your field. This is because users would rarely need the internal id information. Remember that the less clutter you have on your form, the more users would be able to focus on what they need to do.

9 Test by creating or editing an existing sales order and saving it.

Differentiate Between GET and POST Request

Note: By determining how the user is sending the request, we'll be able to filter the execution of our suitelet. If the user sends a GET request (script redirections are GET requests) then the suitelet will be displaying the form. If the user sends a POST request (form submissions are POST requests) then the suitelet will be updating the sales order and will redirect back to the sales order.

10 In your suitelet script, add an if statement to check if the request if it is a GET request.

Note: This value can be tested against the **method** property of your **request** object. Use the string "GET" for your condition.

11 Move all form related statements inside your if statement so that the form will be displayed if the user is sending a GET request.

Note: This includes processing the parameters and writing to the response.

12 Add an else block. This is where the form submission will be processed.

To load a record, which SuiteScript API module would you need to include in your script?

Answer:

13 Add the module you specified in the previous step.

14 Load the sales order record using the id from the request.

CAUTION

Since you're processing a POST request, that means your data is coming from your form, not from your GET request parameters (passed from the user event script).

Getting the data from the form is similar to GET request. You will be extracting the values from the parameters property of your request object. The difference is that you'll be using the ids of the form fields. Again, this is because you're processing the form that you've submitted.

15 Set the financing price value on your sales order from the updated value that the user entered on the suitelet.

16 Save your changes to the sales order object.

17 Redirect the user from the suitelet back to the sales order. This is done using the redirect module's `toRecord()` method.

Note: Use the Type enum from the record module to set which record the user will be redirected to.

Test

18 Perform the same test that you've done previously. The exercise is complete if the user was able to update the financing price field from the suitelet and get redirected back to the sales order to confirm that the field was updated.

EXERCISE SOLUTIONS

EXERCISE 01: Create Custom UI Page

4 What system prefix should be used for field IDs when creating fields on form objects?

Answer: custpage

Note: The internal ID must be in lowercase, contain no spaces, and include the prefix custpage if you are adding the field to an existing page. For example, if you add a field that appears as Purchase Details, the field internal ID should be something similar to custpage_purchasedetails or custpage_purchase_details.

EXERCISE 03: Return to the Sales Order

12 To load a record, which SuiteScript API module would you need to include in your script?

Answer: The record module.

WEB SERVICES

MODULE EXERCISES

Required Exercises	
01	Hide Client Side Business Logic

EXERCISE 01: Hide Client Side Business Logic (Required)

Scenario: An extra coupon code validation is to be implemented. The only valid coupon code is ABC12. SuiteDreams has the following requirements surrounding this coupon code validation:

- End users must have immediate validation feedback in the same way as the current coupon code validation regarding its length
- The set of valid coupon codes is not to be exposed in the browser (i.e. in the html of the web page). This is for security reasons.

The solution to SuiteDream's requirements is to embed the business logic inside of a RESTlet and then call the RESTlet from the client side script. The solution is implemented in this exercise.

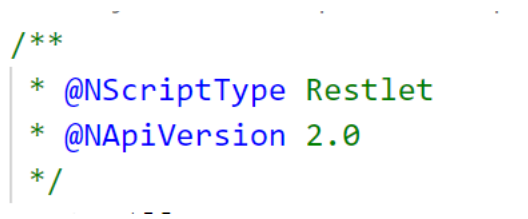
DID YOU KNOW?

This exercise illustrates an important use case. There are some instances where you want to hide values from users who may be familiar with programming. By moving the validation to the server side, actual values are hidden even if the user tries to examine the code from the browser debugger.

Create RESTlet

1 Create a RESTlet script and name it **sdr_rl_coupon_code.js**.

2 Add the following JSDoc annotations at the top of your script:



```
/**
 * @NScriptType Restlet
 * @NApiVersion 2.0
 */
```

3. Create a `get()` function and create a variable that would accept a coupon code value from the url parameter. Assume that the parameter is named **custparam_couponcode**.

   Note: URL parameters in RESTlets are passed directly to the function. Unlike suitelets where parameters are extracted from the request object, RESTlets would take in parameters with this format `requestParams.myParameter`.

4. Add a condition that would return the string "valid" if the coupon code is equals to **ABC12** and "invalid" if it's not.

5. Upload the script and create a script record.

Modify Client Script

6. Go to your customer client script and add the **https** and **url** modules as dependencies.

7. Comment out the if statement that validated the coupon code.

   Note: Use the `saveRecord` entry point if you were not able to do the validation on the `validateField` entry point.

8. Add an if statement to replace the commented out validation so that it triggers only when the APPLY COUPON CODE is checked and the COUPON CODE field is populated.

9. Inside the if statement, get the RESTlet URL using the url modules `resolveScript()` method.

   Note: Pass the `scriptId` and `deploymentId` as parameters to dynamically get the RESTlet URL using `resolveScript()`.

