Sample SuiteCloud Customization Projects

January 15, 2020
should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Master Agreement, Oracle License and Services Agreement, Oracle PartnerNetwork Agreement, Oracle distribution agreement, or other license agreement which has been executed by you and Oracle and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Sample Code

Oracle may provide sample code in SuiteAnswers, the Help Center, User Guides, or elsewhere through help links. All such sample code is provided “as is” and “as available”, for use only with an authorized NetSuite Service account, and is made available as a SuiteCloud Technology subject to the SuiteCloud Terms of Service at www.netsuite.com/tos.

Oracle may modify or remove sample code at any time without notice.

No Excessive Use of the Service

As the Service is a multi-tenant service offering on shared databases, Customer may not use the Service in excess of limits or thresholds that Oracle considers commercially reasonable for the Service. If Oracle reasonably concludes that a Customer's use is excessive and/or will cause immediate or ongoing performance issues for one or more of Oracle's other customers, Oracle may slow down or throttle Customer's excess use until such time that Customer's use stays within reasonable limits. If Customer's particular usage pattern requires a higher limit or threshold, then the Customer should procure a subscription to the Service that accommodates a higher limit and/or threshold that more effectively aligns with the Customer’s actual usage pattern.

Beta Features

Oracle may make available to Customer certain features that are labeled “beta” that are not yet generally available. To use such features, Customer acknowledges and agrees that such beta features are subject to the terms and conditions accepted by Customer upon activation of the feature, or in the absence of such terms, subject to the limitations for the feature described in the User Guide and as follows: The beta feature is a prototype or beta version only and is not error or bug free and Customer agrees that it will use the beta feature carefully and will not use it in any way which might result in any loss, corruption or unauthorized access of or to its or any third party's property or information. Customer must promptly report to Oracle any defects, errors or other problems in beta features to support@netsuite.com or other designated contact for the specific beta feature. Oracle cannot guarantee the continued availability of such beta features and may substantially modify or cease providing such beta features without entitling Customer to any refund, credit, or other compensation. Oracle makes no representations or warranties regarding functionality or use of beta features and Oracle shall have no liability for any lost data, incomplete data, re-run time, inaccurate input, work delay, lost profits or adverse effect on the performance of the Service resulting from the use of beta features. Oracle's standard service levels, warranties and related commitments regarding the Service shall not apply to beta features and they may not be fully supported by Oracle's customer support. These limitations and exclusions shall apply until the date that Oracle at its sole option makes a beta feature generally available to its customers and partners as part of the Service without a “beta” label.
Send Us Your Feedback

We'd like to hear your feedback on this document.

Answering the following questions will help us improve our help content:

- Did you find the information you needed? If not, what was missing?
- Did you find any errors?
- Is the information clear?
- Are the examples correct?
- Do you need more examples?
- What did you like most about this document?

Click here to send us your comments. If possible, please provide a page number or section title to identify the content you're describing.

To report software issues, contact NetSuite Customer Support.
Table of Contents

Sample SuiteCloud Platform Customization Projects Overview ....................................................... 1
Importing a Sample SuiteCloud Platform Project ............................................................................... 1
Project Sample: Golf Courses ........................................................................................................... 3
Project Sample: Map Local Customers ............................................................................................. 3
Project Sample: To Do List ............................................................................................................... 4
Project Sample: Manage Opportunities ............................................................................................. 4
Project Sample: SuiteScript 2.0 Unit Testing ...................................................................................... 5
Sample SuiteCloud Platform Customization Projects Overview

If you are new to the SuiteCloud platform, it may be challenging for you to get started and to determine which SuiteCloud tools to use to customize NetSuite for your organization. For information about each of the SuiteCloud platform customization tools, see the help topic SuiteCloud Platform Customization.

This section includes a collection of sample customizations done with SuiteCloud platform tools. You can review these customization projects to get ideas of how to use the various tools. These samples are in a rough form, but they can help you to learn how different SuiteCloud components interact to provide robust customization capabilities.

The customization objects that make up each sample are represented in file format. You can use NetSuite's SuiteCloud Development Framework (SDF) to download the files for a project. In SDF and SuiteCloud IDE plug-in, you can review the files to get an understanding of the objects that are included and how their files are formatted. You can also use SDF to import the customizations into a NetSuite test account and see how they work. For information, see the help topic Deployment Logs.

