NETSUITE | SuiteTraining

NetSuite Certification: SuiteCloud Developer Exam Preparation

Student Guide

NetSuite Inc.
2955 Campus Drive, Suite 100
San Mateo, CA 99403-2511

TABLE OF CONTENTS

PREFACE

About Your NetSuite Training Account

As a student in a SuiteTraining course, you are provisioned with a NetSuite Training account which is used to complete all the hands-on exercises provided in each course.

This training account is available to you for the next thirty (30) days. Please note that it may or may not have the same functionality as your current production NetSuite account.

Occasionally, the material presented may be different from the data provided in the training account used for all class exercises.

Requesting Instructor Support

Your instructor is happy to answer any specific questions you may have about the course, its content, and/or the course materials.

While we encourage all participants to ask pertinent questions, we also want to ensure the course moves along at an acceptable pace for all student participants. Questions or requests for assistance regarding your organization's NetSuite implementation are outside the scope of what instructors are able to provide during class.

For questions or assistance with your NetSuite implementation, please contact NetSuite Support or your account manager.

Module 01 | Session Introduction

MODULE 01 | SESSION INTRODUCTION

Session Overview

This module provides an overview of the *SuiteCloud Developer Exam Preparation* session.

Topics discussed in this module:

- About This Study Session
- Study Session Audience
- Prerequisite Knowledge
- Session Goals
- Session Objectives
- Session Agenda
- Student Exercises
- Tips for Success

About This Study Session

This study session is intended to act as final preparation for the SuiteCloud Developer Exam. In this session you can investigate a variety of pre-built scripts, workflows, and integration code that are tied to the exam objectives. You can also read about tips, tricks, and best practices related to these objectives. In many cases this information is documented inside of the scripts and integration code.

Study Session Audience

The target audience for this session includes candidates qualified to take the SuiteCloud Developer Exam. A description of a qualified candidate can be found on the NetSuite certification page: http://www.netsuite.com/portal/services/training/certification.shtml.

Prerequisite Knowledge

Given this session is focused on candidates qualified to take the SuiteCloud Developer Exam, students are expected to have working knowledge on creating scripts, workflows, and integrations.

Session Goals

The goal of this session is to provide you with a review of SuiteCloud Developer Exam objectives via pre-built scripts, workflows, and integration code.

Session Objectives

This session provides a review of SuiteCloud Developer Exam objectives. There are 51 exam objectives across seven subject areas, with code samples tied to many of them. At the time of this writing the 22 most missed exam objectives have been identified. Instructor review will focus on the most missed

exam objectives, but you are urged to ask questions related to any of the objectives. The most missed objectives are identified within each module where the objectives are covered.

Upon completion of this exam study session, you will have reviewed exam objectives across each subject area of the exam:

- SuiteScript
- SuiteFlow
- SuiteTalk
- SuiteBundler
- SuiteBuilder
- SuiteAnalytics
- Design Fundamentals

Session Agenda

Agenda
Module 01: Course Introduction
Module 02: Review SuiteScript Exam Objectives
Module 03: Review SuiteFlow Exam Objectives
Module 04: Review SuiteTalk Exam Objectives
Module 05: Review SuiteBundler Exam Objectives
Module 06: Review SuiteBuilder Exam Objectives
Module 07: Review SuiteAnalytics Exam Objectives
Module 08: Review Design Fundamentals Exam Objectives
Module 09: Taking the Exam

Student Exercises

This course is not about gaining completely new skills across each of the exam subject matter areas. NetSuite offers courses to provide you training in exam subject matter areas you are unfamiliar with. For example, 5-day courses are available on SuiteScript and SuiteTalk, as well as courses covering SuiteFlow. The exam study guide identifies recommended training courses. These training courses come with many in-depth hands-on exercises.

This course does not have in-depth hands-on exercises. Since it is expected you already have skills across exam subject matter, the approach in this course is to give you prebuilt code samples. The code samples allow you to review skills as related to the exam objectives. Documented code samples across SuiteScript, SuiteFlow, and SuiteTalk are included in your training account. You will get an opportunity to run some of these samples during class, but you can also run these outside of class.

Tips for Success

Here are some tips to ensure you get the most out of this course:

- Ask questions…lots of them! This is your opportunity to ask questions related to the exam objectives.
- Please be patient and consider other students.
- Do not email or browse the web during review of exam objectives.
- Turn mobile devices to silent mode.
- Return on time from scheduled breaks.
- Have fun!

MODULE EXERCISES

Exercises	
01	Logging into your NetSuite Training Account

EXERCISE 01: Logging into your NetSuite Training Account

Scenario: In this hands-on exercise, you will login to your NetSuite training account using the login information provided by your instructor.

Logging into your Training Account

1 Open a browser and navigate to *www.netsuite.com*.

If this is the first time you are using your browser to log in to NetSuite, then you will need to click on the **Login** link toward the top of the page.

2 Skip to the next step if you already have a page that allows you to enter an Email Address and Password.

3 Fill in the Email Address and Password provided by your instructor. The initial password is usually *training1* (this must be entered in all lowercase characters).

DO NOT CHANGE YOUR PASSWORD!

Your instructor will be unable to assist you if you change the password and forget it. Your instructor will be unable to reset your password if you forget it.

4 Click the **Login** button.

Entering Security Questions

5 You may select **Remind Me Later**, but you can only do this a maximum of 5 times.

Use the following standard set of answers:

What was your childhood nickname?	nickname
In what city did you meet your spouse/significant other?	other
What is your maternal grandmother's maiden name?	name

DO NOT CHANGE THE SECURITY QUESTIONS!

If your instructor logs into your account from a different browser in order to assist you, your instructor will be prompted to answer the specified security questions above.

If you change these security questions and subsequently forget what you entered as answers, your instructor will not be able to login and assist you. If everyone enters the same answers, your instructor will be able to log in and assist, as necessary.

However, you may change the answers to the specified security questions at any time by selecting Update Security Questions from the Settings portlet on your home dashboard.

Accessing your Training Account as an Administrator

6 A message page will be displayed requiring you to click a checkbox to confirm you have read the message. A "Getting Started" message page always displays, but there may be others as well. Click the checkbox and then **Continue**.

7 Upon successful login, you will be directed to the **Home Dashboard**. Note that you are logged in as "Larry Nelson (Administrator)". The user and role you have logged in with will display in the upper right-hand corner of the page.

You are now ready to continue with the class.

Module 02 | Review SuiteScript Exam Objectives

MODULE 02 | REVIEW SUITESCRIPT EXAM OBJECTIVES

Module Overview

This module provides a review of SuiteScript subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the SuiteScript subject matter. The most missed objectives are identified with an asterisk (*) in the title.

Working with SuiteScripts in your training account

There are scripts associated with many of the the objectives in this and later modules. Each objective identifies related script records as applicable. All of the User Event and Client scripts in your account are undeployed by default. You can redeploy User Event and Client scripts by opening up the list of script deployments at **Customization > Scripting > Script Deployments**. Check the SHOW UNDEPLOYED box, and then find your script in the SCRIPT dropdown. Check DEPLOYED beside the script deployment and click **Submit.** It is recommended to undeploy User Event and Client scripts after testing as there could be conflicts with other scripts and workflows in the account.

Other script types such as Portlets and Suitelets are already deployed, and this is okay.

Some scripts are not associated with a script record. These contain **NoScript** in the name and can be found in the file cabinet **SuiteScripts > Developer Certification** folder. These scripts can be tested by copying and pasting into the script debugger.

There is a two character abbreviation in the name of each script record and associated script file that identifies the script type. These are the abbreviations:

- UE (User Event)
- SL (Suitelet)
- PL (Portlet)
- CS (Client)
- BI (Bundle Installation)
- SC (Scheduled)
- WA (Workflow Action)
- RL (RESTlet)

Select the SuiteScript code snippet that implements a described business process

You'll need to be able to visually inspect a set of code and see if it matches a scenario.

Check script record **SuiteDreams UE Describe Bus Process**. Change the AFTER SUBMIT FUNCTION based on the option you are testing. See script file for details. Some invalid options may generate exceptions.

Recognize the purpose of Plug-Ins in SuiteScript

Plug-ins allow you to define interfaces that can be overriden with multiple implementations, for example different ways to perform tax calculations. There are plug-ins that you can create yourself, and there are plug-ins created by NetSuite which you can build alternate implementations for. The ones from NetSuite are called Core Plug-ins. There are a couple Core Plug-ins as of this writing:

- Custom GL Lines Plug-in
- Email Capture Plug-in

The focus of the sample scripts are ones where you build both the interface and the implementation.

Plug-ins are defined at **Customization > Plug-ins > Custom Plug-in Types**. There is one sample plug-in named **SuiteDreams Number Manipulation**. Each plug-in is associated with a default implementation based on the DEFAULT IMPLEMENTATION field.

Each plug-in may have zero or more alternate implementations defined. These are found at **Customization > Plug-ins > Plug-in Implementations**. There is one sample alternate implementation named **SuiteDreams Number Manipulation Alt1**.

The CLASS NAME on the Custom Plug-In Type page for **SuiteDreams Number Manipulation** is named **NumberManipulation**. The default and alternate implementations of this class are instantiated in Portlet **SuiteDreams PL Plugins Number**. This Portlet is already set to display on the home dashboard. Make sure you read the notes at the top of the script file.

Identify the functionality and capabilities of SuiteScript debugger

The script debugger is accessed at **Customization > Scripting > Script Debugger**. You'll be prompted to log in to a specific debugger domain. Note the database is the same database as when you are logged in to the regular domain.

You can test two different pieces of sample code from file **SuiteDreams_NoScript_Script_For_Debugger.js**. Copy and paste each section of sample code into the

New Script window of the debugger and click **Debug Script**. The first code sample uses a saved search named **Get Orders for Script Debugger**.

Make sure to read all the debug tips at the top of the script file.

Given a scenario, identify the proper pattern to invoke a scheduled script to support a long-running process

Pattern: User Event script calls Scheduled script

Sometimes you may have long running processes in the After Submit of a User Event script. End users running User Event scripts must wait until completion of code in the After Submit function, as it always runs synchronously. The synchronous execution can negatively impact user experience as response time increases. To improve performance you can call a Scheduled script to perform the processing previously in the After Submit function. Scheduled scripts run asynchronously. As soon as a Scheduled script is called from a User Event script, control is returned to the script. This pattern works very well as long as the After Submit processing can be performed asynchronously. There is a sample User Event script calling a Scheduled script to highlight this pattern: User Event **SuiteDreams UE Invoke Sched Script** calls Scheduled script **SuiteDreams SC Invoked From Other Script** via **nlapiScheduleScript**.

Pattern: Suitelet script calls Scheduled script

Like with User Event scripts, a user interface Suitelet can offload processing from its POST section to a Scheduled script in exactly the same way. There is a sample Suitelet script calling a Scheduled script to highlight this pattern: Suitelet **SL Invoke Sched Script** calls the same Scheduled script as with the User Event sample, **SuiteDreams SC Invoked From Other Script**.

Pattern: Scheduled script re-queues itself

Sometimes a Scheduled script may require processing beyond its governance limit. You can re-queue another version of the same script to continue processing as needed. An easy way to do this is by yielding the script via **nlapiYieldScript**. See Scheduled script **SuiteDreams SC Requeue Using Yield** for an example. You can test this script by selecting **Save and Execute** on its Script Deployment.

Calculate the governance of a script

If you design scripts without taking governance into account, then you could end up with a situation where you exceed governance. A primary workaround when exceeding governance is to call a scheduled script, but you'll have a lot of work to do if you don't build this into your design up front. See objective "**Given a scenario, identify the proper pattern to invoke a scheduled script to support a long-running process**" for examples of moving code to a scheduled script.

This objective focuses on the calculation aspect. Unit consumption is different based on the kind of record you are loading, creating, and updating. Be aware about which APIs consume usage and which do not. You'll also want to be aware of limitations on a per script basis.