BEST PRACTICES

While you can use hardcoded RESTlet and suitelet URLs from your script, it is bad practice to do so. These URLs change when transferred from one account to another (i.e., sandbox to production). Hardcoding URLs of any kind is not future proof and should be avoided.

10. Call the RESTlet by using the https module's `get()` method. To pass parameters simply append the parameter value to the URL. Use the name custparam_couponcode.

Note: Get requests use values passes values through the URL. To do this add, an ampersand and the key/value pair at the end of the URL. For example, url +"&key=value".

11 The get method will return a response object. You can get the returned string from the RESTlet from the **body** property of the response.

12 Add a condition that validates the response. If the response is invalid, then display an alert message so the user is aware that the coupon code is invalid.

IMPORTANT CONSIDERATIONS

Module Overview

Before closing the course out, it’s there are a few important considerations that we need to think of before, during and after development.

In this module we’ll cover the SuiteScript development life cycle; what SuiteScript governance is; and how to optimize your code by using the Application Performance Management tool. Lastly we’re going to look at packaging your customizations up using the SuiteBundler tool.

Module Objectives

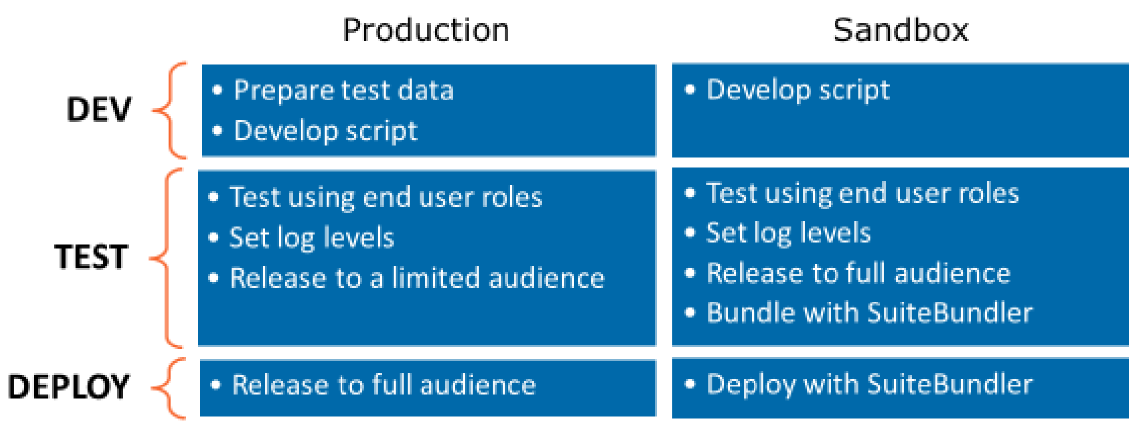
Upon completion of this module, you will be able to:

- Understand the recommended development process
- Handle SuiteScript error objects
- Develop with governance in mind
- Optimize code with APM
- Package customizations using SuiteBundler

SuiteScript Development Life Cycle

While the development life cycle is standard across different programming languages, there are a few things that you need to consider with developing SuiteScript projects. One of those considerations is where you'll be developing. Developing directly on a production account, something that happens often, requires more consideration compared to developing on a sandbox account.

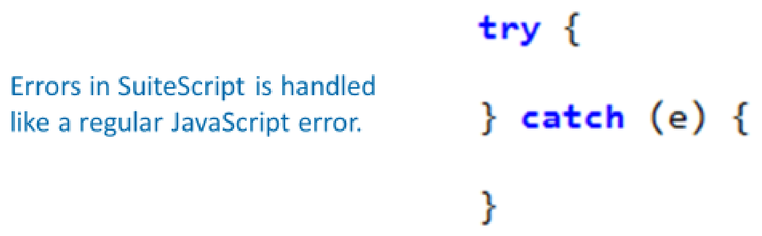
Note: Sandbox accounts are purchased separately. For more information about it, please talk to your local NetSuite sales representative.



	Production	Sandbox
DEV	• Prepare test data • Develop script	• Develop script
TEST	• Test using end user roles • Set log levels • Release to a limited audience	• Test using end user roles • Set log levels • Release to full audience • Bundle with SuiteBundler
DEPLOY	• Release to full audience	• Deploy with SuiteBundler

Error Handling

Handling errors in NetSuite is the same as any other JavaScript API; exceptions are caught and handled using the try/catch block.



Errors in SuiteScript is handled like a regular JavaScript error.