The sample project files are in .zip format and can be downloaded from the file cabinet. You can import the .zip file into SuiteCloud IDE plug-in. For instructions on importing a project sample, see Importing a Sample SuiteCloud Platform Project.

See the following project sample descriptions to download the file for that project:

- Project Sample: Golf Courses
- Project Sample: Map Local Customers
- Project Sample: To Do List
- Project Sample: Manage Opportunities
- Project Sample: SuiteScript 2.0 Unit Testing

**Warning:** These sample projects are illustrations that were created in a short time frame, and they are designed to demonstrate customizations that are supported by the SuiteCloud platform. These samples have not been tested, are not production ready, and should not be used in your production account. If you import them into SuiteCloud IDE plug-in, you may encounter validation warnings or other issues. You should not report any issues you find with these samples to NetSuite Customer Support. If you have feedback about these samples, enter your comments in the feedback area at the bottom of this page.

Importing a Sample SuiteCloud Platform Project

The following procedure describes how to download sample project files so that you can use SDF and SuiteCloud IDE plug-in to review project contents. For information about IDE plug-in and how to set it up, see the following:

- Using IDE plug-in for Eclipse, see the help topic SuiteCloud IDE Plug-in for Eclipse Overview.
- Using IDE plug-in for WebStorm, see the help topic SuiteCloud IDE Plug-in for WebStorm Overview.

You can also import the project components using SuiteCloud CLI. For information, see the following:

- Installing CLI for Java
- _SDF File Structure_
To download and view a sample SDF project with SDF and SuiteCloud IDE Plug-in:

1. Download the .zip file. See the project sample description for a link to download the file.
   - Project Sample: Golf Courses
   - Project Sample: Map Local Customers
   - Project Sample: To Do List
   - Project Sample: Manage Opportunities
   - Project Sample: SuiteScript 2.0 Unit Testing
2. In Eclipse, click File > Import.
3. Select General > Existing Projects into Workspace and click Next.
4. Click Select archive file, then click Browse to navigate to the location on your computer where you have saved the .zip file.
   After you select the .zip file, the project appears in the Projects list.
5. Click Finish. Eclipse imports the .zip file.

All of the script files and objects are imported into the project and can be edited in SuiteCloud IDE plug-in.

To use SDF to deploy the project to a NetSuite account:

1. Right-click the project folder and select NetSuite > Add Dependency References to Manifest.
2. Right-click the project folder and select Validate Project against Account. Ensure that there are no errors before proceeding.

3. Right-click the project folder and select Deploy to Account. Select the appropriate account and role, and then click Deploy.

SDF validates and deploys the SDF project to your NetSuite account. The installation log is displayed in the Console subtab.

Project Sample: Golf Courses

Project file name: com.example.golfschmoozer.zip

Scenario

You are a sales representative, and you take your clients golfing to generate business for your company. You want a way to track which courses your clients prefer.

Functionality Overview

Custom record type: Golf Course
Script types used: Suitelet, Client, User Event
Other customization used: Custom lists

A custom center subtab called Golf is available to enter information in the golf course record.

The golf course record instances can be reviewed on the list page available using the custom center category Golf > Courses > Full List.

When creating or editing golf course record instances, a client script uses the Google Maps API to add the course latitude and longitude based on the address entered.

A custom entity field called Favorite Golf Course on the contact record is used to add the customer's favorite golf course. When editing a contact, click the Set Favorite Golf Course button to select the contact's favorite golf course.

A custom center category available at Golf > Sync > New Courses provides a sync operation that prompts the user for a state. After a state is selected, the Golf World API is used to check for golf courses in that state. If there are any new golf courses, information about the club, including the club logo, is added to the golf course list in NetSuite.

Project Sample: Map Local Customers

Project file name: com.example.customermap.zip

Scenario

You are a sales representative visiting a new city and you want to schedule visits with local customers.
Functionality Overview

Script types used: Suitelet
Other customizations used: saved search
Use the Customer Search Form to run a saved search that enables you to search for customers by city or zip code.
Google Maps Geocoding API is used to change the address to a geocode location.
The location of customers is shown on a Google Maps page. Red markers indicate customer with higher revenue. Black markers indicate a customer with lower revenue. When a marker is clicked, the customer name appears.

Project Sample: To Do List

Project file name: com.example.todoitem.zip

Scenario

You are a NetSuite user who regularly enters tasks, and you want a To Do portlet on your home page.