See sample Suitelet script **SuiteDreams SL Calc Governance** for an example of calculating unit usage.

Identify the implications of deploying multiple User Event scripts against a single record, order of execution, and how user events interact with other records in SuiteCloud technologies

Each User Event script runs synchronously. If there are multiple User Event scripts deployed to a single record, one finishes its execution followed by the next and so on. If some of your User Event scripts have dependencies with one another, you can adjust their execution order at **Customization > Scripting > Scripted Records**. If you are a SuiteApp developer, then you should understand that your User Event scripts could be one of many executing on the same record in an account. Adjusting the order of script execution may need to be included in your script documentation.

You can take a look at 3 sample User Event scripts, all deployed to the vendor record:

- **SuiteDreams UE Vendor 1**
- **SuiteDreams UE Vendor 2**
- **SuiteDreams UE Vendor 3**

Note that when you save a vendor record, all Before Submit events execute first across the 3 scripts, followed by all After Submit events.

A primary restriction on User Event scripts is they cannot call other User Event scripts. This helps to keep from having infinite loop scenarios. User Event scripts can load, create, and update other records. If User Event scripts are deployed to these other record, they do not execute. They will execute when other script types load, create, and update other records (e.g. Suitelets, Portlets, RESTlets, etc.), just not from User Event scripts.

Identify the risks of implementing only client-side validations and strategies to address them

Risks of implementing only client-side validations:

- Though unlikely, there is the potential a client-side validation may behave differently across different browsers and their versions.
- You may be exposing algorithms that should not be made public. Remember that any code sent to the browser can be seen if someone views the page source.

Strategies to address the risks:

- Duplicate client-side validation inside the Before Submit event of a User Event script.
- Hide the validation algorithm inside of a Suitelet. Call the Suitelet from the client-side. This makes an AJAX call, so the user experience is seamless.

SuiteDreams CS Client Side Validations contains two options for implementing a validation:

- Option 1 shows how the validation algorithm is contained within the Client script.
- Option 2 shows how the validation algorithm is moved to a Suitelet script. The Suitelet script is **SuiteDreams SL Client Side Validations**.

SuiteDreams UE Client Side Validations shows an example of duplicating the validation in the Before Submit of a User Event script. The validation is slightly different than in the Client script and Suitelet script, so you can test all validations without changing code.

Identify how to dynamically scale a scheduled script utilization to match available queues in account

By default there is one scheduled script queue, but with SuiteCloud Plus you can purchase licenses that give you either 5, 10, or 15 queues. Your training accounts have 3 SuiteCloud Plus licenses, giving 15 queues.

SuiteCloud Plus allows you to have multiple scheduled scripts running at the same time, one per queue. If you have large scheduled script processes (e.g. process 100,000 records), you can set up an algorithm that allows you to break up the processing across multiple queues, running everything simultaneously. If you have multiple queues, then it is a good idea to learn how to utilize all of the queues, so you don't end up in a situation where some of the queues sit idle when you have multiple scripts pending in other queues.

Suitelet **SuiteDreams SL Available Queues** contains a simple algorithm that utilizes all scheduled script queues in the account to process groups of support cases. See the script file for additional documentation.

SuiteDreams SC Available Queues is the scheduled script that is called by the Suitelet. The Scheduled script has 15 deployment records, one for each queue.

Both scripts make use of saved search **Get Support Cases**. Dynamic filters are added in the script where applicable.

Recognize script deployment configuration across script types

It is very important to understand the deployment options across script types. User Event script **SuiteDreams UE Script Deploy Config** can assist with understanding the deployment options. Read through the documentation in the script file, as it takes you through changing several options and seeing the resultant behavior. The documentation instructs you to create a new employee record with login access. To keep from having email address conflicts with your fellow students, it is recommended the domain be the same as your account id, e.g. **someemployee@TSTDRV111222.com**

Determine which variables would be good candidates to use as a company preference, user preference, or Script Deployment Parameter

Script parameters can help to remove hardcoding of values in your script, allowing end users and administrators to manage values that can change. There are three kinds of script parameters identified by the PREFERENCE setting on the script field:

- Company
  - The value is global for the entire organization. The value can be changed at **Setup > Company > General Preferences**.
- User
  - The value is specific to each NetSuite user and is changeable at **Home > Set Preferences**.
- <blank>
  - No selection for PREFERENCE means the script parameter value can be set on the script deployment. This is great when you have multiple deployments for a script, and need to change a value across deployments.

User Event script **SuiteDreams UE Script Parameters** takes you through a couple script parameter scenarios.

Identify the impact of execution context on user event scripts

User Event scripts can be executed from other script types (e.g. Portlet, Suitelet, Scheduled, etc.) as well as by other NetSuite technologies such as SuiteTalk integrations and CSV Imports. User Event scripts can even be executed by a SuiteCommerce web store.

Because User Event scripts can be executed from all these different contexts, it is very import to know how to filter execution to only those contexts where the script should run. Otherwise you may be unnecessarily impacting performance. For example, your SuiteTalk integrations and CSV Imports may take much longer to process than necessary.

As a good best practice, make sure to filter execution in your User Event scripts by checking the value of `nlapiGetContext().getExecutionContext()`.

SuiteDreams UE Execution Context is a User Event script that behaves slightly different when executed from a Suitelet script versus other contexts. You can execute from the user interface and then follow-up with execution from a Suitelet to see differences. Use Suitelet **SuiteDreams SL Execution Context** for your test. Make sure to read through the comments in both scripts.

*Determine how to interact with custom child record sublists in SuiteScript

Custom child record sublists allow you to have custom records be reflected as children on a parent record. In the user interface the child records display as a sublist on the parent record. You can optionally generate your custom child record sublist as an editable sublist on the parent record. When editable you can apply client-side and server-side SuiteScript to the sublist just as you can with standard editable sublists.

You can also use the capability of custom child record sublists to perform mass updates/creations of the child records with little governance usage. When a custom child record sublist is editable, any line items entered into the sublist are automatically saved when saving the parent record. This is default processing for custom child record sublists. Governance usage is calculated on the parent record only. There is no usage penalty for the sublist entries that invoke the creation and update of records. You can investigate this useful pattern with Suitelet script **SuiteDreams SL Cust Child Rec Sub Upd**. The Suitelet uses the following custom record types:

- **Perf Review Mass Updater**
  - This is the parent record type.
  - The Suitelet creates child records each time a parent record is created.
- **Performance Review**
  - This is the child record type.
  - The parent-child linkage is based on List/Record field **Perf Review Mass Updater**

Determine how to dynamically customize UI field attributes in SuiteScripts

Attributes of fields on NetSuite forms can be adjusted on the server, either during Before Load of User Event scripts, or during the GET processing of form-based Suitelets. In a User Event script you can gain access to an **nlobjField** via the **nlobjForm** parameter coming into the Before Load event. In a Suitelet, a reference to **nlobjField** is returned when creating the fields. You can gain access to **nlobjField** in Client scripts, but the object is read-only. In Client scripts there is field API **nlapiDisableField** that allows you to disable a field that is otherwise enterable.

Some things you can adjust via methods on **nlobjField**:

- Display type: hide or show fields, make field disabled or inline text
- Field labels
- Setting default values. Note that default values can only be set in Before Load on new records
- Marking whether field is mandatory or optional

User Event script **SuiteDreams UE Field Attributes** has an example of setting various **nlobjField** attributes during Before Load. Note that Before Load runs when creating, editing, copying, viewing, and printing a record, so field attribute adjustments can be filtered to these specific events using the event type parameter available to the function. Contrast with Client scripts that only run in edit mode.

*Identify the time zone implications of various ways of setting date and/or time values and the ways in which these values are interpreted in SuiteScript

This is the most missed objective on the SuiteCloud Developer Exam. Take some time to gain an understanding of how date and time values are processed in SuiteScript in regards to time zones. A key thing regarding time zones is that date values created in server scripts are always in the Pacific time zone. The data center containing your account does not alter this.

Two heavily documented scripts are available for you to become comfortable with time zone implications of date/time values. Both scripts make use of a custom record type named **Date Time Converter**. This custom record type contains two **Date/Time** fields. Field **User DateTime** is assigned a DYNAMIC DEFAULT of **Current Date/Time**. The other field, **Server DateTime** does not contain a DYNAMIC DEFAULT. The two scripts:

- User Event script **SuiteDreams UE Timezone Implications**
- Scheduled script **SuiteDreams SC Timezone Implications**
  - To test, set the deployment STATUS to **Scheduled**, setting it to execute as a SINGLE EVENT at the next half hour increment in the future. The date and time set for scheduled script execution is based on the TIME ZONE user preference setting of the script OWNER.
  - Once the script runs, the deployment STATUS is automatically reset to **Not Scheduled**.

Identify the capabilities of UI Objects

UI objects are quite powerful and mirror most of the capabilities you see on standard record forms. For example you can create buttons, fields, field groups, subtabs, and sublists, all things that are on standard record forms such as sales orders and customers.

You can dynamically create additional UI elements in the Before Load of a User Event script, but you can also create them when building form-based Suitelets. Take a look at Suitelet **SuiteDreams SL UI Objects** to see an example of generating a form that has the above mentioned elements.

Module 03 | Review SuiteFlow Exam Objectives

MODULE 03 | REVIEW SUITEFLOW EXAM OBJECTIVES

Module Overview

This module provides a review of SuiteFlow subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the SuiteFlow subject matter. The most missed objectives are identified with an asterisk (*) in the title.

Working with workflows in your training account

There are workflows associated with many of the objectives in this module. Each objective identifies related workflows as applicable. All of the workflows in your account have their RELEASE STATUS set to Not Running by default. Open a workflow and change its RELEASE STATUS to Testing or Released to run it. Make sure to check ENABLE LOGGING so you can view the workflow log.

*Identify how to configure standard actions and define custom actions

Your workflows are nothing without actions. Actions can be standard or custom.

About standard actions:

- You can choose from a standard set of actions such as **Add Button**, **Confirm**, **Create Record**, and **Go To Page**. There are many others.
- Some standard actions are available in client triggers only. For example, **Confirm** and **Show Message** only run in the browser.
- Some standard actions can only be executed at certain trigger points. For example, **Send Email** cannot be executed during Before Record Submit. An alternative is to place the action in After Record Submit.
- Some standard actions have different behavior depending on their triggers. For example, **Return User Error** at Before User Submit pops up a dialog box in the browser. **Return User Error** at Before Record Submit generates an entire error page.
- Some standard actions are only available with certain record types. **Transform Record** is only available on a subset of record types that can have copies created in related record types, for example, the ability to transform an Estimate into a Sales Order.

About custom actions:

- Custom actions are Workflow Action scripts. Once deployed, they can be configured inside of workflows just like standard actions.
- Script parameters define placeholders for values that can be passed into the script from the workflow.
- All custom actions can return a single value to the workflow.
- Workflow Action scripts have access to the current record (the record associated with the workflow instance) in the same way as a User Event script, via the `nlapiGetNewRecord()` statement. Custom actions are limited to server triggers.

Investigate workflow **SuiteDreams Standard Custom Actions** to see usage of a standard action and custom action. The related Workflow Action script is **SuiteDreams WA Standard Custom Actions**.

Identify how to control the User Interface throughout the workflow lifecycle

A NetSuite form can be dynamically changed through execution of workflow actions, either in client triggers or in server triggers. Some capabilities are similar to what you can do in SuiteScript. The capability to adjust the user interface at server triggers is similar to what you can do in a Before Load event of a User Event script. The capability to adjust the user interface at client triggers is similar to what you can do at various events in a Client script.