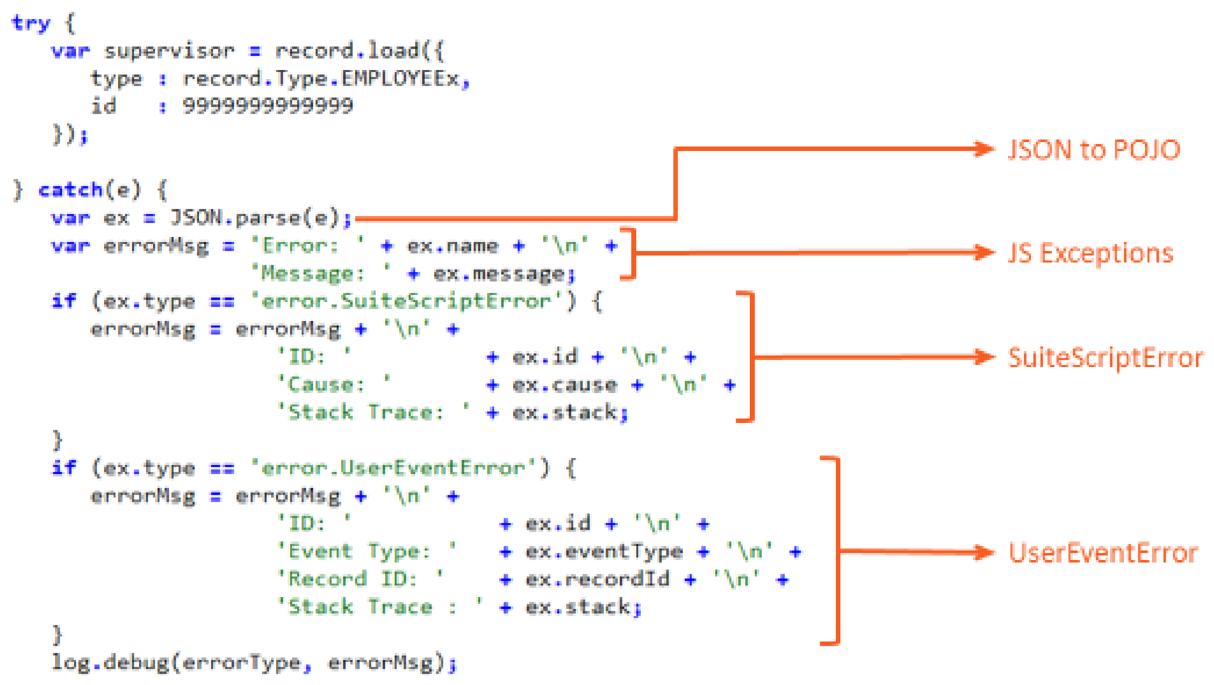
```
try {

} catch (e) {

}
```

Exceptions in NetSuite are returned as JSON strings. The string should then be converted into a plain JavaScript Object (POJO) to easily extract the values off it.

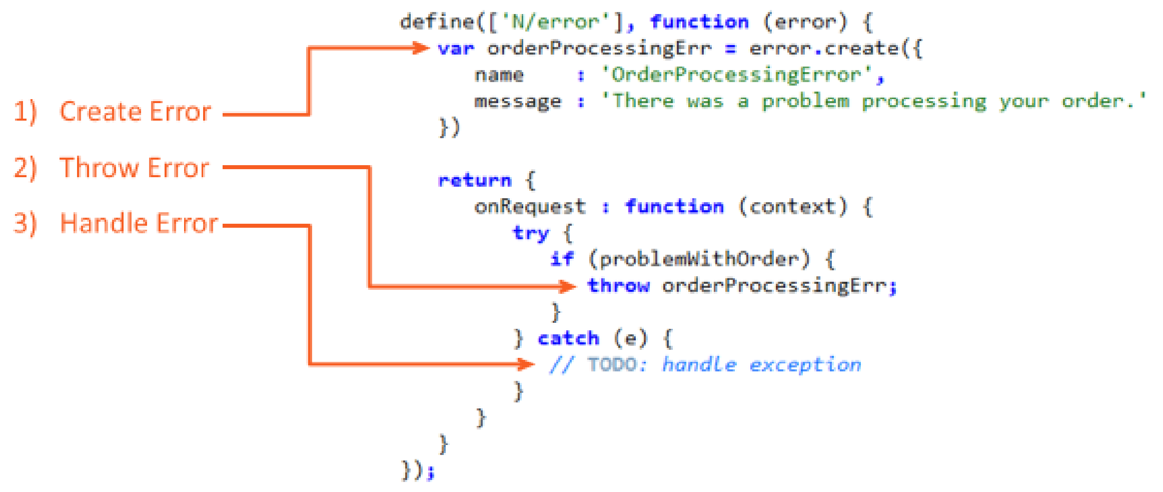
There are three possible exception objects in SuiteScript: standard exceptions, SuiteScript errors, and UserEvent errors. All these error objects have the standard name and message properties. SuiteScript errors have additional properties such as the cause of the error and the stack trace. UserEvent errors have even more information that related to which record caused the error and which event the error happened.



```
try {
   var supervisor = record.load({
      type : record.Type.EMPLOYEEx,
      id   : 9999999999999
   });

} catch(e) {
   var ex = JSON.parse(e);
   var errorMsg = 'Error: ' + ex.name + '\n' +
                  'Message: ' + ex.message;
   if (ex.type == 'error.SuiteScriptError') {
      errorMsg = errorMsg + '\n' +
                 'ID: '           + ex.id + '\n' +
                 'Cause: '        + ex.cause + '\n' +
                 'Stack Trace: '  + ex.stack;
   }
   if (ex.type == 'error.UserEventError') {
      errorMsg = errorMsg + '\n' +
                 'ID: '            + ex.id + '\n' +
                 'Event Type: '    + ex.eventType + '\n' +
                 'Record ID: '     + ex.recordId + '\n' +
                 'Stack Trace : '  + ex.stack;
   }
   log.debug(errorType, errorMsg);
```