Functionality Overview

Custom record types: To-Do Item and Dependency
Script types used: Portlet, Suitelet, Client, User Event
Other customizations: Custom lists, saved search
A To-Do List portlet is available to add to each user's home page. The portlet displays the items, their owners, statuses, and priority. The To-Do List portlet is created using a portlet script. A suitelet populates the portlet with all to-do items.
A list of To-Do Items is available by going to Lists > Custom > To-Do Item. You can create to-do items and assign them to yourself or other employees. Additionally, you can create dependencies between to-do items. For example, you might create a to-do item called Release Cool Product, which is dependent on other to-do items named Build Product, Test Product, and Document Product. You list each dependency on the parent record's Dependencies sublist. When a user tries to mark a to-do item complete in the portlet, a client script ensures that any dependent records have already been completed.
The task record is integrated with the to-do item record. Each time a task record is created, a corresponding to-do item is automatically created and added to the portlet, so that all work can be tracked in one place.
This project could be enhanced so that the creation of other records, in addition to task records, causes the automatic creation of corresponding to-do items.

Project Sample: Manage Opportunities

Project file name: SuiteScriptBeginner.zip
Scenario

Sales representatives want to manage leads and opportunities and follow up on expected closing dates.

⚠️ Important: This project does not contain all of the project files that are required to import the project into SuiteCloud IDE and deploy it. This project is designed to demonstrate scripting approaches for different script types. Examine the individual project files and read the descriptions below to learn more.

Functionality Overview

This example provides sales representatives with a method of capturing lead information.

Script types used: Suitelet, Client, User Event, Scheduled Script

On the Manage Opportunities form, sales representatives can enter lead and opportunity information. The form is validated, and a message appears to indicate whether the new record was created successfully. When a new record is created, a sublist is shown of the Top Items across Orders by Monetary Value.

When a new opportunity record is saved, the expected close date is set to 14 days from the current date. In addition, a scheduled script is run to create a task to follow up on the opportunity.

Project Sample: SuiteScript 2.0 Unit Testing

Project file name: com.example.unittesting.zip

Scenario

You are a SuiteScript developer, and you have created several scripts of various types. You want to write unit tests against these scripts.

This sample project is an example of how to perform unit testing with SuiteScript 2.0. The project uses the Karma test runner and the Jasmine testing framework, but there are other tools that you can use for unit testing. The project is based on a presentation that was delivered at SuiteWorld 2018.

⚠️ Important: Karma and Jasmine are third-party tools that are not part of NetSuite. For full documentation on these tools, see https://karma-runner.github.io/2.0/index.html and https://jasmine.github.io/.

Note: Karma and Jasmine are third-party tools that are not part of NetSuite. For full documentation on these tools, see https://karma-runner.github.io/2.0/index.html and https://jasmine.github.io/.

Project Sample: SuiteScript 2.0 Unit Testing

Project file name: com.example.unittesting.zip

Scenario

You are a SuiteScript developer, and you have created several scripts of various types. You want to write unit tests against these scripts.

This sample project is an example of how to perform unit testing with SuiteScript 2.0. The project uses the Karma test runner and the Jasmine testing framework, but there are other tools that you can use for unit testing. The project is based on a presentation that was delivered at SuiteWorld 2018.

⚠️ Important: This project does not contain all of the project files that are required to import the project into SuiteCloud IDE and deploy it. This project is designed to demonstrate unit testing approaches for different script types. Examine the individual project files and read the descriptions below to learn more.

Functionality Overview

Script types used: Map/Reduce, Suitelet, RESTlet

This sample project includes two script files for each script type used:
A main script that demonstrates the use of the script type. For example, the map_reduce_example.js script file contains a simple map/reduce script. These scripts are located in the /FileCabinet/SuiteApps/com.example.unittesting/src folder in the project.

A spec script that tests different aspects of the main script. For example, the map_reduce_example_Spec.js script file contains tests for the main map_reduce_example.js script. These scripts are located in the /FileCabinet/SuiteApps/com.example.unittesting/test/jasmine/unit/spec folder in the project.

The project also includes mock objects that simulate parts of the SuiteScript API. These mock objects let you run the project tests without connecting to a NetSuite account. Mock objects are included for the N/log and N/record modules (log.js and record.js, respectively), and these objects are located in the /FileCabinet/SuiteApps/com.example.unittesting/test/jasmine/unit/mock/N folder in the project.