Actions that can impact the user interface at the Before Record Load server trigger:

- **Set Field Value** to set a default value on new forms
- **Set Field Display Type**
- **Set Field Display Label**
- **Add Button** to add a workflow button
- **Lock Record** to make a record read only
- **Remove Button** to remove a standard button from a form
- **Set Field Mandatory** to make a field required or optional

Actions that can impact the user interface during Before User Edit, Before Field Edit, After Field Edit, and After Field Sourcing client triggers:

- **Set Field Value** to set a default value on forms
- **Set Field Display Type**
- **Set Field Display Label**
- **Set Field Mandatory**

Actions impacting the user interface must be executed each time you want to change the user interface. The actions are not permanent. For example, **Lock Record** makes a record read-only. This

must be executed every time a record is loaded to continue keeping the record read-only. Consider a multi-state approval process. If a record should be read-only at each state during the approval process, you'll need to execute a **Lock Record** action in each of the states. To keep from duplicating action across multiple states, it is often recommended to create separate single-state workflows that only manage the user interface.

Investigate workflow **SuiteDreams Control UI (Client)** to look at execution of actions in client triggers. It is a single state workflow. DO NOT EXIT WORKFLOW is checked to keep workflow history clean so a new workflow is not starting up every time the record form is loaded into the browser.

Investigate workflow **SuiteDreams Control User Interface** to see actions in server triggers at multiple states during an approval process. DO NOT EXIT WORKFLOW is marked on the end states to not only keep workflow history clean, but to continue to execute user interface impacting actions at completion of the approval process when the record is approved or rejected.

*Describe the ways in which conditions and triggers can be configured to execute actions and transitions, and Identify capabilities and controls available with states and branching

The areas of SuiteFlow to be most careful with are the configuration of triggers during workflow initiation, actions, and transitions. Through misconfiguration you can inadvertently have workflows execute at different times than intended, and skip actions that should have executed. A few examples:

- A workflow directs an approval process which should start after saving a record. Instead you accidentally configure the workflow to start when a new form is loaded in the browser. Now you end up executing one or more states in your workflow before you even submit the form. Pay attention to TRIGGER TYPE on the Event Definition of the workflow!
- You click a button added with the **Add Button** action, but nothing happens. A transition should have been taken, but was not. Make sure you didn't configure a server trigger such as Before Record Submit along with the BUTTON configuration. Clicking **Save** generates Before Record Submit. Clicking a button added with **Add Button** action does not!
- Be careful of actions set to execute on Entry and Exit. These can be very useful, but if the Entry or Exit action is executing at a server trigger it doesn't support, then you'll see that it was **Skipped** in the Log! For example, **Go To Record** is only possible at After Record Submit. It is fine to execute this on Entry, as long as the workflow is running at After Record Submit when the state is entered.

Investigate workflow **SuiteDreams Cond Trigger Action Trans**. There are some issues with this workflow configuration. Can you figure out what some of the problems are and how to correct? The idea behind this workflow is that it coordinates an estimate approval process.

*Identify the functionality of record and workflow fields

There are record fields and two kinds of workflow fields. Here are some key points between the two categories of fields:

- Record fields
  - Fields are unique to the record. Record fields are what you see on any record you open, whether standard or custom. For example, EMAIL, PHONE, and JOB TITLE are all record fields on an employee record.
  - Recall you can have multiple workflows running on a single record. The value of a record field is the same across each workflow.
  - Values in record fields can be changed with the **Set Field Value** action. When **Set Field Value** is executed in a client trigger (e.g. After Field Edit) or at Before Record Load, the value is updated on the form only (there is no database persistence). When **Set Field Value** is executed at Entry, Exit, Before Record Submit, or After Record Submit, the value is persisted to the record in the database.
- Workflow fields
  - Fields are unique to a workflow.
  - Recall you can have multiple workflows running on a single record. The value of a workflow field in Workflow A is not seen in Workflow B. To get the value of a workflow field in Workflow A to be seen in Workflow B, you either need to set it into a record field or pass it as a parameter when calling Workflow B as a sub-workflow via the **Initiate Workflow** action.
  - Values in workflow fields can be changed with the **Set Field Value** action. When executed at server triggers such as Before Record Submit and Entry, the values are persisted to the database, but to the workflow, not the record.
  - Workflow fields cannot be seen on the main are of a form. Their values show up in the OPTIONS column on the Active Workflows and Workflow History subtabs.

There are two kinds of workflow fields:

- Workflow Field
  - A Workflow Field is created at the Workflow level
  - A Workflow Field is exposed as a parameter to another workflow that is calling it via the **Initiate Workflow** action
  - The value of a Workflow Field exists at each state throughout the workflow lifecycle

- State Field
  - A State Field is created within a specific State
  - The value of a State Field can only be set or retrieved at the State where it was defined. However, the value of a State Field is persists throughout the life of the workflow. If you leave a State and then return to it, the value of a State Field is preserved.

Investigate workflow **SuiteDreams Workflow State Fields** to see how workflow fields are manipulated.

*Compare capabilities of SuiteFlow and SuiteScript and determine when to use which technology

The following is a list of unique capabilities in SuiteFlow relative to SuiteScript. Think about using SuiteFlow for the following:

- Approval processing. This cannot be replicated easily in SuiteScript.
- **Add Button** action. In SuiteScript you can add a server button with `nlobjForm.addSubmitButton()`, but it does not operate the same as the **Add Button** action.
- **Lock Record** action. It is not possible to lock a whole record for editing in SuiteScript.
- **Subscribe To Record** action. This action allows a workflow to wake up and evaluate conditions when changes occur on related records. There is no direct equivalent in SuiteScript.
- **Set Field Display Type** at client triggers. In SuiteScript you can toggle whether a field is disabled or enterable at client events via `nlapiDisableField`, but you cannot toggle hiding/showing of fields that you can do at client triggers with **Set Field Display Type**.
- **Set Field Display Label**. The ability to change a field display label is not available in Client scripts.
- **Set Field Mandatory** on client triggers. Setting whether a field is mandatory or optional is not available in Client scripts.

There are some actions in SuiteFlow that have direct equivalents in SuiteScript, but may lack some of the options you have in SuiteScript. Examples:

- **Create Record** action. This is similar to `nlapiCreateRecord + nlapiSubmitRecord`. With `nlapiCreateRecord` you can initialize the record with a specific form. The preferred form is always used with **Create Record**. You cannot use **Create Record** to create transactional records since sublists need to be filled in, and they are not supported in SuiteFlow.
- **Go To Page** and **Go To Record** actions. In addition to navigating to standard pages and records, in SuiteScript you can also redirect users to Suitelet forms and external pages using `nlapiSetRedirectURL`.

- **Transform Record**. In SuiteFlow you are limited to transforming a subset of records where sublist manipulation is not required during transformation. The equivalent in SuiteScript, `nlapiTransformRecord`, allows you to transform more record types.

Some core features of SuiteScript that have no equivalent in SuiteFlow (this is not an expansive list):

- Suitelet scripts
- RESTlet scripts
- Portlet scripts
- Bundle Installation scripts
- Deep UI customizations on record forms, such as adding additional fields, subtabs, and sublists
- Calling external web services
- Retrieve and set sublist fields
- Process search results. In SuiteFlow you can only use a search result to initiate a workflow during SCHEDULED initiation.

Identify the usage of scheduling in workflow

There are 3 primary scheduling capabilities in SuiteFlow:

- You can schedule individual actions to execute beginning at a particular time of day or some amount of time after they have been processed. You can also set the actions to recur at various intervals. A typical usage of scheduled actions is to send approval reminder emails.
- You can schedule transitions to execute after a specified interval. A typical usage of scheduled transitions is with a lead nurturing/drip marketing workflows where you wait a period of time before transitioning to the next state to take further action such as sending a campaign email.
- You can scheduled the entire workflow to run once at a specified date, to run every 30 minutes, or repeat at other intervals on a daily, weekly, monthly, or yearly basis. Configuration is similar to that of Scheduled scripts. A difference is that a scheduled workflow is always driven by the results of a saved search. When a scheduled workflow runs, each saved search result is initiated as its own workflow instance. The record type of a saved search must be identical to the record type of the workflow.

Investigate workflow **SuiteDreams Scheduling** for an example of an entire workflow that is scheduled. The workflow does very little. It only shows the basic set up of a scheduled workflow. There is one action that sets a workflow field to a hardcoded value. This is so you can look at workflow history and see the workflow executed correctly.

Investigate workflow **SuiteDreams Scheduling 2** for an example of a scheduled action. A reminder email is sent to the current approver every hour until the approver clicks **Approve**.

*Determine how to use formulas in SuiteFlow

Formulas in SuiteFlow are very powerful. They fall under two categories:

- Formulas used with server triggered actions (e.g. Before Record Submit, Entry) may contain SQL. They are identical to the kinds of formulas you can embed inside of saved searches. Formulas must follow Oracle syntax rules (e.g. enclose all text in single quotes).
- Formulas used with client triggered actions (e.g. Before User Edit, After Field Edit) do not contain SQL! Instead these formulas are processed as client-side SuiteScript (e.g. `nlapiGetFieldValue` could be embedded inside of a formula)

Formulas can be used within action configuration, depending on the action:

- **Create Record**. Formulas can be used to set data into a field on the record being created.
- **Set Field Value**. Formulas can be used to set the value of a field.
- **Go To Record**. Formulas can be used to initialize fields on the record you are navigating toward.
- **Initiate Workflow**. Formulas can be used to set values on fields you are passing into a sub-workflow.
- **Transform Record**. Formulas can be used to set values on fields in the record you are transforming to.

There are several actions where you can embed fields into its action configuration, but full formula syntax is not supported. In **Show Message**, **Confirm**, and **Return User Error** (client and server triggered), you can embed fields using curly brace syntax into the TEXT parameter, e.g. `Phone: {phone}`. You can embed field values into the BODY text of **Send Email** actions similarly. Additional formula syntax is not supported.

Formulas can also be used inside of conditions at workflow initiation and across all actions and transitions by selecting CUSTOM FORMULA in the condition builder.

Investigate workflow **SuiteDreams Use Formulas** for a couple examples of formulas in actions.

Module 04 | Review SuiteTalk Exam Objectives

MODULE 04 | REVIEW SUITETALK EXAM OBJECTIVES

Module Overview

This module provides a review of SuiteTalk subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the SuiteTalk subject matter. The most missed objectives are identified with an asterisk (*) in the title.

Working with SuiteTalk in your training account

Provided you have set up an integration client in C#.Net or Java, you can dowload and run sample code identified within many of the objectives. Code is referred to by method name. Even if you do not have an integration client set up, you can still download and inspect it in a text editor:

- C#.Net code can be downloaded from the file cabinet in your training account at **Developer Certification > C#**
- Java code can be downloaded from the file cabinet in your training account at **Developer Certification > Java**

*Identify the SuiteTalk (Web Services) support, versioning, deprecation and General Availability policy of endpoints and the process for upgrading them

A new version of a WSDL comes out each NetSuite release, which happens twice per hear. Previous WSDL versions are forward compatible with the latest release. WSDL versions are supported for up to 3 years. If your current WSDL is less than 3 years old, you only need to upgrade in order to take advantage of functionality in the latest WSDL. The bulk of changes to the WSDL in a new release is to support new record types and fields that come with the NetSuite release. Sometimes there may be deeper changes to overall functionality such as new and updated web service operations. Make sure to always recompile your code whenever changing to a different WSDL.

See SuiteTalk method `getDeposit()` for code related to this topic.

*Identify SuiteTalk (Web Services) authentication methods and benefits

You can either log in with the **login** operation or set the **Passport** object into the header of the SOAP request.

You are performing session-level authentication with the **login** operation. You perform the **login** operation once, and then follow with subsequent operations such as **get, add, update,** etc. Don't keep executing the **login** operation unnecessarily, as it is an expensive operation. You need to perform session management when using the **login** operation.

You are performing request-level authentication when you place the **Passport** object within the SOAP request of each operation such as **get, add, update**, etc. The **login** operation is not used with request-level authentication. Request-level authentication is preferred as you do not need to worry about managing sessions.

See the `login()` SuiteTalk method for details around using the **login** operation (session-level authentication).
See the `setPassport()` SuiteTalk method for details around using request-level authentication.