Developers can also create their own error handling system by creating custom error objects.



```
define(['N/error'], function (error) {
   var orderProcessingErr = error.create({
      name    : 'OrderProcessingError',
      message : 'There was a problem processing your order.'
   })

   return {
      onRequest : function (context) {
         try {
            if (problemWithOrder) {
               throw orderProcessingErr;
            }
         } catch (e) {
            // TODO: handle exception
         }
      }
   }
});
```

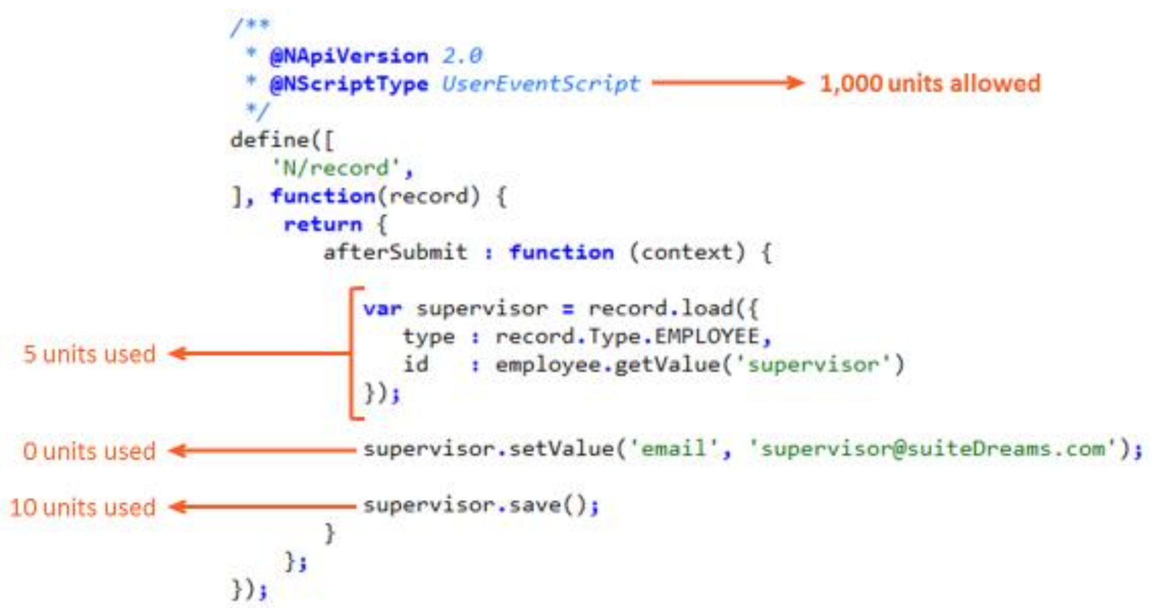
SuiteScript Governance

Another important thing to remember when developing SuiteScript is script governance. So what is script governance? NetSuite regulates the execution of scripts to make sure that no accounts would use up the resources of a server, intentional or accidental. The server does this by allocating a specific number of units to a script type. A user event script, for example, is assigned 1,000 units and a scheduled script with 10,000 units.

Specific SuiteScript statements "cost" a certain number of units per execution. For example, a `record.load()` call to get an employee record costs 5 units and saving changes made to the record would cost 10 units.

DID YOU KNOW?

SuiteScript calls vary depending on the record type. The same statement, `record.load()`, used for get a transaction record would use up 10 units instead of 5 and loading a custom record would use 2.



```
/**
 * @NApiVersion 2.0
 * @NScriptType UserEventScript
 */
define([
    'N/record',
], function(record) {
    return {
        afterSubmit : function (context) {

            var supervisor = record.load({
                type : record.Type.EMPLOYEE,
                id   : employee.getValue('supervisor')
            });
            supervisor.setValue('email', 'supervisor@suiteDreams.com');

            supervisor.save();
        }
    };
});
```

IMPORTANT | CAUTION

Scripts that run out of units during execution will automatically be stopped by the system. It is very important that you are aware of this and always compute for the number of units that your script consumes.