Map/Reduce Example

The map/reduce spec script (map_reduce_example_Spec.js) creates objects that correspond to mock search results, and it calls the map() and reduce() functions of the map/reduce main script (map_reduce_example.js).

For the map() function, the spec expects the key of the resulting pairs to be the entity field of the provided object, which is equal to 30. For the reduce() function, an expected refund is calculated based on the provided formula for refunds. A record is loaded with the record ID returned by the reduce() function and a type of CASH_REFUND. This record is expected to exist, and its entity field is expected to have a value of 30. The sublist data of the record is fetched, and the actual total refund is calculated and expected to equal the calculated expected refund.

Suitelet Example

The Suitelet spec script (suitelet_example_Spec.js) creates spies to replace various functions in the N/record module. It checks to see whether the Suitelet main script (suitelet_example.js) calls these functions with the expected arguments when the onRequest entry point is triggered.

A mock context object is created with a mock favorite golf course and contact ID. This object’s response attribute has its write attribute replaced with an empty spy. A mock record object is also created. This object’s setValue and save attributes are replaced with empty spies, and its load attribute is replaced with a spy that returns the object. The spec script triggers the onRequest entry point and verifies that the spies have all been called in the Suitelet with their expected arguments.

RESTlet Example

The RESTlet main script (restlet_example.js) takes a context object containing a record type attribute, some number of body field attributes (none of which can be arrays), and some number of sublist attributes (some of which are arrays). At the post entry point, the script creates a record with the specified type and sets its body and sublist fields to the provided values. It saves the record and returns the record ID.

The RESTlet spec script (restlet_example_Spec.js) creates a context object and provides it to the RESTlet main script by calling its post entry point. The spec script loads the record that the main script created and saved, and it verifies that the sublist fields equal their expected values.

Installation and Configuration

The following sections describe how to install and configure the tools that are required for this sample project. Make sure that you install and configure all of these tools before you start exploring the project.
Karma


In your project folder, run the following commands at a command prompt to install Karma:

```
npm install karma --save-dev
npm install karma-jasmine karma-chrome-launcher jasmine-core --save-dev
```

To install the Karma command-line interface, which lets you run Karma from any folder location, run the following command:

```
npm install -g karma-cli
```

For full installation instructions, see https://karma-runner.github.io/2.0/intro/installation.html.

Karma Configuration File

This sample project includes a pre-generated configuration file called karma.conf.js, but you can generate your own Karma configuration file. To do so, run the following command at a command prompt:

```
karma init my.conf.js
```

You will be asked a series of questions about how you will use Karma, and a .js configuration file is generated based on your responses. To start Karma with this configuration file, run the following command:

```
karma start [config file].js
```

You can override some configuration options by typing `--[option name]` after the command above. To see a list of available options, run the following command:

```
karma start --help
```

For full configuration instructions, see https://karma-runner.github.io/2.0/intro/configuration.html.

The Karma configuration file sets `module.exports` to a function that takes a configuration object, and it calls `config.set()` to set parameter values. This function has the following form:

```
config.set({[parameter1]: [option1], [parameter2]: [option2], ...})
```

Each configuration option corresponds to a question that is asked when you run the `karma init` command. The `files` option is particularly notable. It is set to an array whose elements correspond to files and their usages. Elements of the array can be filepath strings, optionally with aggregators such as *. Elements can also be JavaScript objects that map the `pattern` attribute to the filepath options according to how the file will be used (for example, the file type, whether to watch the file, and so on). For more information, see https://karma-runner.github.io/2.0/config/files.html.

The `browsers` option sets which browsers are to be used. To install the launchers as plugins, run the following command:

```
npm install karma-[plugin name] --save-dev
```

You can also put the plugin names as strings in the array that corresponds to the `plugins` option. For more information, see https://karma-runner.github.io/2.0/config/browsers.html.
Jasmine

For Karma-Jasmine integration, some useful non-browser plugins are karma-jasmine, karma-requirejs, karma-coverage, and karma-spec-reporter. These plugins are required for this sample project. To install them, run the following command:

```
npm install [plugin name]
```

When you start Karma, it runs your Jasmine tests and reports on their success or failure. For a guide to writing Jasmine tests, see https://jasmine.github.io/tutorials/your_first_suite.

RequireJS

RequireJS is required for this sample project. To install it, run the following command in the same folder as Karma and the plugins that are listed above:

```
npm install requirejs
```