Determine session policy for Web Services vs UI; and how sessions are managed for the same user across multiple integrated applications, including managing sessions with SuiteCloud Plus

A key point to remember: default processing allows you to have one active login for the same user in both web services and the user interface.

With SuiteCloud Plus you can increase throughput for the same user by allowing multiple simultaneous web service logins from a single user. The performance of an individual web services operation does not increase, but you can execute more operations at the same time.

Identify how to develop Data Center-agnostic integrations

NetSuite has two data centers as of the time of this writing. There are more data centers planned. Each data center has its own unique web services domain. An error is returned if you attempt to access your account using the domain of a different data center.

Your accounts can be moved at any time from one data center to the next. You need to make sure you dynamically discover the web services domain so that your web services integrations do not suddenly fail in the future. If you are a SuiteApp developer, then it is imperative that you make sure your integration applications work with any data center.

As of the time of this writing, these are the two data centers:

- U.S. West Coast
  - https://webservices.netsuite.com
- U.S. East Coast
  - https://webservices.na1.netsuite.com

See the `getDataCenterURLs()` SuiteTalk method to see how to dynamically discover your web services domain. As of the time this writing, all training accounts are in the U.S. East Coast data center, so the domain that should be returned is https://webservices.na1.netsuite.com.

Note that when you log in through the NetSuite user interface there is a proxy that redirects end users to the correct domain. As of the time of this writing, you'll see that you are logged into either https://system.netsuite.com or https://system.na1.netsuite.com.

Determine the appropriate search technique to use in SuiteTalk (Web Services)

There are 3 different search techniques you can use when executing a **search** operation. Which technique you use depends upon the type of search object passed into the **search** operation.

The 3 search techniques are as follows:

- Basic Search
  - Use a basic search when both the criteria and columns operate upon the target record of the search, e.g. "*search for sales orders where order date is within the last 30 days; return order date and order amount*"
- Joined Search
  - Use a joined search when the criteria needs to join out to a related record. Columns returned are still on the target record of the search, e.g. "*search for sales orders where customer territory is in the eastern region; return order date and order amount*"
- Advanced Search
  - Use an advanced search when the both the criteria and columns need to join out to a related record, e.g. "*search for sales orders where customer territory is in the eastern region; return order date, order amount, customer email, customer phone*"

See SuiteTalk method `searchEstimatesBasic()` for an example of a basic search.
See SuiteTalk method `searchEstimatesJoined()` for an example of a joined search.
See SuiteTalk method `searchEstimatesAdvanced()` for an example of an advanced search.

*Identify how asynchronous and synchronous Web Services APIs impact integration design and implementation

Asynchronous operations can be used to even out the load on NetSuite servers. Since asynchronous operations are less demanding on the CPU, you can often process larger amounts of data in a single list or search operation versus their synchronous counterparts. See **Web Services Governance** in the Help Center for specifics. To summarize, when processing asynchronously you can often place a larger

number of records in **addList** and **updateList** operations, as well as a use a higher page size when searching.

If your integration application is executing in a part of the world where internet connectivity is spotty, the responses to asynchronous operations can be retrieved at a later time programmatically, whereas if a synchronous operation fails after submitting the request, you cannot retrieve its response programmatically.

Asynchronous operations are **asyncAddList**, **asyncUpdateList**, **asyncUpsertList**, **asyncDeleteList**, **asyncGetList**, **asyncSearch**, and **asyncInitializeList**.

There are 3 example SuiteTalk methods showing how you can execute asynchronous operations:

- `searchEstimatesAsync()` submits the request for an **asyncSearch** operation
- `checkAsyncStatus()` retrieves the status of an asynchronous operation
- `getAsyncSearchEstimatesResults()` retrieves the response to an **asyncSearch** operation

*Identify strategies for accessing and managing account-specific customizations

SuiteTalk WSDLs fully support custom fields and custom record types. You can get and manipulate custom field and custom record data just like standard fields and standard records. Because each account has their own customizations, they are reflected in the WSDL in a generic manner. The WSDL is the same for everyone. A WSDL is not re-generated based on account customizations.

See SuiteTalk method `getCustomerCustomField()` which shows an example of how to access a custom field called lead category. Two things to note when you look at this method:

- The ID (also known as script id) is evaluated prior to casting to a specific type. There can be multiple custom fields returned across different data types. You need to make sure you are processing the intended custom field before doing anything else with it.
- IDs (script Ids) are the same when you deploy custom fields from one account to the next, so there are no special precautions you need to take.

See SuiteTalk method `getPerformanceReview()` which shows an example of retrieving a record from a performance review custom record type. There is one major thing to note with this method:

- When retrieving or manipulating custom records through web services, the identity of the custom record type is identified via its INTERNAL ID. The value of INTERNAL ID varies when the custom record type is deployed to other accounts. You have no control over this. The best practice is to execute the **getCustomizationId** operation to retrieve the INTERNAL ID based on the ID (also known as script id). The ID does not change when you deploy the custom record type from one account to the next. Use of **getCustomizationId** is a best

practice, and you will see it in this example. Your web service integrations could break when you deploy code to a different account if you do not use **getCustomizationId**.

Identify how to build efficient data synchronization

For some data that is stored in NetSuite, the system of record might be an external database. For example, employees are stored in NetSuite, but might also be stored in an external HR system. The primary key that identifies the employee record in the external system can be associated as a foreign key inside of NetSuite. In NetSuite the foreign keys are named External Id. In SuiteTalk you can associate External Id with any record supported in web services.

You can get, add, update, and delete records based on External Id. You can assign an External Id to the record at the time of add or update. There are two specialty operations that are intended for use with External Id: **upsert** and **upsertList.** If the External Id passed into **upsert** or **upsertList** does not exist, the record is added (and External Id gets associated with it), otherwise record is updated based on External Id.

See the `manipulateItemViaExternalId()` SuiteTalk method for a simple example using **upsert**.

*Identify the impact of scripts and workflows on integrations

Scripts and workflows may unknowingly execute during your add and update operations, which can seriously impact performance. When you add or update a record through SuiteTalk, a user event is generated. Any User Event SuiteScript with Before Submit and After Submit events defined will execute. If you do not require the scripts to execute during integration, then you should place a check for execution context at the top of your script, and exit the script if it is running due to a web services integration. E.g.

```
if (nlapiGetContext().getExecutionContext() == 'webservices') {
    return;
}
```

The `addTask()` SuiteTalk method creates a task record.

SuiteDreams UE Impact On Integrations is a User Event script that is deployed to the task record. Deploy this and you'll see that it executes from the `addTask()` method.

Module 05 | Review SuiteBundler Exam Objectives

MODULE 05 | REVIEW SUITEBUNDLER EXAM OBJECTIVES

Module Overview

This module provides a review of SuiteBundler subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the SuiteBundler subject matter. None of the SuiteBundler objectives were in the list of most missed objectives.

Identify the role of installation scripts in SuiteBundler

Installation scripts are associated with a bundle. During bundle installation an installation script can validate if the target account has the necessary features and configuration. Bundle installation scripts can also perform some post-installation work such as add custom record data into the account.

To try out a bundle installation script, install bundle **73322** from production account **TSTDRV1381823**. It is a public bundle. The bundle contains a single saved search, but that is not the focus of the bundle. The focus of the bundle is content contained in the associated bundle installation script.

When the bundle is installed, script **SuiteDreams BI Bundle Automation** is installed with it. You may get an error upon initial install of the bundle (this is intended as part of the example). You can then make fixes in your account (e.g. enabling a feature) and re-install. Note that if you choose the **Update** action from the installed bundle list to try again, Before Update and After Update events get called, and right now there is no code in these events. So to try again, choose **Uninstall** and then re-install.

Following is the Before Install and After Install event code of the bundle installation script:

```
function beforeInstall(toversion) {
    nlapiLogExecution('AUDIT', 'version of bundle about to be installed', toversion);

    var context = nlapiGetContext();

    // Determine if issue management is enabled.
    // Feature at Setup > Company > Enable Features, CRM subtab, ISSUE MANGEMENT
    // (in Support section).
    // When feature enabled, issues can be created at List > Support > Issues
    var issueMgmtEnabled = context.getFeature('ISSUEDB');
    nlapiLogExecution('AUDIT', 'ISSUEDB feature', issueMgmtEnabled);

    // Throw error to stop installation process when issue management is not enabled.
    // The idea is that code inside of the bundle requires the feature, perhaps to
    // create Issue records. When error is thrown, it will show up as red X under the
    // STATUS column when viewing the list of installed bundles. Hover over the red X,
    //and the message that is thrown will display in a pop-up
    if (issueMgmtEnabled){
        nlapiLogExecution('AUDIT', 'Issue Management Enabled');
    } else {
        nlapiLogExecution('AUDIT', 'Issue Management NOT Enabled');
        throw nlapiCreateError
        ('BEFOREINSTALLERROR', 'Issuem management feature must be enabled', true);
    }
}
```

```
function afterInstall(toversion) {
    nlapiLogExecution('AUDIT', 'version of bundle installed', toversion);

    // Submittal of an issue record should fail if the feature was not
    // enabled. The idea is this code could either be part of the SuiteApp
    // itself, or part of the post-install step as is done here. The issue
    // creation is placed here for ease of demonstrating bundle installation
    // scripts. You could easily test this by commenting out the nlapiCreateError
    // statement in the Before install event and making sure Issue Management
    // is not enabled.
    // When this After Install executes and Issue Management is not enabled,
    // an exception is thrown when executing nlapiCreateRecord against the issue
    // record. Because exception is handled, bundle ends up being installed
    // successfully. If exception were unhandled in After Install, the bundle
    // would be installed, but there would be a red X for the bundle STATUS
    // in the list of bundles.
    try {
        var recIssue = nlapiCreateRecord('issue');
        recIssue.setFieldValue('issueabstract', 'sample issue to test bundle');
        var issueId = nlapiSubmitRecord(recIssue);
        nlapiLogExecution('AUDIT', 'issue id', issueId);
    } catch (e){
        nlapiLogExecution('AUDIT', 'issue could not be created, ' +
                          'likely because feature was not enabled');
    }
}
```

Determine how SuiteBundler handles collisions or conflicts during installation or update

In this section there is a summary of what happens when bundles are deployed from one account to another, including when the source account is a sandbox.

With regular accounts (when not dealing with sandbox)

Installing new bundle

- When script id of bundle object does not exist in target account
  - Result is that object is added
- When script id of bundle object exists in target account
  - A conflict is identified since objects with script ids are used to determine whether there are conflicts
  - You can choose to replace the object
  - Or you can choose to add the object as new. If object is added as new, then a sequence number is appended to the end of the script id. **Note:** SuiteScripts that are in the same bundle have any references to the renamed objects automatically modified.

Updating bundle

- When script id of bundle object does not exist in target account
  - Result is that object is added
- When script id of bundle object exists in target account
  - If the object existed in the bundle during original bundle installation, then it is seen as an updated object and there is NO option to choose to rename the script id. The definition of the object in the target account is automatically replaced.
  - If the object did not exist in the bundle during original bundle installation, then it is seen as a conflicting object and the installer has the choice to replace or add as new just like when installing a new bundle.

With sandbox accounts (when deploying from sandbox to production)

Updates to objects are automatically managed when installing between sandbox and production.

Updating bundle

- When script id of bundle object does not exist in production account
  - Result is that object is added

- If object existed in bundle during original bundle installation, there is an automatic determination of whether the object was changed in the sandbox account
  - If object was changed in the sandbox account, then the update proceeds with the object in the production account
  - If object was not changed in the sandbox account, then no update occurs to the object in the production account. The object is not replaced.
- Object changed in production but NOT in sandbox
  - Nothing happens during bundle update, so change in production is left alone.

Identify the impact of a sandbox account refresh on bundles

One important tip regarding sandbox account refresh:

- Make sure you have deployed bundles to production before performing a sandbox account refresh, otherwise the bundles are lost since all content in production replaces all content in sandbox.