Also consider the amount of records that your script processes. Once the script is deployed to production, your script would most likely consume more units that you might have expected.

Optimizing Unit Usage

There are several ways of optimizing the script to make sure that it doesn’t run out of governance units. Here are a few examples:

- **Use** `record.submitFields()` **instead of** `record.load()` **+** `record.save()`

  To update fields on a record, it's best to use the `record.submitFields()` method instead of the `record.load()` and `record.save()` combination. The submitFields method use 2 units while the load and save method use 15 units.

- **Use** `search.lookupFields()` **instead of** `ResultSet.each()`

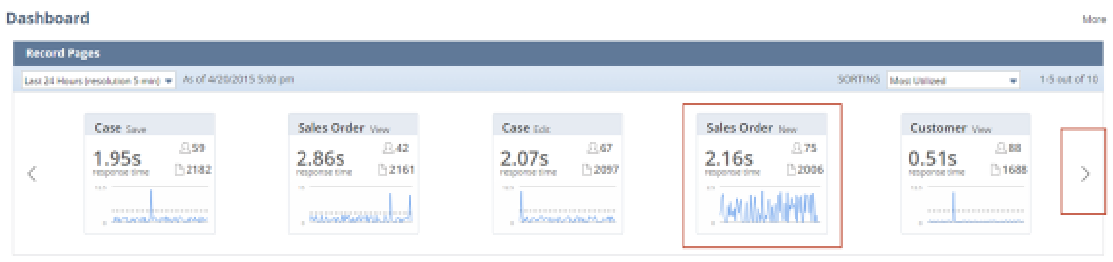
  Similar to the `record.submitFields()` record, the `search.lookupFields()` only cost 1 unit per execution while the `ResultSet.each()` costs 10.

- **Offload processing to scheduled or map/reduce scripts**

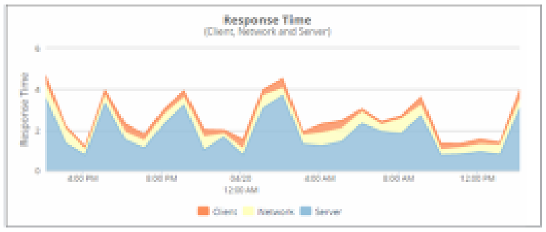
  One of the most common way to optimize the unit usage is to push processing to a scheduled script or a map/reduce script. Unlike a user event script which only has 1,000 units a scheduled script has 10,000 units. Map/reduce scripts automatically handle governance so it's easy to process large amounts of data using that script type.

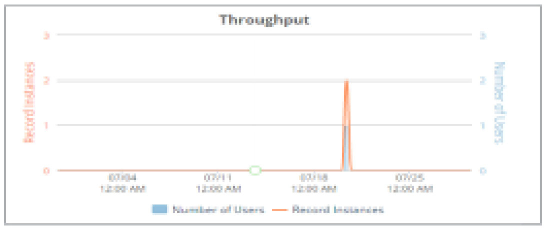
Application Performance Management

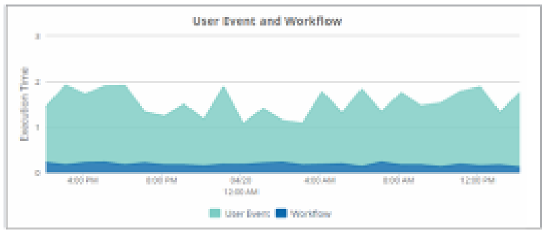
To further optimize the script execution, it's important to gather data on how a record performs with the scripts deployed to it. That way developers would know if the records are running significantly slower and would need optimization.

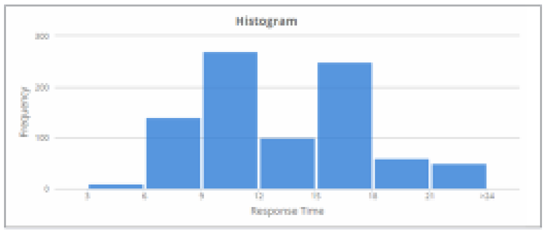


There are four graphs in the APM to measure the response time, throughput, user event and workflow execution times, and histogram of the response time.









The **Response Time** graph displays how much time is spent processing the record on the **client** (browser execution time), the **network** (load time from client to the server), and server (time spent processing the record). This is useful in determining where the slowdown is happening. Optimizing the script might not help if the slowdown is caused by a slow computer or a slow internet connection.

The **Throughput** graph displays the number of record instances and the number of users over a period of time. This allows you to determine how many records are accessed by how many people. Start optimizing the high volume records to provide the most impact.