Important note on the behavior of bundle definitions during sandbox account refresh:

- You create a bundle in sandbox, so its definition exists in sandbox. You deploy the bundle to production. The bundle is installed in production, but the bundle definition exists only in sandbox. When you run a sandbox account refresh, the system is smart enough to recreate the bundle definition back in the sandbox account. Your bundle definitions that were originally defined in sandbox are preserved during a sandbox account refresh.

Module 06 | Review SuiteBuilder Exam Objectives

MODULE 06 | REVIEW SUITEBUILDER EXAM OBJECTIVES

Module Overview

This module provides a review of SuiteBuilder subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the SuiteBuilder subject matter. The most missed objectives are identified with an asterisk (*) in the title.

*Determine the impact of form customization on records

Impact of form customization in SuiteTalk

A field's display type (standard and custom fields) can impact add and update operations from SuiteTalk web services. SuiteTalk behaves differently than SuiteScript when it comes to setting field values that are otherwise not editable in the user interface.

Add and update operations through SuiteTalk fail when trying to set a value on a field that is hidden, inline text, or disabled. You can only update field values through SuiteTalk if the field is editable in the user interface. SuiteTalk behaves like an end user and uses the preferred form unless otherwise designated. You can take fields that are editable by default and then make them non-editable (hidden, inline text, disabled) through form customization.

Make sure to use a web services only form in your integrations and explicitly identify it through the **customForm** property in your record objects. Otherwise an administrator could customize a form, making it the preferred form, and on the custom form disable some fields for editing that are used in your integration. This will break your integration.

See SuiteTalk method `updateEmployee()` for a form customization example. The method is documented with additional details. The method uses a custom form named **Employee Form Customization Impact**.

Impact for form customization in SuiteScript

Impact of form customization in SuiteScript is just the opposite of SuiteTalk behavior. You can set fields in SuiteScript that are non-editable based on a custom form, and the updates work. In SuiteScript you can set values into fields that are hidden, inline text, or disabled in the user interface.

You can test behavior in SuiteScript with some sample code in **SuiteDreams_NoScript_FormCustomizationImpact.js**. Copy and paste the sample code into the script

debugger. The script uses the same form as in the SuiteTalk example: **Employee Form Customization Impact**.

Identify the performance implications of adding custom fields and the strategies to mitigate performance impact

Having too many custom fields on your forms can lead to performance issues multiple ways:

- It can take longer to load a form in the user interface
- It can take longer to load a record via SuiteTalk
- It can take longer to load a record via SuiteScript

You can try the following things to help mitigate performance impact:

- Use a web services only form when adding and updating records so you can reduce the number of fields to process. The idea is to hide all the fields that are not required for the integration.
- Be careful with custom fields that source in data from other fields. Fields initialized with the result of formulas can be CPU intensive.
- So that searches are not inadvertently created with too many custom fields, make sure to turn off the SHOW IN LIST option on the custom field definition

SuiteDreams_NoScript_PerfImplicationsCustomFields.js shows an example of creating a custom field that sources in a default value based on the result of a summary search. This is CPU intensive. The example goes on to show how you can keep the sourcing from occurring during execution of functions such as `nlapiLoadRecord` by manipulating access levels. The script example is heavily documented with additional details. The example uses the following custom objects:

- Custom Transaction Body Field: **Perf Implications Custom Field**
- Saved Search: **Perf Implications Custom Field Summary Field**

SuiteTalk method `getQuote()` repeats the same script example in web services. In web services the custom field with its default value is returned unless access is removed or the field is set to hidden. You should inspect the SOAP response to test. If the field is being pulled back with the default value, you'll see it in the **customFieldList**, otherwise the field is not returned.

*Given a scenario, select the sourcing and filtering criteria, or the defaulting and validation options for custom fields

Options are varied. All examples are defined on the **Furniture** custom record type, but concepts apply the same to custom fields on standard record types.

Data used for testing the options:

- Project Manager: **Larry Nelson**
- Customer: **B&B Design**

Scenario: Source in the Hire Date from the Project manager's employee record

Option: You can source in data from fields on related List/Record fields.

Field Definition: Open Project Manager Hire Date. Look at configuration on the **Sourcing & Filtering** subtab.

Scenario: Calculate the number of days the Project Manager has been employed

Calculation is based on the difference between today's date and the Project Manager's Hire Date.

Option: You can calculate a field value using a formula at the time a record displays.

Field Definition: Open Days Manager Employed. Look at the DEFAULT VALUE on the **Validation & Defaulting** subtab.

Scenario: Default the Project Participant to the Current User

Option: You can default a field value based on dynamic information of the current user.

Field Definition: Open Project Participant. Look at the DYNAMIC DEFAULT on the **Validation & Defaulting** subtab.

Scenario: Display a list of filtered sales order transactions

The transactions are filtered to those that are related to the selected Customer.

Option: You can filter a dropdown list based on static and dynamic data.

Field Definition: Open Related Order. Look at the filters defined on the **Sourcing & Filtering** subtab. The filter for Type filters transactions to sales orders. The filter for Customer filters to those sales orders where Customer is the same as the Customer on the Furniture record. Nothing displays in the Related Order dropdown list until a Customer has been selected.

Field Definition: Open Related Order 2. This performs the same filtering as Related Order. The difference is that SOURCE LIST and SOURCE FILTER BY are used to configure the dynamic portions of the filter.

Scenario: Display a list of sales order transactions filtered by multiple dropdowns

The transactions are filtered to those that are related to the selected Customer plus those where the Project Manager is the Sales Rep on the transaction.

Option: You can filter the values in one dropdown list based upon the selections in two or more other dropdown lists. This capability is called *Multiple Dependent Dropdowns*.

Field Definition: Open Related Order By Project Manager. This is identical to the Related Order field, but adds an additional dynamic filter that relates Sales Rep (transaction) with Project Manager (furniture record). Nothing displays in the Related Order By Project Manager dropdown list until both a Project Manager and a Customer have been selected.

Scenario: Sum the amount of line items on a quote and place into custom field

Option: You can calculate the value of a field using the result of a summary saved search.

See details for objective **"Identify the performance implications of adding custom fields and the strategies to mitigate performance impact"**

Identify record-locking behavior and options

Optimistic locking is built into all standard record types. Optimistic locking is optional (based on checkbox selection) in custom record types. The best practice is to enable optimistic locking for custom record types as it could otherwise lead to data integrity issues. The option to disable optimistic locking is for backward compatibility purposes.

SuiteDreams_NoScript_RecordLocking.js has a detailed example of record locking behavior for standard and custom records.

*Identify the various options available under the permissions model for custom records

Access to custom record types and their fields can be controlled by configuration on the Custom Record Type definition, individual Fields on the Custom Record Type, or a custom Role.

On the Custom Record Type definition there are 3 choices under ACCESS TYPE. These do not impact data access for searching:

- When ACCESS TYPE = Require Custom Record Entries Permission, configuration on custom Roles is used
  - Customize a standard role or edit an existing custom role.
  - Go to the **Permissions** subtab, **Custom Record** sublist

    - Add permissions for Custom Record types
- When ACCESS TYPE = Use Permission List, configuration on the **Permissions** sublist of the Custom Record Type definition is used. Note that when you add custom record permissions to custom roles, by default they show on the **Permissions** sublist.
- When ACCESS TYPE = No Permission Required, anyone can gain access to the custom records.
    - You might combine this setting with unchecking ALLOW UI ACCESS. This allows scripts and integrations full access, but the record type is not accessible via the user interface
    - As an example, you'll find No Permission Required used on some custom record types that come with our SuiteCommerce Advanced product where the data is considered public. For example, there is a Product Reviews custom record type that is considered public and is set to "No Permission Required". The idea is that product review data is accessible by any shopper.

You can edit the individual custom field for more fine-grained field specific access. You can also designate access via searching. Edit a custom record type's custom field and view the **Access** subtab.

If a field on your custom record type is a List/Record of type Subsidiary, Department, Location, or Class, then you can check the APPLY ROLE RESTRICTIONS box on the custom field to restrict employees to viewing custom records where the values in these fields match what is on their employee records.

Identify implications of deleting or inactivating custom record types

Data on all custom records is lost if you delete the Custom Record Type definition. It is not possible to delete a Custom Record Type if there are custom fields referencing the Custom Record Type, e.g. a List/Record custom entity field that points to the custom record must be deleted first. The system alerts you to this.

Inactivating a Custom Record Type removes the definition from the list of record types unless you explicitly check the SHOW INACTIVES checkbox. Menu links for the Custom Record Type are no longer visible, but the link to list custom records is still active. No custom records are deleted. You may continue to add and modify custom records. You can continue to search on custom records.

Module 07 | Review SuiteAnalytics Exam Objectives

MODULE 07 | REVIEW SUITEANALYTICS EXAM OBJECTIVES

Module Overview

This module provides a review of SuiteAnalytics subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the SuiteAnalytics subject matter. The most missed objectives are identified with an asterisk (*) in the title.

Identify implications of using saved searches and coded searches

Coded searches:

- Harder to maintain
- May be necessary in some instances

Saved searches:

- Easier to maintain
- If you only need part of the search to be dynamically coded, then you may be able to create a base saved search, and then add dynamic filters through code
- There is an extra layer of permissions
  - Public versus private
  - Audience
  - Run Unrestricted

The best practice is to use saved searches in SuiteTalk and SuiteScript when you can.

SuiteDreams_NoScript_SavedSearches.js is a simple example of executing a saved search from script that can be run in the script debugger. It executes the search **Find Estimates in Specific Subsidiary**. The search includes a formula field to show how you can extract formula results in script.

See SuiteTalk method `searchEstimatesAdvancedSavedSearch()` to see an example of executing a saved search from SuiteTalk. An Advanced Search must be used to execute a saved search. A difference with SuiteTalk is that formula field results cannot be returned from the **search** operation because the SuiteTalk schema does not support formula fields. The SuiteTalk example uses the same saved search as for SuiteScript.

*Identify implications of managing a large volume of search results

There are two methods of searching in SuiteScript:

- nlapiSearchRecord
  - `nlapiSearchRecord` has a 1000 result limit. There are no errors generated when the limit is reached. If you sort your data by internal id, then you can set up a loop to keep re-executing the search with the addition of a dynamic filter (e.g. where internal id > 3333). This will allow you to get through all results after having performed multiple executions of `nlapiSearchRecord`.
  - If you are going to exceed governance limits based upon the size of the result set and/or how much processing you do with each search result, you can always wrap your design inside of scheduled script processing. You can also take advantage of the schedule script yield capability.
  - **SuiteDreams_NoScript_LargeVolumeSearchResults.js** is a fully functioning example of processing search results beyond 1000 using `nlapiSearchRecord`. You can copy/paste the code and run it in the debugger.
- nlobjSearch via nlobjSearchResultSet.getResults()
  - There is a 1000 result limit when using `getResults()`. However, you can keep re-executing the method to get another 1000 results.
  - **SuiteDreams_NoScript_LargeVolumeSearchResults2.js** is a fully functioning example of processing search results beyond 1000 using `nlobjSearchResult.getResults()`. You can copy/paste the code and run it in the debugger.
  - Like `nlapiSearchRecord`, you can wrap your design inside of scheduled script processing. However, `nlobjSearchResultSet` cannot be saved during a yield. You must null out the object if you are executing a yield.
  - Some informal testing has shown that `nlapiSearchRecord` performs better than `nlobjSearchResult.getResults()`. Each execution of the `getResults()` method is internally performing another search.
- nlobjSearch via nlobjSearchResultSet.forEachResult()
  - There is a 4000 result limit when using `forReachResult()`.
  - There is no code example provided for this, but you should be able to use a large volume search result pattern similarly to `nlapiSearchRecord`.