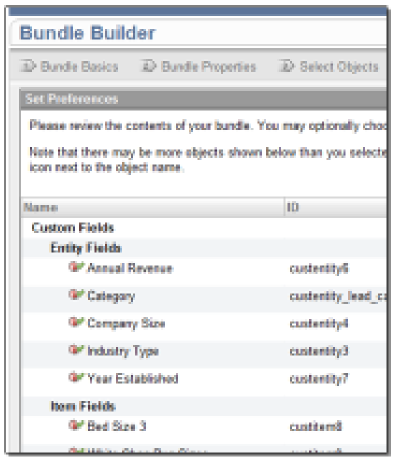
The **User Event and Workflow** graph shows the time it takes to execute user event scripts and server workflows on a record.

Lastly, the **Histogram** shows the record instances grouped by response time. This helps developers know if the high response time was only caused by an anomaly.

SuiteBundler

SuiteBundler is a tool used to transfer customizations from one account to another. It can be used to package up a variety of customizations including scripts, custom fields, custom records, searches, etc.

SuiteBundler is particularly useful for people who need to create customizations for their own customers or transfer customizations from their sandbox account to their production account.



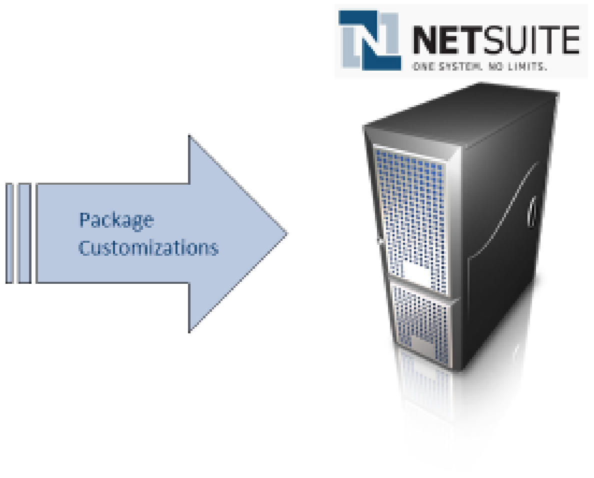
Bundle Builder

Set Preferences

Please review the contents of your bundle. You may optionally choo

Note that there may be more objects shown below than you selecte
icon next to the object name.

Name	ID
Custom Fields	
Entity Fields	
Annual Revenue	custentity6
Category	custentity_lead_c
Company Size	custentity4
Industry Type	custentity3
Year Established	custentity7
Item Fields	
Bed Size 3	custitem8