In SuiteTalk you can use **searchNext**, **searchMore** and **searchMoreWithId** operations to manage large volumes of search results, though only **searchMoreWithId** can be used with request-level authentication. The **search** operation executes a search and then returns a number of results up to the limit specified by the **pageSize** preference that is either set globally at **Setup > Integration > web Services Preferences**, or in code on a per request basis. Page size maximum can vary, though is often at 1000. See Help Center topic on Web Services Governance for specifics. SuiteTalk method

`searchEstimatesProcessPages()` shows an example of using **searchMoreWithId** to process multiple pages. The **pageSize** preference is set very low to make it easy to run a paging example.

*Identify implications of search techniques in SuiteTalk and SuiteScript

Most of the implications of search techniques can be found by going through earlier objectives, especially the last two: "Identify implications of using saved searches and coded searches" and "Identify implications of managing a large volume of search results".

Here are a few additional points that mainly draw out the differences between SuiteScript and SuiteTalk searching.

SuiteScript

- Execute and process results of summary searches
- Process result of formula fields
- Using nlobjSearch
  - Create saved searches dynamically
  - Redirect end user to search results
- Execute saved searches or coded searches

SuiteTalk

- Cannot execute summary searches
- Cannot process result of formula fields
- Basic Search and Joined Search
  - No ability to execute saved searches
- Advanced Search
  - Execute saved searches or coded searches

Module 08 | Review Design Fundamentals Objectives

MODULE 08 | REVIEW DESIGN FUNDAMENTALS OBJECTIVES

Module Overview

This module provides a review of Design Fundamentals subject matter from the SuiteCloud Developer Exam objectives.

The following section titles mirror the exam objectives within the Design Fundamentals subject matter. The most missed objectives are identified with an asterisk (*) in the title.

Determine how to set roles and permissions for a given situation in various environments

Always think about reducing the set of permissions a user has when possible. Remember that a user can be logged into the NetSuite user interface, running SuiteScript, running SuiteFlow, or running integration code.

Integration

- Don't just use the Administrator role.
- Create a custom role to use with your web services integration, configuring only the permissions required by the integration code. This applies to SuiteTalk and RESTlet based integrations.
- If you are logging in through SuiteTalk with a user that also has UI access, make sure to mark the custom role as a WEB SERVICES ONLY ROLE. This is on the role definition.

SuiteScript

- Be aware that by default when you deploy a User Event, Portlet, Suitelet, Client, Workflow Action, or Mass Update script, they always run with the permissions of the current user. It is very important to make sure your scripts will run once they are deployed to production. It is likely you will develop a lot of these scripts as an Administrator, so be careful and make sure they also run under the roles that will be executing the scripts. For example, if a script creates a Task record, you need to make sure that all the roles running this script have Create permission on this record type.
- For User Event, Portlet, Suitelet, Workflow Action, and Mass Update scripts: it is possible to change the role the script executes under by changing the script deployment EXECUTE AS ROLE value to something other than Current Role. Be careful not to abuse this. Only change the role when needed.

    - If one of these scripts needs to call a scheduled script using `nlapiScheduleScript`, then you will need to set the script to run as an Administrator. Only Administrators can run scheduled scripts, and this includes when they are executed via `nlapiScheduleScript`.
- Client scripts always execute under the role of the current user. You cannot change this! One workaround in client scripts is to call a Suitelet when needing permissions not normally available. See objective "**Identify the risks of implementing only client-side validations and strategies to address them**" for the approach of calling a Suitelet from the client-side.
- Scheduled and Bundle Installation scripts always run as an Administrator.
- RESTlets run based on the role configured on the http Authorization header.
- Do not mimic permission restrictions inside of a script. For example, you could create roles that grant Full permission to a particular record type and then evaluate that role directly in script (`nlobjContext().getRole()`) to limit access to View or Create. This is not recommended. Instead fully use our role and permissions model, appropriately setting the access level on the role's permissions.

SuiteFlow

- Workflows running on client triggers (e.g. Before User Edit) act just like Client scripts. They always execute under the role of the current user.
- Workflows running on server triggers (e.g. Before Record Submit) can be set to run as an Administrator by checking EXECUTE AS ADMIN in the Basic Information section of the Workflow. You may need to do this for the following common use case: "The next approver is the next manager up the supervisor hierarchy". To get the next manager up the supervisor hierarchy requires access to the SUPERVISOR field on the employee record. Permission to the employee record is needed, and it is unlikely the end user will have access to the employee record.

SuiteBuilder

- For custom record types, see the objective "**Identify the various options available under the permissions model for custom records**"
- Custom fields have access control configuration. Configure as applicable for the custom field. See the **Access** subtab of a custom field.

*Identify strategies for and implications of role management and authentication when integrating with external systems

Inbound Integration (external system to NetSuite)

All methods of inbound integration described below are documented in the NetSuite Help Center.

Never store NetSuite passwords in an external system!

Inbound Single Sign-on

If there is a SuiteTalk integration that does not request a user to enter their password at the time of integration (e.g. through a GUI), then you should use NetSuite's Inbound Single Sign-on feature instead of storing passwords in an external system. When you use this feature with SuiteTalk, you authenticate using the **ssoLogin** operation. You can set up an initial mapping with the **mapSso** operation.

Note that when **ssoLogin** is used, you cannot use request-level authentication

You can also use Inbound Single Sign-on to log into NetSuite outside of SuiteTalk, for example from an external user interface.

Specialized inbound integrations

Google applications

- There is a special form of single sign-on integration when coming from a Google application to NetSuite called OpenID Single Sign-on

SAML Single Sign-on

- Allows access to NetSuite from any SAML 2.0 compliant identity provider, e.g. Microsoft Active Directory

RESTlet integration

Authentication in RESTlets is provided via the http Authorization header. This can use either the NLAuth or OAuth methods.

- NLAuth embeds the password of a NetSuite user in the request, so like with SuiteTalk, do not use this method of authentication unless a user is being requested to enter credentials at the time of integration in an external application.
- Oauth supports token-based authentication, so user credentials are not passed in with the request.

Outbound Integration (NetSuite to external system)

All methods of outbound integration described below are documented in the NetSuite Help Center.

Never store passwords for external systems inside of NetSuite!

Outbound Single Sign-on (also called SuiteSignOn)

Use SuiteSignOn as a means to integrate with an external system when authentication with that system is required. SuiteSignOn makes it so you do not have to store external system passwords inside of NetSuite.

nlapiRequestURLWithCredentials

When SuiteSignOn is not possible, you can request credentials from a user via credential fields, and then pass them to `nlapiRequestURLWithCredentials`. Credentials fields can be added in a Suitelet using `nlobjForm.addCredentialField()`.

Suitelet script **SuiteDreams SL Auth Ext Systems** provides a partial example of adding credential fields, followed by a call to `nlapiRequestURLWithCredentials`.

*Identify NetSuite functionality and recommended practices related to restricted data fields (PII, PCI, e.g.)

PCI

- PCI = Payment Card Industry
  - See https://www.pcisecuritystandards.org for PCI security standards
  - NetSuite is compliant with PCI Level 1 DSS (data security standards)
  - PCI compliance requirements dictate the many restrictions of credit cards in NetSuite
- View Unencrypted Credit Cards permission
  - This permission must be enabled in the account. This permission allows you to view the credit card number in edit mode, but not view or print it.
  - If a NetSuite user has this permission, then passwords must be changed every 90 days and the minimum password length is 7 characters. This applies equally to web service users.
  - Standard roles with this permission: Administrator, Accountant, Bookkeeper, Controller, A/R Clerk.
  - A Strong password policy is required in the account when any user has the View Unencrypted Credit Cards permission.

- Credit cards may be unmasked when editing a customer record if the user has this permission.

- Storing of credit card data
  - You must only use the **Credit Cards** subtab to store credit card information. For example, open a customer record and go to the **Financial, Credit Cards** subtab.
  - Do not store the card security code in NetSuite.
  - Credit cards are masked except when first entering them, unless a role is being used that has the View Unencrypted Credit Cards permission.
  - Credit card masking applies to the user interface, SuiteTalk, and SuiteScript. It is not applicable for SuiteFlow since SuiteFlow cannot access sublists.
  - In the user interface, SuiteTalk, SuiteScript, and SuiteFlow: you cannot search on a specific credit card number in the account. You can only search for whether a credit card is empty or not empty. This is true within web services, SuiteScript, and the user interface.

PII

- PII = Personally Identifiable Information such as Social Security Number
- See http://en.wikipedia.org/wiki/Personally_identifiable_information
- The Social Security Number in NetSuite has special restrictions
  - Administrators have default access to the Social Security Number
  - For other roles, access to Social Security Numbers is granted through the Employee Social Security Numbers permission. Roles can be granted access to view the full number or view a masked version.

Determine how to optimize performance, scalability, and reliability in SuiteScript, SuiteFlow, and SuiteTalk

Technology	Performance, Scalability, and Reliability Information
SuiteTalk	Searching. Prefer Advanced Searches because you can limit result to the desired columns. This reduces size of the SOAP response. See objective "**Determine the appropriate search technique to use in SuiteTalk (Web Services)**"
SuiteTalk	Searching. If you are searching and need most of the record except for sublists, then perform a Basic or Joined Search, but set the **bodyFieldsOnly** preference to true.
SuiteTalk	Working with a list of records. Prefer using the list operations instead of executing multiple single record operations where applicable. For example, use a single **getList** instead of multiple **get** operations. This reduces the number of requests. You can mix record types within a single **getList**, **addList**, **updateList**, or **deleteList**.
SuiteTalk	Prefer request level authentication versus session level when applicable. It is easier to manage authentication with request level authentication, and therefore more reliable. See objective "**Identify SuiteTalk (Web Services) authentication methods and benefits**"
SuiteTalk	When not using request level authentication, do not continuously execute the **login** operation. Make sure to reuse sessions whenever you can. The **login** operation is expensive.
SuiteTalk	Create web services specific roles and forms. This will enhance reliability. You don't have to worry about administrators changing roles and forms used by users in the NetSuite application, which could then break an integration. And with web services forms you can minimize the number of fields on the form to only those fields needed. This will improve performance. See objective "**Determine the impact of form customization on records**"
SuiteTalk	SuiteTalk has a set of asynchronous operations that are equivalents to **getList**, **addList**, **updateList**, **deleteList**, and **search**. Asynchronous operations can be better performing. Think about using these when a real-time response is not needed. See objective "**Determine how asynchronous and synchronous Web Services APIs impact integration design and implementation**"
SuiteScript	Only execute user event scripts when needed. Always check `nlapiGetContext().getExecutionContext()` and end the script if the user event is running in a context where script execution is not necessary. This is often the case for SuiteTalk integrations and CSV Import. You can often gain huge performance improvements with your integration code just by checking execution context. See objective "**Identify the impact of execution context on user event scripts**" See objective "**Identify the impact of scripts and workflows on integrations**"

SuiteScript	Only execute user event scripts when needed. Check the **type** parameter (this represents event type) in your Before Load, Before Submit, and After Submit functions, ending the script when it is not necessary to execute under a particular event type. See objective "**Recognize script deployment configuration across script types**"
SuiteScript	Only execute user event scripts when needed. You can configure your user event script deployments to execute for a specific event type. In this way you can keep the script from even beginning to execute. With this approach, checking for **type** in the event function is not necessary.
SuiteScript	Only log when necessary. Make sure to change the LOG LEVEL on script deployments as you release a script into production. Changing LOG LEVEL to something other than Debug increases performance since all of your `nlapiLogExecution` statements with DEBUG as the 1st parameter will no longer write to the log. This increases performance for production scripts. See objective "**Recognize script deployment configuration across script types**"
SuiteScript	Offload user event After Submit processing to a scheduled script to enhance performance for your end users. An end user waits for completion of all After Submit code before a response is returned to them in the user interface. Any code that can be offloaded to a scheduled script can increase performance because the scheduled script executes asynchronously. See objective "**Given a scenario, identify the proper pattern to invoke a scheduled script to support a long running process**"
SuiteScript	Prefer `nlapiSubmitField` over the combination of `nlapiLoadRecord` & `nlapiSubmitRecord`. There is less governance usage and `nlapiSubmitField` is better performing.
SuiteScript	Prefer `nlapiLookupField` over `nlapiLoadRecord` when possible. They have the same governance usage, but `nlapiLookupField` is better performing
SuiteScript	Do not re-load records in After Submit. If you need to change the current record, then do so in Before Submit, so the changes can be captured as part of the automatic persistence to the database just after Before Submit.
SuiteScript	Do not load every record in the result of a saved search. If you need to return additional data, then build that into the search.
SuiteScript	For reliability, wrap your code in try/catch blocks.
SuiteScript	If you need to bring in some additional data to the browser, think about executing `nlapiRequestURL` with the **callback** parameter from the Page Init function of a client script. This executes asynchronously, so does not impact performance.
SuiteScript	In client scripts, filter script execution in Field Changed event functions based on field id to improve stability and performance.
SuiteScript	Be careful of using SQL formulas in your scripted searches as they could negatively impact performance because the formulas may run inefficiently against the database.

SuiteScript & SuiteTalk	Use **startswith** and **haskeywords** search operators instead of contains whenever possible. Use of **contains** can negatively impact performance.
SuiteScript & SuiteFlow	For reliability, execute code in After Submit instead of Before Submit where applicable. For example, if you create a record based on submittal of the current record to the database, and you create the record in Before Submit, there is no rollback of your created record if the submittal of the current record fails just after Before Submit. Note: equivalents in SuiteFlow are Before Record Submit and After Record Submit.
SuiteFlow	Only execute workflows when needed. Workflow initiation and actions can be filtered based on CONTEXT. This is identical to `nlapiGetContext().getExecutionContext()` in SuiteScript. It is important to filter your workflows based on CONTEXT as workflows, like script, can execute from SuiteTalk and CSV Import.
SuiteFlow	Only execute workflows when needed. Workflow initiation and actions can be filtered based on EVENT TYPE. This is identical to event type in user event scripts (i.e. the **type** parameter in your event functions). Filter execution when applicable.
SuiteFlow	Be careful of the code you have inside any workflow action scripts, as that can impact performance.
SuiteFlow	Execute **Set Field Value** actions in Before Record Submit whenever possible. Executing them in After Record Submit causes the system to internally reload and resave the record.
SuiteFlow	Make sure to uncheck the ENABLE LOGGING checkbox when releasing a workflow in production, otherwise the logging can negatively impact performance.

*Determine how to trouble-shoot and debug in SuiteScript, SuiteFlow and SuiteTalk

Take any number of scripts, workflows, and web service operations supplied with your training account and run them using the trouble-shooting and debugging tools described below.

SuiteScript

Without Script Debugger

- Use `nlapiLogExecution`. This runs in server scripts and record-level client scripts.
- Use the `alert` statement temporarily while debugging. This is part of the browser implementation of JavaScript, so it runs in client scripts only.

With Script Debugger

- You are working with the same data when logged in through the debugger domain. You are **NOT** working with a copy of your account data, so be careful of any changes you are making to your data based on the environment you are running in.
- **Debug Script** button
  - Use when debugging a set of script that is not related to a specific script type
  - When you test in this way realize you may get some unexpected behavior with certain statements, so it may be a good idea to comment the following statements (there may be others) when copying and pasting into the debugger:
    - Some `nlobjContext` methods. Some methods may not return valid information, such as methods to get script id or script deployment id (methods `getScriptId` and `getDeploymentId`).
    - `nlapiGetNewRecord()` and `nlapiGetOldRecord()`. These only have context inside of a user event script. If you are copying and pasting some user event script code into the debugger, you can temporarily add an `nlapiLoadRecord` statement so you have an `nlobjRecord` object to work with.
- **Debug Existing** button
  - Make sure STATUS on the script deployment is set to **Testing** for the script to be available in the debugger.
  - Most server scripts can be debugged in the script debugger.
- About the data you see in an `nlobjRecord` object
  - The properties inside the **fields** object of an `nlobjRecord` object is a mix of public and private data. To see what is officially scriptable, you must go to the SuiteScript Records Browser and look in the applicable **Fields** section. The debugger can still be helpful, especially if you are working with a record type containing many fields without user interface labels (this means you cannot click a label link to retrieve the field id). In the user interface, set unique data into those fields not having labels. In the debugger, enter a simple script to load a record and find the field containing the unique data. This is a way to find the correct field id.
  - Be careful of public versus private data in other objects exposed to the debugger. In some cases the objects themselves are private. For example you may see object **nlobjSearchResultCell** when looking through the results of a search. This object is private.

SuiteTalk

- Web Services Usage Log

- Inspect the SOAP request and response xml to determine issues with the construction of your SOAP request message and the processing of the response

- Web Services Process Status
  - This page logs information about asynchronous web service operations, including SOAP request and response xml. Inspect these in the same way as those in the Web Services Usage Log.

SuiteFlow

- View information in the **Active Workflows** and **Workflow History** sublists to determine workflows that executed, what states were executed, and which ones are still active. From the **Active Workflows** sublist you can cancel a workflow if necessary.
- With the ENABLE LOGGING checkbox turned on, you can access the Workflow Log and see details of actions: trigger execution, context, and event type. You can also see the Result of action execution
  - **Executed**.
    The action executed.
  - **Considered**.
    Conditions on the action were evaluated, but returned false, so action did not execute.
  - **Skipped**.
    The action was never considered because it would have been invalid to execute based on the currently running trigger, e.g. an attempt to execute **Send Email** at Before Record Submit. It is not valid to execute **Send Email** at Before Record Submit.
- You can mark an action INACTIVE so it does not execute. This is great while developing a workflow. It allows you to temporarily turn off some actions without having to delete them.

*Identify considerations when working with different environments (e.g. differences in behavior, testing, customizations, configurations, settings, and preferences, etc.) (e.g. production vs. nonproduction, and different accounts)

Sending Emails

Be aware of differences in email behavior between different kinds of accounts. This includes when using `nlapiSendEmail` or **Send Email** action in SuiteFlow:

- Production Account
  - Email sent to recipient

- Sandbox and Release Preview Accounts
  - Follows settings at **Setup > Company > Printing, Fax & Email Preferences > Email subtab > Sandbox and Release Preview section**
- Test Account (TSTDRV, e.g. your training accounts)
  - Emails only sent to **@netsuite.com** email addresses

NetSuite Products

If you are developing SuiteApps, be aware that some of your customers may have NetSuite OneWorld and some may not. NetSuite OneWorld uses Subsidiaries.

Script Behavior

- Sometimes scripts may run differently depending on features that have been enabled (**Setup > Company > Enable Features**)
  - Use `nlobjContext().getFeature()` to check if a feature is enabled.
- Sometimes scripts may run differently depending on preference settings
  - Use `nlobjContext().getPreference()` to check the setting of a particular preference.
- Sometimes scripts may need to run differently based on the environment they are running in
  - Use `nlobjContext().getEnvironment()` to determine whether a script is running in production, sandbox, or release preview.

Deploying to Production

- When a workflow is deployed to production, make sure to set RELEASE STATUS to Released and uncheck ENABLE LOGGING. Also consider adding a CONDITION at workflow initiation to restrict execution to a specific set of users OR user subsidiary, department, class, location or role.
- When a script is deployed to production, make sure to set the script deployment STATUS to Released and LOG LEVEL to Audit or Error. Fill in the Audience subtab as applicable.

*Given a scenario, identify applicable technologies for inbound and outbound integrations, along with their implications

Eight integration methods using a variety of technologies are detailed in the tables below: SuiteTalk, SuiteTalk + User Event Triggers, RESTlet, Custom Record + SuiteTalk/RESTlet, nlapiRequestURL / nlapiRequestURLWithCredentials, SuiteAnalytics Connect, CSV Import, CSV Import + User Event Triggers.

Integration Method	SuiteTalk	SuiteTalk + User Event Triggers
Client	external application	external application
Integration Style	SOAP	SOAP
Data Transport Format	XML (SOAP)	XML (SOAP)
Capabilities	Push data into NetSuite. Pull data out of NetSuite. Synchronous or asynchronous.	Push data into NetSuite. Pull data out of NetSuite. Synchronous or asynchronous.
Scenarios	System to system integrations. Operations that are generally CRUD style. There are other forms of integration better suited for mobile devices and when a large number of records is being pulled out of NetSuite	SuiteTalk scenarios. Additional business logic that can be shared between web services and the user interface, e.g. sending emails.
Client Coding	Any client supporting SOAP, C#, Java, & PHP fully tested with NetSuite. PHP Toolkit when PHP.	Any client supporting SOAP, C#, Java, & PHP fully tested with NetSuite. PHP Toolkit when PHP.
Server Coding	n/a	User Event SuiteScripts and workflows (SuiteFlow) with Before Submit/After Submit event triggers.
Security	Each request requires authentication with NetSuite email, password, and role. SSO login supported.	Each request requires authentication with NetSuite email, password, and role. SSO login supported.
Developer Skills	Client language, concepts of SOAP, WSDL, NetSuite record structure.	Client language, concepts of SOAP, WSDL, NetSuite record structure, SuiteScript.

Integration Method	RESTlet	Custom Record + SuiteTalk/RESTlet
Client	external application	NetSuite (custom record) + external application
Integration Style	REST	SOAP or REST (for external application)
Data Transport Format	JSON, plain text	XML (SOAP) or JSON (for external application)
Capabilities	Push data into NetSuite. Pull data out of NetSuite.	Pull data out of NetSuite.
Scenarios	Many integration use cases, especially good for mobile device integration. CRUD style operations, but also many other things such as sending email, as based on the breadth of functions in the SuiteScript API.	Near real-time integration based on user events. Add ticklers to custom record type containing internal id of record added/updated. Then have follow-up SuiteTalk or RESTlet request to get the data.
Client Coding	Any client supporting http requests.	SuiteScript (custom record) + coding for SuiteTalk/RESTlet
Server Coding	SuiteScript	SuiteScript (RESTlet)
Security	Each request requires authentication with NetSuite email, password, and role. Token-based authentication possible.	Data added to custom record type is made from within authenticated NetSuite session, often the NetSuite user interface. Security for SuiteTalk or RESTlet.
Developer Skills	Client language, REST concepts, SuiteScript, NetSuite record structure	SuiteScript + skills for SuiteTalk/RESTlet

Integration Method	nlapiRequestURL nlapiRequestURLWithCredentials	SuiteAnalytics Connect
Client	NetSuite	external application
Integration Style	REST or SOAP	ODBC, JDBC, ADO.Net
Data Transport Format	JSON, XML (SOAP), other XML, plain text	n/a
Capabilities	Push data out of NetSuite.	Pull data out of NetSuite.
Scenarios	Real-time or near real-time integration based on user events (e.g. adding customer record in NetSuite). Options: ➢ Send all data in request ➢ Send internal id only; have server make follow-up RESTlet or SuiteTalk request to get all the data ➢ `nlapiRequestURL` may have a performance impact, so call a scheduled script from the User Event script to execute `nlapiRequestURL` ➢ Use `nlapiRequestURLWithCredentials` if sending userid/password to external system (makes https request)	High speed throughput on getting data out of NetSuite. Use when extracting large numbers of records out of NetSuite. SuiteAnalytics Connect supports read-only access to your NetSuite data.
Client Coding	SuiteScript	Any client supporting ODBC, JDBC, or ADO.Net drivers, for example Java (JDBC) or C# (ADO.Net)
Server Coding	Varies	n/a
Security	Each request is made from within an authenticated NetSuite session, often the NetSuite user interface, but may also be from a scheduled script. Outbound security when using nlapiRequestURLWithCredentials.	Using database driver requires authentication with NetSuite email, password, and role
Developer Skills	Server language, SuiteScript, REST/SOAP concepts, NetSuite record structure	Client language, SQL, data model (views), NetSuite record structure

Integration Method	CSV Import	CSV Import + User Event Triggers
Client	NetSuite	NetSuite
Integration Style	CSV	CSV
Data Transport Format	CSV files	CSV files
Capabilities	Push data into NetSuite	Push data into NetSuite
Scenarios	Import CSV files. Use NetSuite CSV import tool in the user interface for one-time or infrequent imports. Automate CSV import jobs using SuiteScript. This uses CSV import mapping from the Import tool. `nlapiSubmitCSVImport` is used. Send CSV file to NetSuite via SuiteTalk or RESTlet, then associate with CSV import job. CSV file can be sent with base64 encoding. If RESTlet, the RESTlet can also perform the import. Use scheduled scripts to automate CSV imports.	CSV Import scenarios. Additional business logic that can be executed during addition or update of each record during import, e.g. sending emails.
Client Coding	SuiteScript when import is automated	SuiteScript when import is automated
Server Coding	n/a	User Event SuiteScripts and workflows (SuiteFlow) with Before Submit/After Submit event triggers.
Security	Data import is done from within authenticated NetSuite session, either through user interface or from SuiteScript. Security for SuiteTalk or RESTlet if they are part of the solution.	Data import is done from within authenticated NetSuite session, either through user interface or from SuiteScript. Security for SuiteTalk or RESTlet if they are part of the solution.
Developer Skills	CSV files, NetSuite CSV Import tool, SuiteScript, NetSuite record structure. Skills for SuiteTalk or RESTlet if they are part of the solution.	CSV files, NetSuite CSV Import tool, SuiteScript, NetSuite record structure. Skills for SuiteTalk or RESTlet if they are part of the solution.