Customization > SuiteBundler > Create Bundle

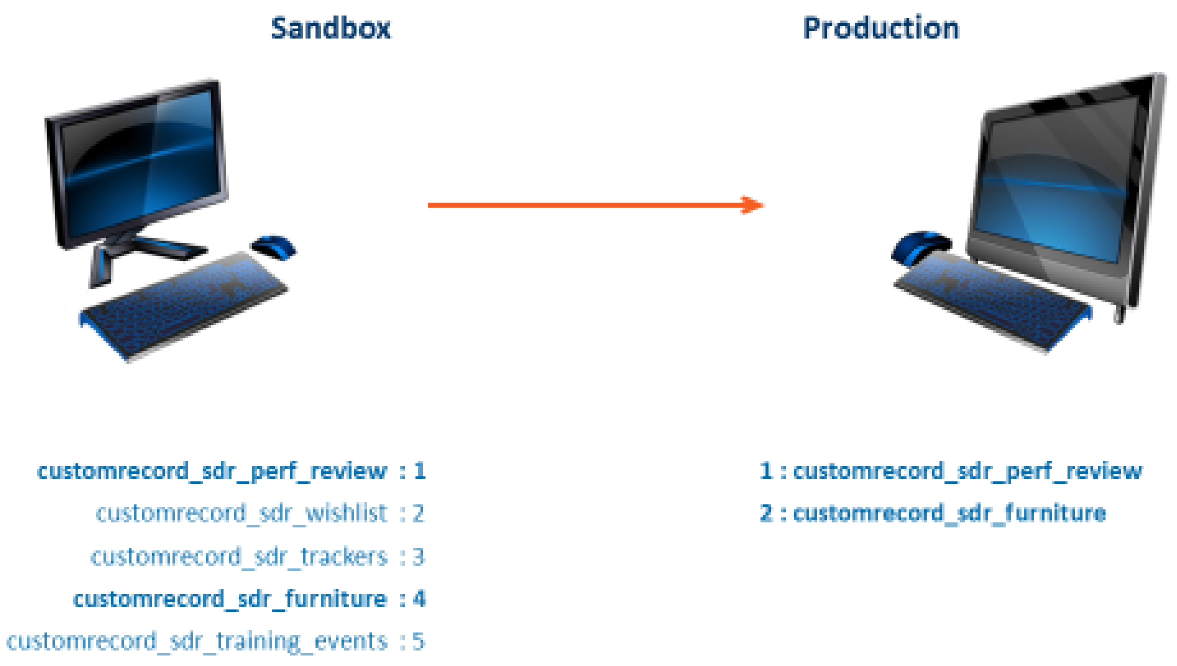
BEST PRACTICES

The SuiteBundler tool is a very effective tool for SuiteApp creation and distribution. Developers can add a Terms of Service page that users must agree on before installing a bundle. Documentation can also be attached to the bundle so admin and users would know how to use the bundle they installed. Customizations packed up in a bundle can also be locked so to hide the implementation code, protecting valuable intellectual property.

There's more to the SuiteBundler tool than just packaging up customizations so it's highly recommended that you look into and practice using the tool more.

Accounts and Internal ID

When transferring customizations using the SuiteBundler tool or doing it manually, it's important to take note that internal IDs will change. This can potentially break your customization if internal ids are hardcoded.


Sandbox
Production
customrecord_sdr_perf_review : 1
customrecord_sdr_wishlist : 2
customrecord_sdr_trackers : 3
customrecord_sdr_furniture : 4
customrecord_sdr_training_events : 5
1 : customrecord_sdr_perf_review
2 : customrecord_sdr_furniture

IMPORTANT

When creating SuiteScripts, or any application for that matter, it's never a good idea to hardcode values that can potentially change. If there's a way to get values, like internal ids, dynamically then it should be done.

Another alternative is to use the script id instead of the internal id as the script id will not change unless it is explicitly changed by an admin.

APPENDIX A | JAVASCRIPT SYNTAX

SuiteScript API-related Syntax

The following are syntax guides for those who are new to programming in JavaScript and the SuiteScript API.

Note: This course assumes that the learner is already familiar with JavaScript programming and will not go into the intricacies of the language. The guide is only covers a high-level overview of the syntaxes.

JavaScript Objects

One thing you have to understand in JavaScript is that everything are objects. An object is an object, an array is an object, even a function is an object. While you wouldn't need to worry about that in this course, it's a good thing to keep that concept in mind whenever you're developing.

With SuiteScript 2.0, you'll be using anonymous object for several things so it's important that you know how to create objects. The quick way to create an object is:

```
var object = {
   property1 : value1,
   property2 : value2
   // and so on
};
```

Properties in JavaScript is the same as any object-oriented programming language. Values on the other hand, can be anything. It can be any simple values like numbers or strings, or it can be functions. In fact, this is how you're going to define a function in SS 2.0. From your define statement, you'll be returning an object with property names based on the event you want to trigger.

Using the Define Function

All scripts in NetSuite are triggered from an entry point in your `define` function. The syntax for the define statement is:

```
define(function () {
   return {
      entryPoint : function (context) {
         // Do something
      }
   }
});
```

When using the define statement, make sure to use the right name based on the entry point that you want to trigger. If a wrong entry point name is used, it will be not be triggered or may cause errors in some cases.

TIPS AND TRICKS

You can also add statements and function before your return statement. This allows you to create functions that can be used across the multiple entry points.

Variable Naming Convention

JavaScript is a loosely-typed language. This means that variables created in your script is not given a type. For people coming from typed languages like C# or Java, this might be confusing since you can't quickly determine the type of a variable. Some developers deal with this using the Hungarian Notation. This naming convention prefixes the data type before the variable name. Here are a few examples:

- intTotal (integer)
- stName (string)
- bFlagged (boolean)
- recSalesOrder (record object)

The JavaScript development community is divided regarding this issue with most favoring the regular variable naming convention. Whatever convention you use though; you need to make sure that you stick to it so that you maintain consistency. Inconsistent naming conventions tend to make the code harder to read and therefore harder to maintain.

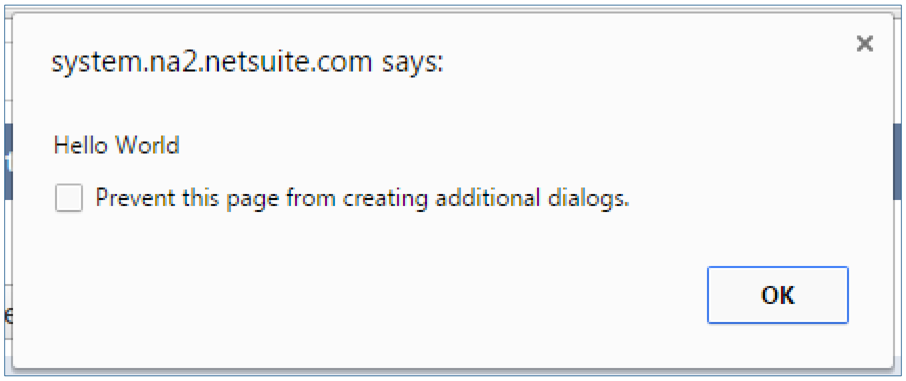
Client Notifications

There are two statements you can use to display a message to the user in the client side, the alert and confirm statements.

alert()

The alert() statement is used to display a simple message on the screen. This is useful if you want to display information on the screen that the user doesn't need to take action on.

```
alert('message');
```



system.na2.netsuite.com says:

Hello World

Prevent this page from creating additional dialogs.