Identify how to detect and prevent duplicate records

NetSuite has built-in duplicate detection capabilities in the user interface. You can automate duplicate detection and merge processing through script. And with script you can add your own custom rules to identify when two or more records are duplicates.

SuiteDreams_NoScript_DuplicateDetection.js is an example script automating duplicate detection with a custom rule. Take a look at the script to find more details about duplicate detection in general as well as the automation aspect.

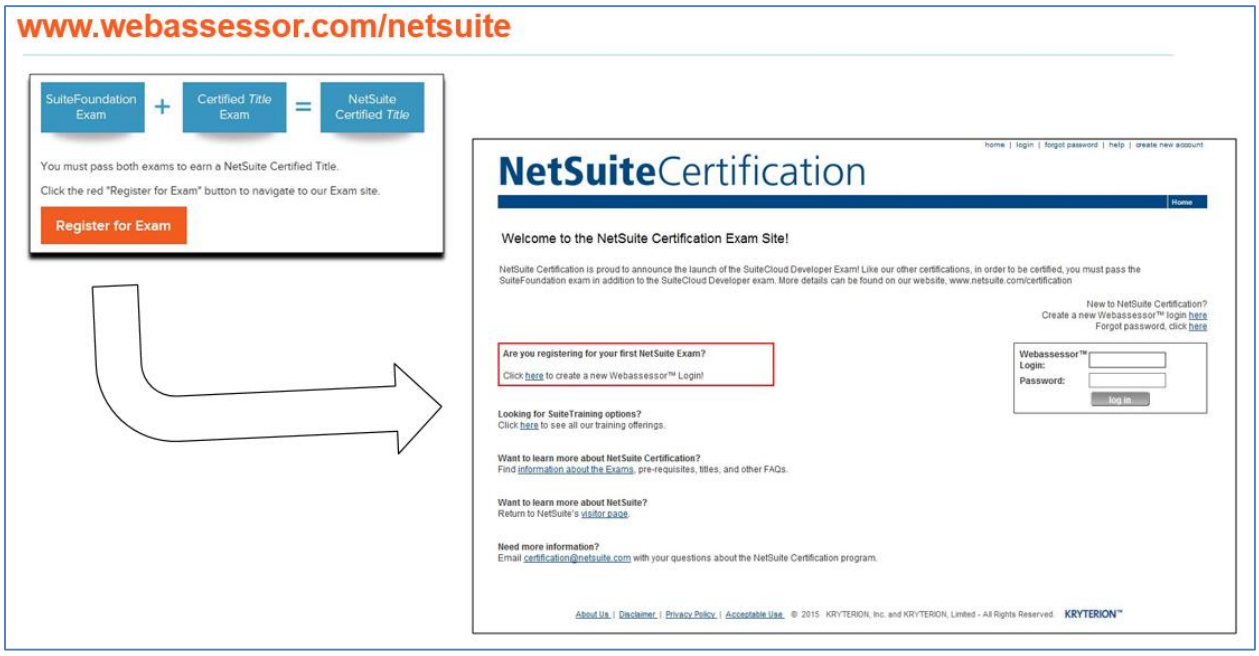
Module 09 | Taking the Exam

MODULE 09 | TAKING THE EXAM

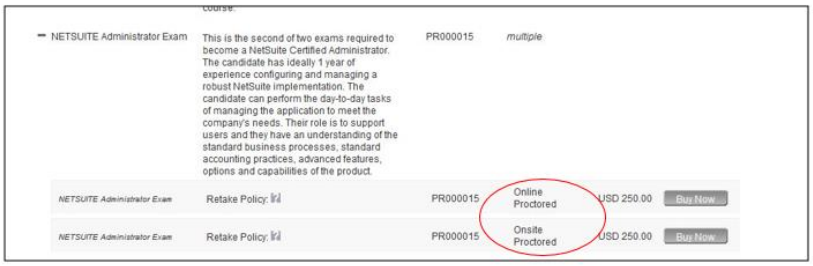
Module Overview

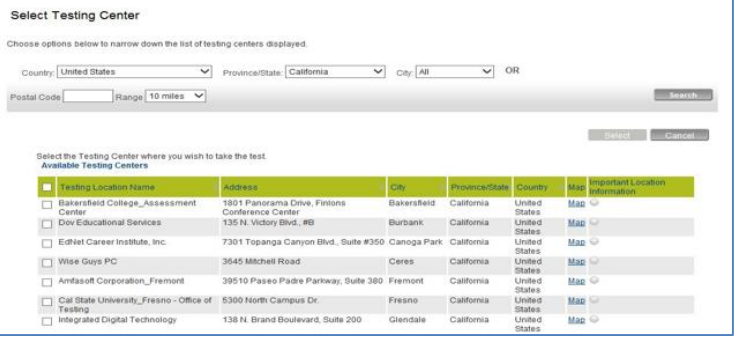
This module reviews some information about taking the SuiteCloud Developer Exam.

Registering for the Exam



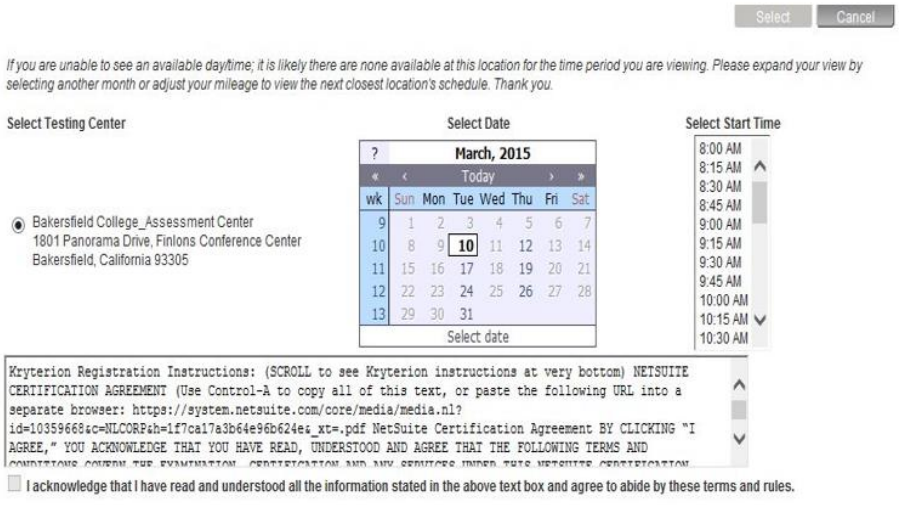
OnSITE vs: OnLINE—Pay Attention!





Select time, Note location, Agree to Certification Agreement

Last chance to confirm the Delivery Type



Online or Onsite Exam?

The Exam Process – Online Examinations

Advantages:

- More scheduling options, evenings, weekends
- Easy to get to
- You control your own environment (less stress?)

Possible Disadvantages:

- Technical setup and software installation required ahead of time
- Separate webcam required
- What is your physical environment like?
- No breaks during the exam

The Exam Process – Online Examinations

When done online, the examination is proctored via an approved Webcam.

The proctor **must** be able to see your keyboard, mouse and hands at all times throughout the examination

- **Built-in cameras are not acceptable** – the camera must be positioned to your side, to see your screen, pointing at your keyboard and mouse
- You must also have a working microphone – the proctor must be able to hear you – if you speak during the examination, your proctor will immediately suspend your exam
- Your desk must be visibly clear of all clutter – notepads, pens, books, tablet computers/phones, etc.
- You cannot take a break during the examination
- No talking, no leaning back out of view, no other people present

The Exam Process – Online Examinations continued

You must prepare your computer – that is, install the examination software (Sentinel) on your PC or laptop – 48 hours prior to sitting the examination

Download the Guide here: http://www.kryteriononline.com/docs/PreparingForYourExam.pdf

- PC or Mac is fine
- Photo submission and typing biometrics required for registration and exam launch
- During the examination, the software will take control of your computer, disabling you from being able to swap to another application during the examination
- This software enables the Proctor to see what is on your screen, a bit like a Webex
- The Sentinel software may be removed at the completion of the examination – but keep it for your next exam?
- Keep the webcam for next time

The Exam Process – In-Person Examinations

Advantages:

- 600 locations around the world
- Controlled setting
- Bathroom break may be allowed

Possible Disadvantages:

- Limited evening/weekend hours
- Must travel to a location
- Less control over the setting
  - Preview it ahead of time or check social media sites like Yelp
- Possible external noise—be prepared to use earplugs
- MUST BRING PHOTO ID AND CONFIRMATION CODE (See email)

The Exam Process – In-Person Examinations continued

During the examination:

- No written reference materials may be used
- You must use the testing facility's computer, not your own laptop
- The computer will be locked into the Examination application session
- All personal belongings must be left with the proctor until the completion of the examination
- You must be signed out by the proctor.

Maintaining your Certification

Maintaining NetSuite Certified SuiteCloud Developer

There will be 2 ongoing requirements for maintaining your NetSuite Certified SuiteCloud Developer title:

- Annual Release Maintenance Quiz
- Refresh Developer Exam

Maintaining Certification: Annual Release Maintenance

NetSuite will publish a once per year "Upgrade Test" to validate awareness of new NetSuite features that have been released in the current year.

- Open book, online exam
- About 30 questions

Information on availability of this Upgrade Test will be posted to the following:

- Emailed out via registration details in the Webassessor testing system
- Included in the Announcements posted to the NetSuite Certification webpage (www.netsuite.com/certification)
- Closed LinkedIn NetSuite Certified SuiteCloud Developer Group

Maintaining Certification: Refresh SuiteCloud Developer Exam

NetSuite will rewrite the SuiteCloud Developer Exam every 3 years.

- NetSuite Certified SuiteCloud Developers will be required to retake and pass this Exam every 3 years to maintain their Certification.

Day of the Exam

ONSITE: Print out your email instructions and EXAM CODE, WITH 2 photo IDs

ONLINE: Check page 4 of the Preparing For Your Exam PDF Guide

Assistance the Day of the Exam:

Our Test Vendor, Kryterion, is available:

- Sunday 6am – Friday Midnight
- Saturday from 6am - Midnight at: (Phoenix Arizona Time, UTC -7)
- During your scheduled Exam time outside the above windows

- **Voice**: 877-313-2008 (U.S.) or 001 602-659-4679 (International)
- **Live Chat**: www.KryterionOnline.com/Support, then click on "LIVE HELP"
- **E-Mail**: OLPsupport@KryterionOnline.com