DID YOU KNOW?

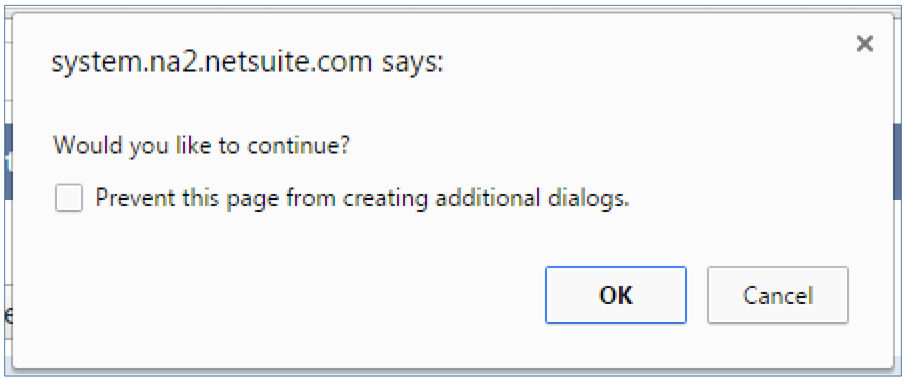
Strings in JavaScript can be defined using single or double quotes. What you use depends on your team's coding preferences. Just make sure that you don't mix and match the two. If you use single quotes, make sure that you use single quotes for all your scripts.

Note: The alert statement only works on client-side scripts.

confirm()

The confirm() statement is also displays information on the screen, similar to the alert() statement, but it also asks the user to take an action.

```
confirm('Would you like to continue?');
```



system.na2.netsuite.com says:

Would you like to continue?

Prevent this page from creating additional dialogs.

These messages display two button, Ok and Cancel. Clicking Ok will return a true and a Cancel will return false. This is perfect for instances where you want to ask your users if they want to continue.

Truthy & Falsey Checks

In JavaScript, the values 0, “” (empty string), null, undefined, NaN, and false are considered false values. An easy way to check for those values is to pass them as your condition. Here’s an example:

```
if (!someValue) {
   // do something if someValue is falsey
}
```

This statement is similar to:

```
         if (someValue == 0 || someValue == "" || someValue == null ||
someValue == undefined || someValue == NaN) {
            // do something
         }
```

Take note of the ! (not operator) in the first example. Without this, the if statement will execute if the value is not falsey; which is fine if that’s what you need.

For Loop

The for loop syntax in JavaScript is practically the same as other object oriented programming languages such as Java or C#.

```
for (var int = 0; int < array.length; int++) {

}
```

The `int` variable represents your counter and the `array.length` tells the loop how many times it would repeat.

Loading SuiteScript Modules

To load SuiteScript modules, just add an array of strings as a first parameter to your require or define function (before your main function definition). Then add a variable on your main function definition as a parameter to hold module object. For example to load the record module, use:

```
define(['N/record'], function (record) {
   // entry points here
});
```

The N\record is the path to the module that you want to load and the **`record`** parameter is that module object that you’ll be using in your script. To load another module, just add another module path and another parameter to hold your module object. So to add the email module, you’ll:

```
define(['N/record', 'N/email'], function (record, email) {;
   // entry points here
});
```

Note: The order of variables should correspond to the order of module paths that you've added. If you've loaded the record module path first, your variable for the record module object should be declared first and so on.

Module Annotations

Additional annotations need to be added to added to the definitions to make content-assist work. The syntax for it is:

```
define(['N/record'],
/**
 * @param {record} record
 */
function(record) {
```

The first value defines the name of the module that you want to load without the 'N/'. To load the N/email module, it will be {email}. The next value refers to the name you're using to store the module. Typically, module variables use the same name as the module.

Search Expressions

Search expressions allow developers to quickly define search filters and columns when creating a script search. To use search expressions for search filters, use:

```
[[<fieldid>, <operatorEnum>, <value>], '<and/or>',
 [<fieldid>, <operatorEnum>, <value>]
]
```

Here's an example of how that's used:

```
[['type', search.Operator.ANYOF, 'SalesOrd'], 'and',
 ['mainline', search.Operator.IS, true]
]
```

For search columns, just create an array of search column ids.

```
['entity', 'type', 'total']
```